Robotic Soccer

Robotic Soccer

Edited by
Pedro Lima

I-Tech

Published by I-Tech Education and Publishing

I-Tech Education and Publishing
Vienna
Austria

Abstracting and non-profit use of the material is permitted with credit to the source. Statements and opinions expressed in the chapters are these of the individual contributors and not necessarily those of the editors or publisher. No responsibility is accepted for the accuracy of information contained in the published articles. Publisher assumes no responsibility liability for any damage or injury to persons or property arising out of the use of any materials, instructions, methods or ideas contained inside. After this work has been published by the I-Tech Education and Publishing, authors have the right to republish it, in whole or part, in any publication of which they are an author or editor, and the make other personal use of the work.

© 2007 I-Tech Education and Publishing
www.i-techonline.com
Additional copies can be obtained from:
publication@ars-journal.com

First published December 2007
Printed in Croatia

A catalogue record for this book is available from the Austrian Library.
Robotic Soccer, Edited by Pedro Lima
p. cm.
ISBN 978-3-902613-07-3
1. Robotic Soccer. 2. Multi-Robot. I. Pedro Lima

Preface

Soccer Robotics is nowadays a vibrant field where a huge number of people carry out many of their research and education activities in Robotics worldwide. The reason for such an excitement is rooted on the diversity of scientific and technical challenges raised by the problem of developing a team of cooperating robots working together to defeat an opponent team, in a very dynamic and uncertain environment.

Two major competitions - RoboCup and MiroSot - have evolved in recent years towards becoming major scientific events where Artificial Intelligence and Robotics researchers meet annually to compare their latest results, as well as to show them to general audiences and to stimulate young kids to learn Robotics by designing, developing and testing their own robots. Most of the papers in this book are authored by researchers participating regularly in such events, which include real and simulated robot competitions posing different challenges – typically, real robots challenges concern mobility, (cooperative) perception and localization, while simulated robots are the perfect environment for testing teamwork, learning and coordination strategies.

If I were asked to use a single word to define the most relevant scientific challenge brought up by Soccer Robotics, my preference would go to integration. Current and past research in Robotics tends to privilege particular subsystems of a (team of) robot(s), such as perception, navigation, or coordination, to name but a few. Though the results of such research endeavors are certainly relevant, they miss a comprehensive view of the overall system, whose goal is to optimize the whole, instead of each of its composing parts. As a consequence, and not surprisingly, some of the papers in this book propose development environments, architectures, middleware and realistic simulators, to handle the integration of robot teams, whose application is not limited to robot soccer.

Nevertheless, many papers in the book concern advanced research on (multi-)robot subsystems, naturally motivated by the challenges posed by robot soccer, but certainly applicable to other domains: reasoning, multi-criteria decision-making, behavior and team coordination, cooperative perception, localization, mobility systems (namely omni-directional wheeled motion, as well as quadruped and biped locomotion, all strongly developed within RoboCup), and even a couple of papers on a topic apparently solved before Soccer Robotics – color segmentation – but for which several new algorithms were introduced since the mid-nineties by researchers on the field, to solve dynamic illumination and fast color segmentation problems, among others.

The number of new research opportunities fostered by Soccer Robotics keeps growing everyday. Two of the papers in the book deal with non-soccer applications of great relevance. RoboCup@Home is a competition recently introduced in RoboCup, "with the aim to foster the development of applications in the domains of service and assistance robotics, ambient intelligence and human-robot interaction". And another paper proposes a new sports robotic competition - Water Polo – whose goal is to extend the research-advances-through-competitions concept to Underwater Robots.

This book is certainly a small sample of the research activity on Soccer Robotics going on around the globe as you read it, but it surely covers a good deal of what has been done in the field recently, and as such it works as a valuable source for researchers interested in the involved subjects, whether they are currently "soccer roboticists" or not.

Editor
Pedro U. Lima
Institute for Systems and Robotics
Instituto Superior Técnico
Lisboa, PORTUGAL
E-mail: pal@isr.ist.utl.pt

Contents

Preface V

1. Communication and Collaboration in Heterogeneous Teams of Soccer Robots 001
Philipp A. Baer and Roland Reichle

2. Positioning in Robots Soccer 029
Hesam T. Dashti, Shahin Kamali and Nima Aghaeepour

3. Non-monotonic Reasoning on Board a Sony AIBO 045
David Billington, Vladimir Estivill-Castro, René Hexel and Andrew Rock

4. Color Classification and Object Recognition for Robot Soccer Under Variable Illumination 071
Nathan Lovell and Vladimir Estivill-Castro

5. Towards Model-based Vision Systems for Robot Soccer Teams 095
Murilo Fernandes Martins, Flavio Tonidandel and Reinaldo A. C. Bianchi

6. Probabilistic and Statistical Layered Approach for High-Level Decision Making in Soccer Simulation Robotics 109
Carlos Bustamante and Leonardo Garrido

7. Simulated Environment in Robot Soccer 135
Gregor Klancar and Rihard Karba

8. A Robust and Scalable Pareto Optimal Ball Passing Algorithm for the Robotic Soccer 153
Vadim Kyrylov

9. FC Portugal - High-level Coordination Methodologies in Soccer Robotics 167
Nuno Lau and Luis Paulo Reis

10. Desktop Robot Soccer 193
Frederic Maire, Joaquin Sitte and Narongdech Keeratipranon

11. Embedded Behavioral Control of Four-legged Robots 203
David Herrero Pérez and Humberto Martínez Barberá

12. A Comprehensive Framework for Perception in Robotic Soccer 227
Luciano Oliveira, Augusto Loureiro and Leizer Schnitiman

13. Task Management for Soft Real-Time Applications Based on General Purpose Operating Systems 243
Paulo Pedreiras and Luís Almeida

14. Integrating Autonomous Behaviour and Team Coordination into an Embedded Architecture 253
Bernd Kleinjohann, Lisa Kleinjohann, Willi Richert and Claudius Stern

15. Learning to Play Soccer with the SimpleSoccer Robot Soccer Simulator 281
Jeff Riley

16. Analysing the Difficulty of Learning Goal-Scoring Behaviour for Robot Soccer 307
Jeff Riley and Vic Ciesielski

17. Motion Detection and Object Tracking for an AIBO Robot Soccer Player 337
Javier Ruiz-del-Solar and Paul A. Vallejos

18. Comprehensive Omni-Directional Soccer Player Robots 347
Mehdi DaneshPanah, Amir Abdollahi, Hossein Ostadi and Hooman Aghaebrahimi Samani

19. Event-driven Hybrid Classifier Systems and Online Learning for Soccer Game Strategies 375
Yuji Sato

20. Robust and Accurate Detection of Object Orientation and ID without Color Segmentation 395
Hironobu Fujiyoshi, Tomoyuki Nagahashi and Shoichi Shimizu

21. Behavior Acquisition in RoboCup Middle Size League Domain 407
Yasutake Takahashi and Minoru Asada

22. Multicriterial Decision-making Control of the Robot Soccer Team 421
Petr Tucnik

23. Robust and Efficient Robot Vision Through Sampling 447
Alex North and William Uther

24. Robot Localisation Using a Distributed Multi-Modal Kalman Filter 473
Oleg Sushkov and William Uther

25. Impossibles: A Fully Autonomous Four-legged Robot Soccer Team 493
Hamid Reza Vaezi Joze, Jafar Habibi and Nima Asadi

26. RoboCup@Home: Creating and Benchmarking Tomorrows Service Robot Applications 521
Tijn van der Zant and Thomas Wisspeintner

27. VolksBot - A Construction Kit for Multi-purpose Robot Prototyping 529
Thomas Wisspeintner and Walter Nowak

28. Collaborative Localization and Gait Optimization of SharPKUngfu Team **549**
Qining Wang, Chunxia Rong, Guangming Xie and Long Wang

29. The Robotic Water Polo and Underwater Robot Cooperation Involved in the Game **575**
Zhang Lee, Guangming Xie, Dandan Zhang and Jinyan Shao

1

Communication and Collaboration in Heterogeneous Teams of Soccer Robots

Philipp A. Baer and Roland Reichle
University of Kassel
Germany

1. Introduction

The RoboCup tournaments foster research in the area of autonomous robotics and cooperative behaviour. Recently, modifications to the rules were adopted promoting further developments towards a typical human playing ground. Some simplifications such as constant lighting were dropped and further modifications will follow in the next years. Regarding team cooperation and coordination, the most important change in 2007 is the enlargement of the playing field. The maximum number of players in a team has been increased to 6; for the long time goal the number of players will approach 11.
For research groups it may be difficult to keep up with the enlargement of team sizes, for newcomers it even constitutes a virtually infeasible financial effort. This is why so-called mixed teams gain a lot of popularity. Here, two or more research groups pool their resources together to provide a joint, more powerful team (Nardi et al., 1999; Castelpietra et al., 2000). This implies that different hardware and software systems have to communicate and collaborate. A number of problems have to be faced which arise from the heterogeneity of the systems involved. Among other things, the interpretation of different representations, the fusion of information to a consistent world view, and the realization of team-play strategies on the different platforms are predominant questions.
In order to cope with these challenges, we have adopted a model-driven software development approach. Below we introduce our development environment for communication infrastructures. Afterwards, we summarize our research activities towards a model-driven development approach for modelling cooperative behaviour in teams of autonomous soccer robots. A detailed example describes the creation of a software infrastructure for a mixed-team of soccer robots. It illustrates the benefits of our development environment and highlights our contribution. We conclude with a presentation of our vision for further developments.

2. Problem Description

When RoboCup was announced in 1995, it was a research challenge to build autonomous mobile robots (AMRs) that were able to find the ball and the goals, to avoid collisions with other players, to estimate their position on the field, and to score goals. Nowadays, a large

number of approaches are available for solving these problems. The research focus therefore shifted towards creating teams of robots that cooperatively play soccer.
To realize a team-play, robots must be able to exchange information with their team-mates, interpret the exchanged data, and fuse the information to a consistent world view, as already outlined in the introduction. This is the basis for coming to an agreement about the current situation on the field and coordinating the cooperative behaviour of the team. Nowadays, almost every team in the RoboCup middle-size league implements some kind of team-play, which in most cases is tailored to the capabilities and the needs of the underlying robotic software framework. Due to the lack of standard software and because of the variety of different software frameworks, heterogeneity issues play a decisive role when forming a mixed-team.

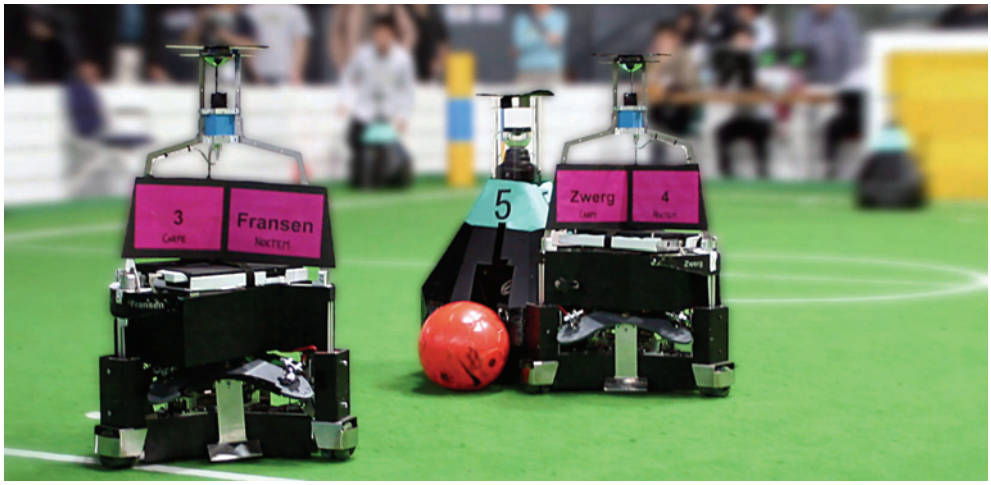


Fig. 1. Carpe Noctem fighting against another team

During the RoboCup World Championships 2006 in Bremen, Germany, the teams Carpe Noctem from Kassel University – shown in Fig. 1 – and the Ulm Sparrows from Ulm University formed a mixed-team. The Ulm Sparrows use Miro (Utz et al., 2002), a middleware framework that is implemented in C++ and heavily relies on CORBA. Greater parts of the Carpe Noctem software framework are realized in C# using the Mono (http://www.mono-project.org/) framework. To set up team cooperation a suitable communication infrastructure had to be established first. The Ulm Sparrows relied on an IP multicast-based group communication scheme over which the SharedBelief (Utz et al., 2004; Isik et al., 2007) data structure was exchanged. In order to talk to the other team, Carpe Noctem thus needed to provide a corresponding implementation along with suitable data conversion techniques. The implementation as such was a quite time consuming task.
For real interoperability it is not sufficient to only communicate data, they also have to be interpreted with regard to their semantics and representations. Teams have to either agree on a standard representation which includes common measurement units and coordinate systems or provide appropriate data conversion routines. However, recent discussions in the RoboCup community show that it is quite difficult to reach an agreement on a standard

representation, as almost any team provides a self-defined representation scheme. Fortunately, the measurements units and the coordinate systems used by the Ulm Sparrows and Carpe Noctem were quite similar.
To be able to agree on the current situation on the field and to realize dynamic role assignment, Carpe Noctem provided a cooperative world model named *SharedWorld*. It fused the exchanged information to a consistent world view and provided realizations of different team-play strategies which were used as a basis for the role assignment. *SharedWorld* was also capable of calculating the ball position by taking into account the trustworthiness and impreciseness of observations, gathered from the different robots. For the Ulm Sparrows a module with nearly the same functionality was created because decisions had to be taken consistently among all players. A re-implementation of *SharedWorld* was necessary because of the incompatible software frameworks of the two teams.
This example makes it quite obvious that the realization of cooperative behaviour in heterogeneous teams of robots is a challenging and time consuming task. This problem has even more effect in groups consisting of more than two teams. Therefore, methods and development support is needed to ease the realization of communication and collaboration in heterogeneous teams of soccer robots.

3. Our Contribution

The previous section showed that the realization of team-play strategies in heterogeneous teams of soccer robots is a quite time consuming task. In order to reduce the development effort we present SPICA, a development framework for communication and collaboration infrastructures in teams of AMRs. SPICA assists in integrating software systems realized in different programming languages and developed for different platforms in heterogeneous distributed environments. It further provides patterns for data and sensor fusion and facilitates the development of cooperative behaviour in groups of AMRs.
To be able to cope with heterogeneity issues, we have adopted a model-driven development approach for SPICA. It supports the specification of communication and collaboration infrastructures of AMRs at an abstract and platform-independent level. Models are then automatically transformed to platform-specific source code which can easily be integrated into existing software frameworks. The modelling support is based on the SPICA modelling language which consists of several domain-specific sublanguages tailored to the different aspects of communication and collaboration infrastructures. The total of all sublanguages form the SPICA modelling language, also referred to as the *Abstract Architecture Specification* (AAS) language.
With the *Message Description Language* (MDL) a developer may specify the structure of network messages in an efficient and platform-independent manner, similar to ASN.1. Communication among AMRs is mostly event-based, so we decided to apply concepts of *message-oriented middleware* (MOM) architectures as they turned out to be most appropriate.
The *Data flow Description Language* (DFDL) supports the specification of communication infrastructures in terms of modules and the data flow between them. Module stubs are created from the specifications which are basically adapters to the underlying communication infrastructure. The DFDL also facilitates the specification of the data management behaviour. For this purpose, it provides so-called *Data Management Containers*

(DMCs) which are data structures used for managing incoming and outgoing data. DMCs further build the foundation of the general purpose *Data-Analysis Description Language* (DADL). It provides modelling support for filters that operate directly on the contents of the DMCs. DADL comprises a Matlab-like syntax allowing calculations on the exchanged data to be specified in a platform-independent manner. Examples are the calculation of a robot's role or the agreed ball position. In addition, the DADL also provides some predefined filter patterns to fuse the exchanged data to a consistent world view. The integration of other services such as data encryption or authentication is possible as well. Apart from these sublanguages we employ the concept of ontologies. They allow us to establish a common understanding of the involved semantic concepts and different representations and therefore help to realize automatic conversion of data representations.

With the help of the tools provided by the SPICA development environment, the resulting *platform-independent models* (PIMs) of the communication and collaboration infrastructure can be transformed into platform-specific implementations in C#, C++, and Java. Our template-based approach allows for easy integration of further programming languages.

The model-driven development approach proved to significantly reduce the development effort for the realization of communication and collaboration infrastructures for heterogeneous teams of AMRs. A suitable communication and collaboration infrastructure has to be developed only once by specifying the desired functionality in a platform-independent manner. The corresponding platform-specific implementations are generated automatically and can be integrated into new or existing software frameworks very easily. In addition, the SPICA development framework also completely relieves the developers from the burden of dealing with encoding and decoding issues, heterogeneous data representations, and synchronization issues. In this aspect, the SPICA-based implementations are comparable to Remote Procedure Call-based (RPC) solutions. The main difference here is that generated implementations are tailored to the characteristics of event-driven AMR group communication. Using SPICA, the developers furthermore do not have to deal with the time-consuming implementation of data fusion and analysis schemes for each of the involved platforms.

In the following, we will introduce the SPICA development environment in more detail along with its modelling language and associated capabilities. Afterwards, an elaborate example will outline the steps required to specify a communication and collaboration infrastructure between two different groups of AMRs.

4. The Spica Approach

The concept of the SPICA development environment was first published in 2007 (Baer et al., 2007). The first generation of SPICA was capable of generating message structures, the second generation added support for modelling data flow. The third generation we outline here, brings major language cleanups, enhancements, and new features such as dynamic module binding, semantic annotation, and automatic data conversion.

Dynamic module binding relies on a service discovery engine embedded into the generated implementation. The creation of channels is based on the availability of resources. It is also possible to create static channels which do not require the service discovery engine.

Automatic data conversion relies on the semantic annotation of the specified data structures. Here, a common understanding of the semantics of data structures and the relations

between them may be established using an ontology specification. Our framework also foresees the integration of ontologies provided by third parties.
For several reasons we decided to develop a textual domain-specific modelling language instead of using existing general purpose ones. First of all, the development of a novel *domain-specific language* (DSL) enables us to provide a very compact modelling notation, reduced to the needs and tailored to the semantics of our modelling domain. The SPICA sublanguages cover these areas where more specific modelling support is required. They are designed in such a way that they combine to the overall SPICA modelling language in a consistent fashion. This is not only advantageous for the model transformation process but also for the developer, who does not have to deal with different semantics of different description languages. A textual notation is furthermore sufficient for most modelling tasks. If designed with simplicity in mind, it is often even more convenient to use than graphical notations and it may allow for rapid prototyping.

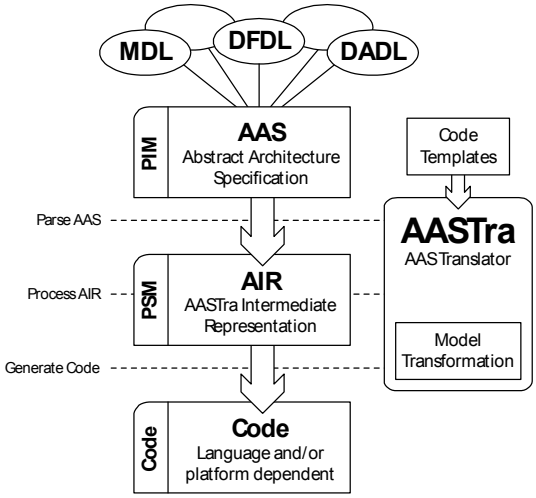


Fig. 2. Internal workflow of the SPICA development environment

DSLs, of course, require us to create tools that interpret and transform models into concrete implementations. The corresponding tool developed for SPICA is the *AAS Transformator* (AASTra), covering the whole process from interpreting a model down to generating concrete source code. It follows a three-layered approach as shown in Fig. 2, representing the reduction of abstraction from the topmost down to the lowest level of modelling. The SPICA modelling language resides on the topmost layer named AAS; it represents the PIM in the context of Model-driven Development (MDD). On the second layer, an in-memory intermediate representation of the AAS models is generated which is referred to as the *AASTra Intermediate Representation* (AIR). Processing is carried out in two steps: First, the model is parsed and references are resolved. Afterwards, the model is processed and transformed into a representation more suitable for the final code transformation. Here, the consistency of the model is checked and the required non-trivial transformations are performed. Language compilers follow a very similar approach when transforming a language specification to assembler or executable binaries.

The actual code transformation is performed on the third layer. Here, a template engine keeps the transformation process very flexible and customizable. Templates can access to the AIR directly. Reoccurring patterns are encapsulated in smaller sub-templates. To reduce redundancy in the code, frequently used functionality is relocated to libraries.
We will now introduce the SPICA modelling language along with all its sublanguages, starting with a description of common language features.

4.1 Common Model Semantics

4.1.1 Blocks

Related statements are grouped together in logical blocks. Each such block has a type and a name. The body of the block is enclosed in curly braces. The example below outlines the general structure.

```
<type> <name> [<inheritance spec>] [<annotations>] {
        <body>
}
```

The type of such a block is given by one of the predefined keywords `header`, `message`, `container`, `coord`, or `module`. The first four keywords are belonging to MDL while the remaining one is part of DFDL. A type is followed by a name, an inheritance specification, and additional annotations. The optional inheritance specification is available in the MDL only. Annotations are lists of key-value pairs enclosed in square brackets; values may be omitted. They parameterize the respective elements and thus influence the code generation process. Annotations are supported in every sublanguage but are optional as well.

4.1.2 Semantic Model

In a heterogeneous distributed environment, where a-priori unknown systems have to communicate and interpret the exchanged data, it is essential to establish a common understanding of their semantics and representations. Therefore, the SPICA AAS language supports semantic annotation of data structures. The semantic model provides two types of classes: concepts and representations. A concept defines the conceptual appearance of an element while a representation defines its concrete representation. A base ontology defines fundamental concepts like ball or player, but also coordinate systems and representations such as physical units. It thus builds the foundation for an automatic conversion of representations, a conversion from mm to m, for instance. Due to the availability of coordinate system specifications defined with regard to a reference system also non-trivial operations like converting the representation from one coordinate system to another can be provided automatically. For even more complex tasks we also allow the definition of custom conversion methods in a way similar to (Strang et al., 2003).
Semantic descriptions provided by third-parties may be used as well. The base ontology and third-party additions are managed by a simple, distributed storage system which supports retrieval or insertion of ontology classes in a lightweight manner.
Two semantic annotations are created for all blocks in a SPICA model automatically if not specified explicitly: `concept` specifies the ontology class name while `rep` specifies the representation of the element. To reference a block type, its concept, or representation, the

`reftype`, `refconcept`, and `refrep` annotations are provided. The type reference replaces the other two. References to coordinate system are modelled using the `refcoord` annotation.
Ontology classes such as concepts or representations are referenced using URNs. Unique URNs are assigned to elements implicitly, providing a reference name to their concept and representation. In order to allow for a compact specification the developer can use abbreviations to URN in terms of prefixes. The default prefix # references the SPICA URN namespace `urn:spica`.

4.1.3 Variants

Block variant identifiers have been introduced because several blocks with the same name may be defined. Variant identifiers are arbitrary strings representing the target AMR platform. They are appended to the block name, wrapped into angle brackets. If there is more than one target platform, it is possible to specify multiple variant identifiers delimited by colons.
The existence of variant identifiers changes the automatic generation of concept and representation URNs for blocks. The following concept and representation URNs are created by default if no variant names are given:

```
urn:spica:<type>:concept:<name>
urn:spica:<type>:rep:<name>
```

If a variant name is provided, the URNs read as follows:

```
urn:spica:<variant name>:<type>:concept:<name>
urn:spica:<variant name>:<type>:rep:<name>
```

4.2 Message Specification

As already outlined above, the concept of message-oriented communication is well suited for AMRs. This has several reasons: The network infrastructure of mobile autonomous system resembles the characteristics of mobile ad-hoc networks, so communication links are likely to exist only for a limited period of time. This is why a message-oriented and connection-less communication scheme has clear advantages over a connection-oriented one. Another reason stems from the communication behaviour of AMRs. As environmental monitoring and sensing are known to be mostly event-driven, message-oriented communication here directly reflects their characteristics. Messages may further get lost during transmission where a dependency to previous messages may average the usefulness of information.
Based on these observations we created the MDL as a sublanguage of the SPICA modelling language. It is used to specify messages and containers for the SPICA communication infrastructure. The modelling concept is closely related to structure definitions in programming languages such as C, but offers more advanced features like single inheritance, dynamic arrays, and strings. It provides a set of commonly used primitive types. Complex types are containers in the SPICA context, which are made up of primitive types or other containers again. A customizable serialization and de-serialization interface allows arbitrary message encodings to be used. Support for automatic conversion of

message values is supported by augmenting the model with references to semantic concepts and representations.
The objectives of ASN.1 and MDL are quite similar. However, ASN.1 lacks support for some fundamental techniques required by SPICA: It does not provide, for example, the mandatory concept of semantic annotations. In contrast, a custom modelling language like MDL may be designed in such a way that all required functionality is covered in a lean fashion. The specification support of MDL is sufficient for SPICA and we can easily extend it to our specific requirements.
We will now describe the modelling entities of MDL in more detail, covering the definition of message headers and containers. Messages, as they represent a specialization and aggregation of these concepts respectively, are introduced afterwards.

4.2.1 Headers

A header specification represents a special form of container used only for structuring the fields of the message header. These fields are used to control the way a message is handled and processed. There is only one mandatory field in SPICA: A special field holds the type identifier of the message as messages are strongly typed. A unique identifier is generated automatically using a suitable hash function. Other, mostly optional control fields in the header include the endian flag or the message identifier. Message headers may have different variants and be derived from each other. Headers may not be instantiated. Every message must be derived from exactly one header. From this point of view, a header represents an abstract class in the context of object-oriented programming.
The example below depicts the layout of a header specification using a minimal header with only one field identifying the type. A reference to the corresponding concept is required here to establish the meaning of the field.

```
header MessageBase {
        uint16 type [refconcept=#message:concept:type];
}
```

As outlined above, default context and representation URNs are assigned automatically. For this example they read `urn:spica:header:concept:MessageBase` and `urn:spica:header:rep:MessageBase` respectively.

4.2.2 Containers

Container specifications create composite types consisting of zero or more primitive or composite types. Containers may be derived through single inheritance from other containers. Compared to ordinary structures in C, instances of containers have the additional capability of being able to serialize and de-serialize themselves. Besides, they are not bound to the SPICA communication infrastructure only, but can be used as general purpose data structures as well.
A container is similar to a header but differs in one point: headers are only used as supertypes for messages while containers may only be used as a part of the message body. The example below shows a simple container. It defines the field `double d` referencing a semantic concept as well as a representation. A unique type identifier based on the

representation URN is assigned to each container implicitly. The example further defines the container as being a variant of type `cn`.

```
container Distance<cn> {
        double d [refconcept=#coord:distance, refrep=#rep:units:mm];
}
```

The concept URN reads `urn:spica:cn:container:concept:MessageBase`; its representation counterpart `urn:spica:cn:container:rep:MessageBase`. Considering the referenced concept and the representation, the value of `d` is a distance measured in mm.

4.2.3 Messages

One header and zero or more containers make up a message. The specification of a message differs from that of headers or containers in two ways. First, the topmost message in the inheritance hierarchy must be derived from exactly one header definition, providing all the required header fields. A container contains an implicitly defined type identifier which, however, is not represented as a field. A header is thus not required for a container. The second point in which a message differs from a container is that no primitive types may be added to the payload of a message; only containers are allowed. The example below outlines the definition of a message.

```
message DistanceMessage : MessageBase {
        Distance<cn> dist;
}
```

4.2.4 Coordinate Systems

In the area of AMRs, many containers are most likely to be used to store the position of some objects or observations in terms of their coordinates. In order to facilitate a correct interpretation of the fields of the corresponding container, the MDL also includes specification means to define coordinate systems. We include this modelling support into the MDL as it can be seen as additional semantic description of the containers – a container can reference a certain coordinate system. The specification of coordinate systems with regard to reference systems retains the freedom of choosing your own coordinate system and allow for automatic conversion between different ones.

In our modelling approach we currently support three basic types of coordinate systems – `Cartesian2D`, `Cartesian3D`, and `Polar2D` – and two different views – ego and allo. Egocentric coordinates resemble the egocentric view of the robot, whereas allocentric coordinates resemble the view of an external observer of the field. For the different types of coordinate systems we have some predefined concepts like an x-coordinate (`#coord:xcoord`) or a distance (`#coord:distance`) indicating the distance of the object from the pole of an polar coordinate systems. For a correct interpretation of container fields they have to refer to such a predefined concept.

In order to define a new coordinate system, the new origin, the axes (or a zero-ray for polar systems), and the view have to be specified. Some example specifications are shown in section 5.

4.3 Data flow Specification

One intention of the SPICA modelling language is to describe the communication behaviour in groups of AMRs in an abstract and platform-independent fashion. AMRs are basically hardware agents that are asked to accomplish a job or mission, similar to software agents. Just as software agents, AMRs can be regarded as modules which are mostly independent from each other. They further initiate mutual communication to exchange information and to collaborate. This is why the Data flow Definition Language (DFDL) follows an inherently modularized approach. Each AMR may be made up of several modules that communicate with each other or other AMRs. Such a scenario can easily be modelled given a modular software architecture where modules are connected via network links. There is, in fact, no difference between local and remote modules. For local communication, however, more suitable communication schemes might be chosen whereas communication between robots should be based on proven network communication schemes.
SPICA now introduces specification means for modelling data exchange between modules in an abstract manner. Modules are the main modelling entity here. For each module the requested as well as the offered message types have to be specified. At least one of the two options must be present; the module otherwise exhibits no functionality. For each communication direction – i.e. incoming and outgoing –DMCs are responsible for the management of messages and containers. They are used by the communication engine and by filters for passing data. Finally, the message transmission schemes have to be specified. They represent specific communication techniques tailored to the communication behaviour of AMRs. The block layout below outlines the basic structure of a module specification.

```
module Communication {
   offer { ... }
   request { ... }
   export { ... }
}
```

The offer, request, and export blocks will be introduced below in more detail. We will first start with the offer and request blocks that describe the basic communication structure of a module. The DMCs and transmission schemes are outlined thereafter. This section closes with the presentation of the description of filters incorporating the DADL sublanguage.

4.3.1 Message offers and requests

Let us now have a look at the specification of the most important parts of the module model. Offering and requesting messages is a fundamental functionality of MOM-based communication systems. In the DFDL model, *offer blocks* provide the information about offered, i.e. outgoing messages whereas *request blocks* deal with the reception of messages.
Every message that is provided by a module has to be listed in an offer block using the `message` directive. Along with the name of the message the specification of a transmission scheme is mandatory as it determines in which way the given message is handled. Apart from that, DMCs are required as input buffers and as temporary data storage for filters. The example below outlines the structure of an offer block.

```
offer {
   message DistanceMessage scheme ...;
   dmc ...;
}
```

It has to be noted here that arbitrarily many **message** and **dmc** statements may be listed. The **scheme** keyword defines the transmission scheme to be applied here. A request block is specified in exactly the same way except that it is not mandatory to specify a transmission scheme.
The relations between messages and DMCs are not explicitly modelled in the above example. It is, however, established automatically during model transformation. The next subsection will show how this can be accomplished.

4.3.2 Data Management Containers

Data Management Containers (DMCs) have been mentioned earlier already. They resemble data management structures for messages or containers in SPICA. They also perform basic synchronization tasks. The most important characteristic of the DMCs is the fact that each DMC is responsible for a specific semantic concept and a respective realization. This is where the relations between messages and DMCs are identified automatically.
In request blocks the identification of relations even goes one step beyond: For each incoming message not only the message type but also the types of the enclosed containers are checked. If the semantic type of a message container conforms to the referenced semantic type of a DMC, it is added to this DMC automatically. The representation of the container is further adapted to the DMC's representation if required. This way, further processing on the incoming data is possible in a very efficient manner. Irrelevant information is further discarded without manual intervention.
DMCs are implemented as linear lists with characteristics specific to the application domain: message passing. Queues and ringbuffers are more elaborate instances of linear lists and well-known examples of data structures in this context. All DMCs exhibit a consistent interface through which elements can be added, accessed, or retrieved. The semantic of these operations depends on the actual parameterisation, though. The retrieval operation, for example, may change the number of elements in the DMC, i.e. remove the element in question, or leave it alone.
The following DMCs with the stated characteristics are available in SPICA so far. More may be added if required. The size (`size`) and the management scheme (`scheme`) of a DMC may be changed using the appropriate annotations.

list: A list implements the semantics of an ordered list using a fixed-size buffer space. If the buffer is full, no new elements may be added. Elements have to be removed explicitly. The management scheme defaults to FIFO.

queue: A queue implements the semantics of an ordered list using a fixed-size buffer space. If the buffer is full, no new elements may be added. Elements are removed on retrieval except when using indexers. The management scheme defaults to FIFO.

ringbuffer: A ringbuffer implements the semantics of an ordered, circular list using a fixed-size buffer space as if it were connected end-to-end. If the buffer is full, the oldest element is overwritten if a new one is added. The management scheme defaults to LIFO.

The generic annotations `refconcept`, `refrep`, and `reftype` as introduced earlier are supported by every DMC. They are required to specify the element type.

Arrays of DMCs are supported as well. The array semantic is, however, not quite as expected: A DMC array is managed in such a way that each array element is uniquely assigned to one specific system that attends the communication. Every array thus has the same number of elements in the array, each of which corresponds to the same system. The DMC of the local system is also contained in the DMC array and can be retrieved using a dedicated operation on the DMC.
In order to complete the **`dmc`** statement in the example above, we will present a possible parameterization below. We will assume that a ringbuffer with only one element is used which accepts elements of the type `DistanceMessage`:

```
dmc ringbuffer dist [type=DistanceMessage, size=1];
```

DMCs have been defined only in the context of the model so far. It is very likely that more than one DMC is used and only a subset of these need to be accessible from userspace. The DFDL provides the **`export`** block for this purpose. DMCs that have to be visible from userspace only have to be added to the export block. The example below illustrates this.

```
export { dist; }
```

4.3.3 Transmission Schemes

Transmission schemes fulfil another very important task especially for offering messages. Data transmission in groups of AMRs is assumed to be very dynamic. Locations of modules local to a system are normally not subject to change but the location of modules on remote systems: Robots may join or leave a group spontaneously; other types of systems may appear and disappear in the same way.
This is why SPICA introduces the concept of transmission schemes for establishing communication links. In contrast to ordinary socket-based communication establishment, these schemes exhibit a special behaviour which is tailored to the dynamic communication behaviour of AMRs. For local modules, a static scheme that does not change its endpoints is sufficient as it can do without the overhead for dynamic binding. For ad-hoc communication with remote systems, however, heartbeat techniques are tried and tested. This is especially true for unreliable environments. Three schemes are outlined below which implement the requested characteristics of static and dynamic binding. It has to be noted here that only messages may be sent. Containers have to be wrapped in messages for this purpose.

static: The static transmission scheme is a one-to-one communication scheme which supports local communication characteristics. Basically, two modules are bound together statically. The location of communication partner must not be subject to change. It is not possible to change the communication endpoints during runtime. The `module` annotation is used by the sender to reference a destination module.

announce: The announce transmission scheme is a one-to-many communication scheme. The sender announces the availability of a data source to which one or more interested receivers can listen to. This scheme implements the heartbeat technique: The sender provides an alive-signal which contains the respective endpoints of the data source. This information is used by interested receivers to listen to the data source. The `heartbeat` annotation is used by the sender to specify the interval for the alive-signal.

request: The request transmission scheme is a one-to-many communication scheme. It is similar to announce but implements a variant of a publish-subscribe protocol. The sender again announces the availability of a data source but without immediately starting the transmission: It is triggered once at least one receiver is available. Receivers have to emit subscription heartbeats that inform the sender of the availability of an interested party. It depends on the endpoints provided by the receivers and on the number of receivers where and how the sender directs the data to. The `heartbeat` annotation is used to specify the interval for the alive signal for the sender as well as the receiver.

There are some annotations that are available for every transmission scheme. They basically resemble event processing capabilities: Transmission schemes must be either able to send messages periodically or after some events occurred. These annotations may be used both at the same time. The `interval` annotation is used to specify if a transmission should be triggered periodically. With the `on` annotation, the transmission scheme reacts on the events given in the annotation value.

In order to complete the **`scheme`** statement in the example above, we will present a possible parameterization below. We will assume that the announce scheme is used which sends messages with 30 Hz:

```
message DistanceMessage scheme announce [interval=33ms];
```

4.3.4 Filters

In the previous sections we introduced the MDL and the DFDL which allow us to realize communication infrastructures for heterogeneous groups of AMRs. The modules defined in DFDL exchange messages, adjust the representation of received data if required, and store the information into appropriate DMCs automatically. However, in order to also facilitate the development of a collaboration infrastructure in terms of a cooperative world model, we have to go beyond this: it must be possible to interpret the exchanged data and perform arbitrary calculations on them. For this purpose, we introduce the Data-Analysis Description Language (DADL). The design of the language and its modelling elements is based on the following observations.

The realization of a cooperative world model first requires data to be collected from the different robots in the group, which must then be combined to a consistent world view. Here, the impreciseness of the observations – which is mainly due to the physical limitations of the sensors used – has to be taken into account. For this purpose approaches for probabilistic state estimation are commonly applied, such as e.g. Bayesian Filtering (Aström, 1965; Fox et al., 1999; Thrun et al., 2000) and derivates like Kalman-Filtering (Kalman, 1960) or Particle Filters (Handshin & Mayne, 1969; Akashi & Kumamoto, 1977). In order to deal with uncertainty a number of different approaches are available: Dempster-Shafer theory (Dempster, 1968; Shafer, 1967), Bayesian Inference (Pearl & Kaufmann, 1988), and Fuzzy sets (Zadeh, 1978), to mention only some. In most cases, however, the implementation of these approaches is non-trivial and time-consuming. Therefore, we tried to identify frequently used patterns for which appropriate predefined filters are included in the DADL.

Especially in the area of Computer Vision but also in robotics, Matlab (http://www.mathworks.com/) is widely used for rapid prototyping. Matlab is a numerical computing environment and programming language, designed to efficiently deal with

matrices and operations on them. When dealing with arrays or lists it provides a very compact syntax and elaborate indexing methods. Therefore, we decided to adopt a Matlab-like syntax for specifying the algorithms needed to perform calculations on the exchanged data.
The main modelling element in DADL is a filter block because all calculations on exchanged data are considered as a form of filtering operation in SPICA. The basic structure of a filter block depends on whether a predefined filter pattern or a custom filter is required. We will first discuss the predefined filter patterns. Afterwards the custom filters are introduced.
The structure of a predefined filter patterns is given below.

```
filter <name> {
        <spec for the result DMCs>
        predefined <filter type> <annotations> {
                <additional input specs>
        }
}
```

A filter specification starts with a list of DMCs that store the results of the calculations. They are defined in exactly the same way as the DMCs for offer or request blocks in a module. They are furthermore globally accessible, can be referenced throughout the module and in other filters. So, it is possible to specify whole filter chains.
The actual specification for predefined patterns only includes the filter type and associated annotations. The annotations contain basic parameters for the filter and specify when the filter is triggered, similar to the annotations of the transmission schemes described above. The input for the predefined filter is specified in its body. It has to be noted here that it depends on the type of predefined filter which DMCs have to be provided for input and output. However, the SPICA framework is able to investigate the container for fields that correspond with the expected concepts and to associate them to the inputs and outputs of the filter.
Currently three predefined patterns are available to be used for data-fusion: *Kalman-Filter* (KF) (Kalman, 1960), *Multi-Hypothesis-Kalman-Filter* (MHKF) (Reid, 1979), and *Simple-Multi-Hypothesis-Estimation* (SMHE).
A KF is an approach for probabilistic state estimation that can be used to fuse data that represent observations on the field in terms of their Cartesian coordinates. It can also be utilized for object tracking and velocity estimation. However, it should be guaranteed that all the data to be fused correspond to the same object, i.e. the KF is not able to deal with false positives. The KF pattern can be parameterized in several ways. For example, it can be configured to realize a linear or a non-linear KF. In addition, it can be specified if the velocity of the observed objects should be taken into account and estimated and if the filtering should be iterative or non-iterative.
MHKF is an extension of the KF that is able to deal with false positives, i.e. not all observations must belong to the same object. For this purpose, the observations are forming a set of hypothesis for the object state and for each hypothesis a separate KF is applied. The pattern can be configured in the same way as the KF pattern.
SMHE is a simplification of the MHKF used in the Carpe Noctem software framework. It focuses on the multi-hypotheses tracking and avoids the complexity of MHKF through applying some heuristics. The pattern can be parameterized for an iterative or a non-iterative filtering.

We also intend to provide some predefined patterns to estimate situations based on the evidence for hypotheses. Here, Bayesian Inference and Dempster-Shafer theory will be applied. However, work on this is in a very preliminary state and therefore detailed descriptions are omitted here.
Custom filters are specified in a filter block as shown below:

```
filter <name> {
        <spec for the result DMCs>
        call <annotations> {
                <filter body>
        }
}
```

As with the predefined filters, the block for a custom filter starts with the specification of the result DMCs. In contrast to predefined filters, no special rules have to be followed here; the most appropriate DMCs can be chosen to store the results of the calculations. Annotations are used here as well to specify the way a filter is triggered. The filter body is a collection of statements in the Matlab-like DADL syntax, supporting arbitrary calculations on the DMCs.
From the Matlab programming language we have adopted – among other things – the following concepts:

- implicit typing
- control statements like for-loops, while-loops and if-conditions
- basic arithmetic and logical operations
- definition of matrices and vectors
- matrix operations
- array and matrix indexing methods
- a basic set of functions like `min`, `max`, `cos`, `sin`, `tan`, `atan`, `atan2`, `sqrt`, that accept also matrices as input values, perform the calculations per element, and may return matrices as well.

In addition to these concepts, MDL containers and DMCs are seamlessly integrated into the DADL language. DMCs can be indexed just like arrays in Matlab. We also allow calling methods on DMCs as, for example, to create or delete an element. The containers are accessed in the same way as structures in Matlab or in other common programming languages. Due to space limitations, we do not present the whole specification support provided by the DADL for the definition of custom filters here. In section 5, however, two custom filters are presented.

5. Evaluation and Results

In this section, we demonstrate the applicability of our approach. We show how the SPICA development environment was used to realize communication and collaboration in a mixed-team of soccer robots involving two heterogeneous platforms. For this purpose, we return to the scenario that was already outlined in section 2: A mixed-team formed by the Ulm Sparrows (referred to as US) – and Carpe Noctem (referred to as CN).
In our example, SPICA is used to establish a communication infrastructure bridging the gap between the two software frameworks. Afterwards, we show how SPICA facilitates the

development of a cooperative world model which is used to agree on a common ball position within the team and to realize a basic role assignment of the robots.
The cooperative world model can be realized either in a centralized or decentralized way. In a centralized scenario, the robots would share information with only one leading robot acting as hub or data sink. In the decentralized approach, AMR exchange information directly with all their team-mates in a peer-to-peer fashion. There is no designated data sink to which all data is sent to, but all systems process the provided information on their own. Each robot can so decide which piece of information is important and should be further processed.
We decided to go for the decentralized approach, as during the last RoboCup tournaments it has shown to be better suited and less error-prone. This is mostly because robot systems are likely to crash once or several times during a match due to hardware or software failures. Therefore, a team with one leading robot that coordinates the cooperative behaviour should be avoided. Besides, according to our experiences the network infrastructure at RoboCup tournaments is quite unreliable, bandwidth is scarce and the network is sometimes not available at all. For these reasons, we base our communication infrastructure on IP multicast, as it facilitates our decentralized approach and, as a connection less communication scheme, it is also not affected by an unreliable network with regard to blocking issues.

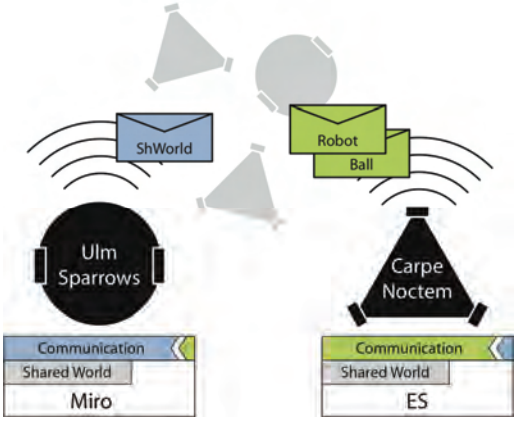


Fig. 3. System architecture of the mixed team "Ulm Sparrows and Carpe Noctem"

Fig. 3 provides a basic overview of the system architecture. Each of the robots of the Ulm Sparrows, depicted by a circle, and of Carpe Noctem, depicted by a triangle, runs an instance of the corresponding software framework which integrates two SPICA-generated modules, namely `Communication` and `SharedWorld`. The `Communication` module of the Ulm Sparrows is responsible for sending a `SharedWorld` message to the multicast group, containing the detected ball position and the robot's position on the field. In contrast, the `Communication` module of Carpe Noctem provides a separate `BallMessage` for the ball position and a `RobotPosMessage` for the robot's position on the field. Here, it is assumed that it fits more into the existing framework to send one combined message or two separated messages, respectively. The `SharedWorld` modules of the Ulm Sparrows and

Carpe Noctem have identical functionality: receiving the messages from all the team-mates, fusing the ball information to an agreed ball position and calculating a basic role assignment. Therefore, this module has to be specified only once and is generated for the two different target platforms: C++ for the Ulm Sparrows and C# for Carpe Noctem.
In the following paragraphs, we show how this architecture can be specified and realized with the help of the SPICA development environment. First, we start with the definition of messages, containers and the coordinate systems used by these two teams.

```
container BallPosition<cn> [refconcept=#concept:BallPosition,
                                    refcoord=#cn:coord:Cartesian2D] {
   double x [refconcept=#coord:xcoord, refrep=#rep:units:mm];
   double y [concept=#coord:ycoord, refrep=#rep:units:mm];
   double probability [refconcept=#refconcept:propability,
                              refrep=#rep:units:pct01];
   cov(x, y) cov;
}
container AlloBallPosition<cn> : BallPosition<cn>
                               [refcoord=#cn:coord:AlloCartesian2D];
container BallPosition<us> [refconcept=#concept:BallPosition,
                                    refcoord=#us:coord:Polar] {
   double d [refconcept=#coord:distance, refrep=#rep:units:mm];
   double alpha [refconcept=#coord:angle, refrep=#rep:units:deg];
   cov(d, alpha) cov;
}
container RobotPosition<cn,us> [refconcept=#concept:RobotPosition,
                                    refcoord=#coord:Cartesian2D] {
   double x [refconcept=#coord:xcoord, refrep=#rep:units:mm];
   double y [refconcept=#coord:ycoord, refrep=#rep:units:mm];
   double heading [refconcept=#coord:heading, refrep=#rep:units:deg];
   cov(x,y,heading) cov;
}
```

Listing 1. Definition of the containers

Listing 1 shows the specification of the containers that have to be defined for our communication and collaboration infrastructure. The listing includes two variants of the `BallPosition` container, one for Carpe Noctem and one for the Ulm Sparrows. Both containers refer to the concept `#concept:BallPosition`, but different representations corresponding to different coordinate systems are used. Carpe Noctem represents the position of the ball in an egocentric Cartesian coordinate system (`#cn:coord:Cartesian2D`), whereas the Ulm Sparrows use an egocentric polar coordinate system (`#us:coord:Polar`). For both containers the covariance matrices, representing the uncertainty of the observations with regard to the corresponding coordinate systems, are defined. In addition, the `BallPosition` container of Carpe Noctem also includes a field indicating the probability of the observation to be correct. The `AlloBallPosition` container of Carpe Noctem has the same fields as the Carpe Noctem `BallPosition` container, however refers to an allocentric coordinate system (`#coord:cn:AlloCartesian2D`). Besides, the `RobotPosition` container definition is identical to Carpe Noctem and the Ulm Sparrows. It is noteworthy here that the semantic annotation of containers can be considered as just a way of commenting, but of course with some guidelines.

```
coord Cartesian2D<cn> [refconcept=#coord:Cartesian2D] {
   origin = (0.0, 0.0) [refconcept=#coord:origin, rep=#rep:units:mm];
   xAxis = (1.0, 0.0) [refconcept=#coord:xaxis, rep=#rep:none];
   yAxis = (0.0, 1.0) [refconcept=#coord:yaxis, rep=#rep:none];
   view = ego [refconcept=#coord:coordView, rep=#rep:coord:coordView];
}
coord AlloCartesian2D<cn> [refconcept=#coord:Cartesian2D] {
   origin = (0.0, 0.0) [refconcept=#coord:origin, rep=#rep:units:mm];
   xAxis = (1.0, 0.0) [refconcept=#coord:xaxis, rep=#rep:none];
   yAxis = (0.0, 1.0) [refconcept=#coord:yaxis, rep=#rep:none];
   view = allo [refconcept=#coord:coordView, rep=#rep:coord:coordView];
}
coord Polar<us> [refconcept=#coord:Polar2D] {
   pole = (0.0, 0.0) [refconcept=#coord:pole, rep=#rep:units:mm];
   zero_ray = (1.0, 0.0) [refconcept=#coord:zero_ray, rep=#rep:none];
   view = ego [refconcept=#coord:coordView, rep=#rep:coord:coordView];
}
```

Listing 2. Definition of the coordinate systems

Listing 2 shows the specification of the coordinate systems used by Carpe Noctem and the Ulm Sparrows. As already mentioned above, the coordinate systems are described by providing values for predefined concepts like the view, the axis and the origin (`pole`) of the coordinate system with regard to the reference systems.

```
message SharedWorldMessage<us> : MessageBase {
   BallPosition<us> ballPos;
   RobotPosition<us> robotPos;
}
message BallMessage<cn> : MessageBase {
   BallPosition<cn> ballPos;
}
message RobotPosMessage<cn> : MessageBase {
   RobotPosition<cn> robotPos;
}
```

Listing 3. Definition of the messages

With the help of these specifications the framework is able to provide the appropriate transformations between the representations in different coordinate systems automatically. Here, the development environment also deals with automatic conversions between the measurement units as well as with automatic transformation of the covariance matrices. As now the specification of all needed containers and of the coordinate systems are available, the corresponding messages can be defined as shown in Listing 3.
As depicted in the overview of the architecture, we have to define a `SharedWorldMessage` for the Ulm Sparrows that includes a `BallPosition` and a `RobotPosition`. The communication module of Carpe Noctem provides the same kind of information but in two separate messages: `BallMessage` and `RobotPosMessage`.

```
module Communication<us> {
   offer {
      dmc ringbuffer sharedWorld [type=SharedWorldMessage<us>, size=1];
      message SharedWorldMessage<us> scheme announce
                                     [on=sharedWorld.new"];
   }
   export { sharedWorld; }
}
module Communication<cn> {
   offer {
      dmc ringbuffer ballMessage     [type=BallMessage<cn>, size=1];
      dmc ringbuffer robotPosMessage [type=RobotPosMessage<cn>, size=1];
      message BallMessage<cn> scheme announce [period=100ms];
      message RobotPosMessage<cn> scheme announce [period=100ms];
   }
   export {
      ballMessage;
      robotPosMessage;
   }
}
```

Listing 4. Definition of the *Communication* modules

Now we can start with the definition of the modules of our architecture. The specifications of the *Communication* modules for the two teams are shown in Listing 4. The Communication module of the Ulm Sparrows offers a `SharedWorldMessage` using the announce scheme, when a new message is placed into the ringbuffer. In contrast, the *Communication* module of Carpe Noctem is defined to offer the two messages, `BallMessage` and `RobotPosMessage`, each of which is sent with 10Hz. In order to allow other parts of the software framework to access the message DMCs, they are included into the export block in both modules.

```
module SharedWorld<cn,us> {
   request {
      message SharedWorldMessage<us>;
      message RobotPosMessage<cn>;
      message BallMessage<cn>;
      dmc ringbuffer[] balls [concept=#concept:BallPosition,
               rep=#cn:container:rep:BallPosition, size=10, ttl=5s];
      dmc ringbuffer[] robots [concept=#concept:RobotPosition,
               rep=#cn:container:rep:RobotPosition, size=10, ttl=5s];
   }
   filter ego2Allo { ... }
   filter calculateSharedBall { ... }
   filter calculateRoleAssignment { ... }
   export { ... }
}
```

Listing 5. Definition of the *SharedWorld* module

As the `SharedWorld` module provides a more complex functionality – receiving messages, fusing the exchanged data and calculating a role assignment – we show the corresponding

specification in several steps. Listing 5 provides a general overview of the description and focuses on the request block for receiving messages.
The module is defined to receive the messages `SharedWorldMessage` from Ulm Sparrows robots, and the `RobotPosMessage` and the `BallMessage` from Carpe Noctem robots. DMC arrays are specified to be filled with the `BallPosition` (`#concept:BallPosition`) and `RobotPositions` (`#concept:RobotPosition`) containers for each of the robots in the team. The corresponding information is automatically extracted from the messages named above.
In order to realize the functionality of calculating the agreed ball position and a basic role assignment, we have to define filters that are able to process the data collected in the DMCs or more precisely DMC arrays. The `BallPosition` containers are represented with regard to an egocentric coordinate system, so we first have to transform the data into an allocentric representation. A common allocentric view is a prerequisite to fuse data about observations collected by a group of robots. This is the purpose of the filter `ego2Allo`. The corresponding specification is shown in Listing 6.

```
filter ego2Allo {
   dmc ringbuffer[] alloBalls [concept=#concept:BallPosition,
                               rep=#cn:container:rep:AlloBallPosition,
                               size=10];
   call [on=balls.new]{
      index = balls.changedIndex();
      heading = robots(index).last.heading;
      alloX = robots(index).last.x;
      alloY = robots(index).last.y;

      alloX = alloX + cos(heading)*balls(index).last.x -
                               sin(heading)*balls(index).last.y;
      alloY = alloY + sin(heading)*balls(index).last.x +
                               cos(heading)*balls(index).last.y;
      alloBalls(index).new;
      alloBalls(index).last.x = alloX;
      alloBalls(index).last.y = alloY;

      //Calculate and assign transformed covariance matrix
      alloBalls(index).last.cov = ... ;
   }
}
```

Listing 6. A filter performing the transformation from egocentric to allocentric view

First, the filter defines a ringbuffer array named `alloBalls` that stores the allocentric ball positions (`#concept:BallPosition`) in the representation `AlloBallPosition` (`#cn:container:rep:AlloBallPosition`) for each of the robots in the team. Afterwards, the body of the filter is defined. It is called every time a new ball position is available. The filter fetches the index of the corresponding ball DMC in the array which represents the number of the respective player in the team. Then variables are declared and initialized with the robot's last position on the field. To this position the egocentric ball position rotated by the heading of the player on the field is added. Now a new container is added to the array and initialized with the coordinates of the allocentric ball position calculated above. At the end of the filter body, the field for the covariance matrix is

assigned. Due to space limitation the corresponding transformation was left out. However, as just a rotation of the matrix is required, the transformation can be specified in basically the same way as the rotation of the egocentric ball coordinates.

```
filter calculateSharedBall {
   dmc list sharedBall [concept=#concept:BallPosition,
                        rep=#cn:container:rep:BallPosition];
   predefined MHKF [period=100ms, iterative, linear, staticObject] {
      input = alloBalls(:).last;
   }
}
```

Listing 7. An MHKF for the agreed ball position

All ball positions are now available in a common allocentric view, thus we can apply a Multi-Hypothesis-Kalman-Filtering (MHKF) to fuse the information and keep track of different hypothesis of the ball position on the field. In our case, the agreed position can be determined as the position hypothesis with the highest probability. Listing 7 illustrates how the corresponding MHKF can be included into the specification.

```
filter calculateRoleAssignment {
   dmc ringbuffer[] roles [concept=#concept:Role, rep=#rep:string,
                           size=1, init="None"];
   call calculateRoleAssignment [period=100ms]{
      [maxProb, maxIndex] = max(sharedBall.probability);
      teamBall = sharedBall(maxIndex);
      ballDistances = sqrt((robots(:).last.x - teamBall.x).^2 +
                           (robots(:).last.y - teamBall.x).^2);
      [minBallDist, AttackerIndex] = min(ballDistances);
      roles(AttackerIndex) = "Attacker";

      minXPos = 20000.0;
      minIndex = -1;

      for i = 1:teamsize()
         if(roles(i) == "None" && robots(i).last.x < minXPos)
            minXPos = robots(i).last.x;
            minIndex = i;
         end;
      end;

      roles(minIndex) = "Defender";

      for i = 1:teamsize()
         if(roles(i) == "None")
            roles(i) = "Supporter";
         end;
      end;
   }
}
```

Listing 8. A filter for calculating the role assignment

The filter `calculateSharedBall` uses a list named `sharedBall` to store the hypothesis for the ball positions returned by the MHKF. The probability of the hypothesis is automatically assigned to the corresponding field in the container which is determined by the associated concept `#concept:probability`. The MHKF is iteratively applied with 10Hz. The options `linear` and `staticObject` indicate that a linear Kalman-Filter is used and that no velocity of the object should be estimated and the velocity is not considered when applying the motion model. Of course, the MHKF also has to know which data it has to work on. The input of the Kalman-Filter is given as the last observed allocentric ball positions of all the robots in the team.

Listing 8 shows the specification of the filter `calculateRoleAssignment`. As for all filters, the definition starts with creating a DMC or DMC array to store the results of the filtering. Here a ringbuffer array named `roles` is defined to store the roles of all the robots (`#concept:Role`). Each array element is a ringbuffer that contains exactly one element of type string that is initialized with the string "None". At the beginning of the filter body the index of the shared ball hypothesis with the highest probability is calculated using the max-function. This index is used to store the corresponding shared ball hypothesis in `teamBall`. Afterwards, the distances of the robots to the team ball are calculated and the index of the player nearest to the ball is determined. The following line associates the role "Attacker" with it. Next, we determine the player which is nearest to the own base line – indicated by the minimal x-coordinate of the corresponding robot position – and has no role associated yet. This is achieved by using a for-loop and an appropriate if-condition. Afterwards, the resulting player gets the role "Defender". All remaining players are associated with the role "Supporter", which is also done using a for-loop.

Now the specification of the `SharedWorld` module is nearly complete. Only the DMCs `sharedBall` and `roles` have to be exported, in order to make them available for access from other parts of the underlying software framework (not shown here).

With the help of all theses specifications, the SPICA development environment is able to generate source code for the whole communication and collaboration infrastructure. For this purpose, the AASTra tool has to be told about the target platform the modules and data structures should be generated for, and the desired communication scheme (IP multicast) has to be configured. After the transformation, the resulting modules and classes can easily be integrated into the underlying communication frameworks. Only an instance of the generated module has to be created as a singleton and the DMCs can be accessed by the generated API.

Listing 9 illustrates the corresponding source code fragments for integration into the Carpe Noctem framework in C#. First, a callback method `GetSharedBallHypotheses` is defined. It is called when the `sharedBall` DMC of the SharedWorld model has been changed. This is the case, every time the MHKF has finished an iteration. In this simple example, the method just writes all hypotheses to the console. Afterwards, an instance of the `SharedWorld` module is created and the callback is added as a delegate to the `Changed`-event of the `SharedBall` property for the respective DMC. The following line shows how the role of the current robot can be accessed (`MyBuffer` returns the DMC of the current robot from the DMC array). The rest of the source code fragment creates an instance of the `Communication` module, creates a new `BallMessage`, initializes its fields, and adds it to the corresponding DMC. The transmision of the message is handled by the `Communication` module as specified above.

```
using Spica.Modules;
using Spica.Messages;

...

protected void GetSharedBallHypotheses(Module m) {
   SharedWorld sw = (SharedWorld)m;

   Console.WriteLine("SharedBall Hypotheses: {0}",
      sw.SharedBall.ToString());
}

SharedWorld sw = SharedWorld.GetInstance();

sw.SharedBall.Changed += GetSharedBallHypotheses;

Console.WriteLine("Own Role: {0}", sw.Roles.MyBuffer.Last);

Communication c = Communication.GetInstance();

BallMessage bm = new BallMessage();

bm.BallPos.X = 1000.0;
bm.BallPos.Y = 1000.0;
bm.BallPos.Certainty = 1.0;

c.BallMessage.Add(bm);
```

Listing 9. C# fragment showing the integration of SPICA-generated code

6. Related Work

Research on robot software architectures in the past mostly focused on middleware frameworks for autonomous robot development. Abstraction layers in this approach simplify access to robotic hardware and make it more convenient to use. By adding abstract interface definitions and APIs, modular programming is promoted.

Our approach shifts the focus right to the development process, a conceptually even more abstract level. We address the way systems have to be developed and the question what has to be implemented. The goal is to make the development process and the implementation more platform-independent, enabling the developer to focus on the actual functionality rather than bothering with characteristics of the platform. Our development environment for robotic software neither has hard dependencies on hard- or software architectures nor on operating systems. We provide modelling facilities that are focused on the respective program domain such as multi-party interaction or distributed sensor fusion. By combining ideas from the model-driven development movement with lessons learned from the development of middleware frameworks, a powerful development tool chain is provided.

As robotic systems are normally quite reactive and the system configuration is likely to be modified during the development process, one key requirement is the ability to incorporate new or existing components into the given software architecture. Furthermore, especially AMRs have to be able to use heterogeneous hardware devices, cope with physical variability in measurement, and bypass architectural mismatches. Several approaches have

been proposed in the last years which try to provide suitable solutions or address similar problems in other related areas. A range of solution and projects is outlined below.

6.1 Middleware Frameworks

Several middleware frameworks utilize the concept of abstraction layers to ease the development of robotic software in a heterogeneous environment.

MARIE (Mobile and Autonomous Robotics Integration Environment) (Côté et al., 2006) is a middleware framework for robots that targets the development and integration of software components. It provides the Mediator Interoperability Layer (MIL), a design pattern that offers a common interaction language for components in the system. MARIE itself is written in C++ for UNIX environments. CLARAty (Coupled Layer Architecture for Robotic Autonomy) (Volpe et al., 2001) is an object-oriented framework for robotic systems which focuses reusability and integration of algorithms and components. It basically reduces the software hierarchy to two layers, a decision and an execution layer; realizations of functional requirements can be integrated into the decision layer while the execution layer is not affected. Another object-oriented framework for robotic applications is MIRO (Middleware for Robots) (Utz et al., 2002). It provides abstraction from system-specific implementations and is based on the ACE/TAO (Schmidt et al., 1997) framework. A device layer features hardware abstraction and takes care of the operating system integration. A communication layer offers services required in distributed systems. A Service Layer finally provides abstractions for sensors and actuators by decoupling the device interfaces from the driver implementations.

Similar to MIRO where skeletons for sensors and actors can be described using an Interface Definition Language (IDL), the Reconfigurable Context-Sensitive Middleware (RCSM) (Yau et al., 2002) uses a newly defined IDL to specify context requirements. It is a middleware framework supporting the development of context-aware applications focusing on spontaneous interactions. Application skeletons are generated from the IDL specifications which interact with the RCSM Object Request Broker (R-ORB), the context management processor in RCSM.

The Pervasive Autonomic Context-aware Environments (PACE) (Henricksen et al., 2005) middleware provides tools for validating context models, generating stubs for different languages, and accessing context from different programming languages and platforms. It provides a context management system (CMS) with a distributed set of content management repositories. The queries to the CMS can be placed using RMI or automatically generated stubs, for example.

In contrast to the approaches outlined above, SPICA is no middleware framework but a development environment aiming at platform-independent specifications and automatic code generation. Therefore, we address a conceptually different level. Besides, its flexible code generation system easily adapts to new target languages and we focus on a convenient modelling and on lean generated code.

AMQP (Advanced Message Queuing Protocol) (http://www.amqp.org/) is an open standard messaging middleware. It was developed first off to meet the needs of investment banks, employing a network-friendly, binary protocol. Similar to the DMCs used in SPICA, AMQP provides queues to accomplish a store-and-forward semantic. Message routing and delivery is due to centralized message broker systems.

The implementation generated by SPICA exhibits MOM characteristics, as well. In contrast to AMQP, however, SPICA also supports decentralized peer-to-peer techniques.

6.2 Development Environments

A quite different approach is followed by the Microsoft Robotic Studio (http://msdn.microsoft.com/robotics/). It is a development environment for robotics that targets different robot platforms. It builds on the .NET framework and offers a runtime as well as a powerful simulation environment. Besides the programming languages available in .NET, a so-called Visual Programming Language may be used for development of robotic software. Therefore, it can be considered as a model-driven software development approach. CoSMIC (Component Synthesis using Model-Integrated Computing) (Gokhale et al., 2003; Balasubramanian et al., 2005) is another development environment which follows the paradigm of MDD. It is a collection of domain-specific modelling languages and generative tools for the development, configuration, deployment, and validation of distributed component-based real-time systems.

Both approaches are similar to SPICA. The Microsoft robotics studio targets rapid development of robot control software but focuses more on prototyping than on efficient and domain-adapted solutions. CoSMIC is a complex, model-driven approach that follows very similar goals. In contrast, our approach aims to be lightweight and allows for rapid development and easy integration.

6.3 Context Management Systems

In the area of context-aware computing applications and middleware services use information about their execution environment to adapt their functional and non-functional behaviour for appropriate quality of service in every situation. For this purpose, context management systems are required which collect context information and make them accessible for adaptation reasoning. However, in a pervasive computing environment it is very likely that context information originate from heterogeneous sources. Therefore, many research activities addressing the development of context management systems also focus on heterogeneity issues. Examples are RCSM and PACE already mentioned above, but also the Context Toolkit (Salber et al., 1999), CoCo (Buchholz et al.) and CoBrA (Chen et al., 2003). While RCSM and PACE aim at providing an infrastructure to integrate heterogeneous context providers, the Context Toolkit, CoCo, and CoBrA focus on the interpretation of context information from heterogeneous sources. In particular, CoCo and CoBrA are related to our approach as they claim the necessity of using ontologies to establish a common understanding of the semantics of context information and their representations. However, here it has to be distinguished between approaches using ontologies for runtime reasoning and for code generation purposes as in our case. Our approach also has many similarities to the Context Ontology Language (CoOl) already mentioned above, as they also deal with different representations and define operations to convert between them. However, in our approach the operations for conversion are not defined explicitly, but we aim at automatically deriving the conversion methods from the definition of coordinate systems and references to measurement units.

In general, the development of a cooperative world model has many similarities to the development of context management systems. Here too, information about the current

environment, like the ball position, player position etc., has to be collected and calculations have to be performed on them. In the area of context aware computing this is referred to as context reasoning. There are also some approaches providing development support and patterns for context reasoning. An example is the work done by Chen et al. (Chen et al., 2004). They propose the use of Context Fusion Networks (CFNs) to provide data fusion services with regard to the aggregation and interpretation of sensor data to context-aware applications.

7. Conclusion and Future Work

Because of the lack of standard software, which prompts every RoboCup team to develop its own software framework, heterogeneity issues play a decisive role. They cause several problems when establishing a mixed-team of soccer robots involving different hardware and software platforms.

In order to cope with theses issues, we presented SPICA, a development environment for communication and collaboration infrastructures for heterogeneous teams of soccer robots. In SPICA, we have adopted a model-driven development approach which is naturally very appropriate to cope with heterogeneity. One of its basic paradigms is the platform-independent specification of software allowing automatic generation of source code for different platforms. Accordingly, SPICA provides a modelling language and tools facilitating the specification of communication and collaboration infrastructures as well as the automatic transformation of the resulting models into source code.

The SPICA modelling language consists of three domain-specific sublanguages, which are tailored to different aspects of the infrastructure. The MDL allows the specification of messages and containers along with their representations. The DFDL provides specification means for module stubs, the data flow between them, and for their data management capabilities. In order to allow a flexible filtering of data and to support the creation of a cooperative world model, the DADL was developed. It is a general purpose language for calculations on the exchanged data and also provides some predefined patterns for data fusion. As illustrated in a detailed example, the development effort for a team-play in heterogeneous teams of soccer robots can be reduced significantly with the help of SPICA. The generated source code can be integrated into the existing software framework very easily and with very little effort.

However, as already mentioned above, the development of SPICA is still work in progress. In particular, this is true for the DADL and the corresponding transformation support. Appropriate support for code generation is available only for a subset of the predefined data-fusion patterns at the moment and only a basic set of predefined functions is integrated into the language. In the future, we will enhance the language and the corresponding code generation tools with regard to especially these issues. We also aim at integrating support for defining functions and calling functions from external libraries. Besides, as not only the programming of a complex communication infrastructure is a challenging task, but also its configuration, we try to include support for self-configuration of the generated infrastructures into the SPICA environment.

However, we are quite confident that with the SPICA development framework one important step was made towards the realization of cooperative team organization. It is our vision that teams provide and publish descriptions of the messages and corresponding data

they would like to communicate. For new mixed teams only the tactics would have to be specified then; the appropriate communication and collaboration infrastructure is generated by the SPICA development framework automatically.

8. References

Akashi, H. & Kumamoto H. (1977). Random sampling approach to state estimation in switching environments. *Automatica*, Vol. 13, pp. 429–434.

Aström, K. J. (1965). Optimal control of markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, Vol. 10, pp. 174–205.

Baer, P. A.; Reichle, R.; Zapf, M.; Weise, T. & Geihs, K. (2007). A Generative Approach to the Development of Autonomous Robot Software. *Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASe 2007)*, pp. 43-52.

Balasubramanian, K.; Krishna, A. S.; Turkay, E.; Balasubramanian, J.; Parsons, J.; Gokhale, A. & Schmidt, D. C. (2005). Applying Model-Driven Development to Distributed Real-time and Embedded Avionics Systems. *International Journal of Embedded Systems, special issue on Design and Verification of Real-Time Embedded Software.*

Buchholz, T.; Krause, M.; Linnhoff-Popien, C. & Schiffers, M. (2004). Dynamic Composition of Context Information. *Proceedings of 1st Ann. International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 04)*, pp. 335-343.

Castelpietra, C.; Iocchi, L.; Nardi, D.; Piaggio, M.; Scalzo, A. & Sgorbissa, A. (2000). Coordination among heterogeneous robotic soccer players. *Proceedings of Intelligent Robots and Systems 2000 (IROS 2000)*, Vol. 2, pp. 1385-1390, ISBN 0-7803-6348-5, Takamatsu, Japan.

Chen, G.; Li, M. & Kotz, D. (2004). Design and implementation of a large-scale context fusion network. *Proceedings of 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous)*, pp. 246–255, IEEE Computer Society.

Chen, H.; Finin, T. & Joshi, A. (2003). Using OWL in a pervasive computing broker. *Proceedings of Workshop on Ontologies in Agent Systems*, July 2003.

Côté, C.; Brosseau, Y.; Létourneau, D.; Raïevsky, C. & Michaud, F. (2006). Robotic Software Integration Using MARIE. *International Journal of Advanced Robotic Systems, Special Issue on Software Development and Integration in Robotics*, Vol. 3, No. 1, pp. 55-60.

Dempster, A. P. (1968). A generalization of Bayesian inference, Journal of the Royal Statistical Society, Vol. 30, Series B, pp. 205-247.

Fox, D.; Burgard, W. & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, Vol. 11, pp. 391–427.

Gokhale, A. S.; Schmidt, D. C.; Lu, T.; Natarajan, B. & Wang, N. (2003). CoSMIC: An MDA Generative Tool for Distributed Real-time and Embedded Applications. *Proceedings of Middleware Workshops*, pp. 300–306.

Handschin, J. E. & Mayne, D. Q. (1969). Monte carlo techniques to estimate the conditionalexpectation in multi-stage non-linear filtering. *International Journal of Control*, Vol. 5, No. 5, pp. 547–559.

Henricksen, K.; Indulska, J.; McFadden, T. & Balasubramaniam, S. (2005). Middleware for Distributed Context-Aware Systems. *On the Move to Meaningful Internet Systems 2005*, LNCS 3760, pp. 846-863, Springer.

Isik, M.; Stulp, F.; Mayer, G. & Utz, H. (2007). Coordination without Negotiation in Teams of Heterogeneous Robots. In: *RoboCup 2006: Robot Soccer World Cup X*, Vol. 4434, ISBN 978-3-540-74023-0, to appear.

Kalman, R.E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, Vol. 82, Series D, pp. 35–45.

Nardi, D.; Adorni, G.; Bonarini, A.; Chella, A.; Clemente, G.; Pagello, E. & Piaggio, M. (1999). ART – Azzurra Robot Team. In: *RoboCup-99: Robot Soccer World Cup III*, LNCS 1856, pp. 15-39, Springer.

Nesnas, I. A.; Simmons, R.; Gaines, D.; Kunz, C.; Diaz-Calderon, A.; Estlin, T.; Madison, R.; Guineau, J.; McHenry, M.; Shu, I.-H. & Apfelbaum; D. (2006). CLARAty: Challenges and Steps Toward Reusable Robotic Software. *International Journal of Advanced Robotic Systems, Special Issue on Software Development and Integration in Robotics*, Vol. 3, No. 1, pp. 23-30.

Pearl, J. & Kaufmann, M. (1988). Probabilistic Reasoning in Intelligent Systems. San Mateo CA, ISBN 0-934613-73-7.

Reid, D. (1979). An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, Vol. 24, Nr. 6, pp. 843–854.

Salber, D.; Dey, A. K. & Abowd, G. D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. *Proceedings of Conference on Human Factors in Computing Systems (CHI '99)*, pp. 434-441, Pittsburgh, PA, May 15-20, 1999.

Schmidt, D. C.; Gokhale, A.; Harrison, T. & Parulkar; G. (1997). A high-performance endsystem architecture for real-time CORBA. *IEEE Communications Magazine*, Vol. 14, No. 2.

Selic, B. (2003).The pragmatics of model-driven development. IEEE Software, Vol. 20, No. 5, pp. 19–25.

Shafer, G. (1976). A Mathematical Theory of Evidence. Princeton University Press.

Strang, T.; Linnhoff-Popien, C. & Korbinian, F. (2003). CoOL: A Context Ontology Language to enable Contextual Interoperability. *Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS 2003)*, LNCS 2893, pp. 236-247, ISBN 3-540-20529-2, Springer.

Thrun, S.; Fox, D.; Burgard, W. & Dellaert, F. (2000). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, Vol. 128, No. 1-2, pp. 99–141.

Utz, H.; Sablatnög, S.; Enderle, S. & Kraetzschmar, G. K. (2002). Miro–Middleware for Mobile Robot Applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, Vol. 18, No. 4, pp. 493-497.

Utz, H.; Stulp, F. & Mühlenfeld, A. (2004). Sharing Belief in Teams of Heterogeneous Robots. In: *RoboCup 2004: Robot Soccer World Cup VIII*. LNCS 3276, pp. 508-515, Springer.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R. & Das, H. (2001). The CLARAty Architecture for Robotic Autonomy. *Proceedings of IEEE Aerospace Conference*, Big Sky, Montana.

Yau, S. S.; Karim, F.; Wang, Y.; Wang, B. & Gupta, S. K. S. (2002). Reconfigurable Context-Sensitive Middleware for Pervasive Computing. *IEEE Pervasive Computing*, Vol. 1, No. 3, pp. 33-40.

Zadeh, L. A. (1978). Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems, Vol. 1, pp. 3–28.

2

Positioning in Robots Soccer

Hesam T. Dashti [1,3] , Shahin Kamali [2] and Nima Aghaeepour [3]
1. Center of excellence in Biomathematics, Faculty of Science, University of Tehran
Iran
2. Department of Computer Science, Concordia University
Canada
3. Department of Computer Science, Faculty of Science, University of Tehran
Iran

1. Introduction

In this chapter, the focus is on an important issue of robots' decision making process which is positioning. Positioning involves making the best decision for the agents who do not possess the ball regarding the team strategy and consequently finding the best target for them. Current section would express some evidences to prove the positioning criticality.
Multi-agent frameworks usually restrict the agents' communications; therefore understanding the game state and collaborators' situation is based on the information provided by the world model. Regards to this information the agent must decide about game state and his action.
In robot soccer competition, separate tasks need to be defined for the agent defending the goal (the goalie) and the agent intercepting the ball (the active agent). The other agents are *strategic agents* who need to do positioning, i.e. getting distributed in the field regarding the team strategy. The positions of these strategic players have impressive effects on decision making process of both teammate and opponent agents. In the following paragraphs we explain the effects of good positioning on teammates tasks.

- The collaboration between goalie and goal defenders is of crucial importance for saving the goal. A simple kind of this collaboration occurs when defenders' positions cover some parts of the goal coordinates; this enables goalie to take care of a small part of the goal area rather than whole of it.

- The importance of collaboration between active agent and strategic agents is important as well. To see that, we need some knowledge about active agent tasks and decision making process. Active agent can kick the ball by different velocities and (in some soccer frameworks) different kicking angles. Applying these options, different actions can be defined for active agent. For example, a kick with small velocity is a dribble action and a kick by higher velocity can be a pass action. Assume the active agent wants to pass the ball to a teammate. The teammate receiving the pass should be close enough to the active agent (to avoid world-model's noise affects) and also far from opponent agents (to reduce the probability of losing the ball). If the active agent concerns just these factors, the positioning process can be defined as distributing the

agents in the way of increasing the number of eligible agents. Provided with more eligible agents, the active agent can perform its action with higher accuracy.

To see the effect of good positioning on opponents decision making process, assume that the opponent's active agent wants to select one of his teammates to pass the ball. By blocking, i.e. following and sticking to the other opponents, the opponent's active agent can not find free teammate and its decision making process would be corrupted. Also through blocking, the probability of seizing the ball would be increased and the opponents' attacks would be counteracted. As it is clear blocking should be defined as a part of positioning process.

We come to conclusion that a perfect positioning methodology is an important issue for a team. As described above a good positioning of strategic agents improves the performance of goalie and active agent. Moreover a good positioning is vital for blocking opponents in a defensive situation. 'Offside trap' is another plan that can be applied through a good positioning method to keep the goal area safe.

After describing the benefits of perfect positioning methodologies, we will see some obstacles of developing such methods; during the competition, regards to the ball position, strategic agents have to traverse long paths to reach suitable position and driving to the destination, needs agent's stamina. Whereas robots (like humans) have restricted stamina, saving the stamina is vital and developing positioning methods in which agents spend minimum stamina is another challenge in positioning process. Another aspect for developing beneficial positioning process is determining effective parameters for positioning process. Some of these parameters could be the attraction vectors to the ball, teammates, opponents and aggressiveness vector to the opponent goal.

Positioning decision making is based on the information received from agent's environment. Analyzing this information enables agent to decide about its further position. After processing the information the agent does not need more interactions to exterior components. It means that the positioning process is independent of robot's class (simulated in 2 or 3 dimensions, middle size, humanoid, etc.) and the protocol of gathering information. Here we illustrate the positioning methodologies implemented in Robocup soccer simulation framework; however they can be extended to other robot soccer frameworks.

Different approaches are presented in Robocup soccer simulation framework. The common approach is Dynamic Positioning which is considered in this chapter. There are two popular dynamic positioning methods in Robocup soccer simulation framework. Each of them has some advantages and disadvantages. The first approach is Situation Base Strategic Positioning (SBSP) presented by FC Portugal soccer simulation team (Reis et al., 2001(a)), and the second is Dynamic Positioning based on Voronoi Cells (DPVC) which presented by UTUtd soccer simulation team (Dashti et al., 2006).

The SBSP method defines target position for strategic agents; an agent's target position is calculated regarding the agent's role and the current formation of teammates. Also some home positions are assigned to agents such that each agent is allowed to move just in the specific area defined around its home position. These home positions are used for assigning roles to agents. The target position is calculated based on agents' home positions. Formation is arrangement of agents in the game field and involves agents' roles. Regarding the game situation different formations and respectively different home positions are defined. These home positions are one of the restrictions of the SBSP method. DPVC solves this problem and introduces a methodology in which no home position and formation is defined.

In section 2 we have an overview on present positioning methods in Robocup soccer framework. Centroidal Voronoi Diagrams would be discussed in sections 3. DPVC as a

dynamic positioning approach of positioning is presented in section 4. Section 5 contains the results of some experiments in which DPVC and SPSB are compared.

2. Previous Works

2.1 Strategic Positioning

Generally at a certain moment, the positioning destination of an agent is called its strategic position. Strategic positioning method discusses about problems of choosing the strategic position and way of driving to there. There is always one player who is associated with the ball and obeys a different decision method based on the strategy; other agents should move toward their best position in the field. So positioning should be carefully implemented in the team strategy. The most popular strategic method for positioning in Robocup Soccer Simulation is SBSP (Situation Based Strategic Positioning) (Reis et al., 2001[a]) which is presented by FC Portugal team (Reis et al., 2001[b]). This method defines specific target positions for agents who do not possess the ball; these target positions are calculated with regard to the current formation of the team and roles of agents in this formation. For active situations, the agent position on the field is calculated using specific ball possession, ball recovery or playoff decision mechanisms. To calculate its strategic positioning, the agent analyses which is the game situation, tactic and formation in use and its positioning (and corresponding player type). Using the tactic, formation and positioning the agent calculates its base strategic position in the field in that formation.

This position is then adjusted according to the ball position and velocity, situation (attack, defense, scoring opportunity, etc.) and player type strategic information. The agent then issues a command that moves it towards that strategic adjusted positioning. This behavior enables the team to move similarly to a real soccer team, keeping the ball well covered while the team remains distributed along the field. Fig. 1 (Comes from Reis et al., 2001(a)) illustrates the graphical schematic of strategic positioning which described in SBSP method.

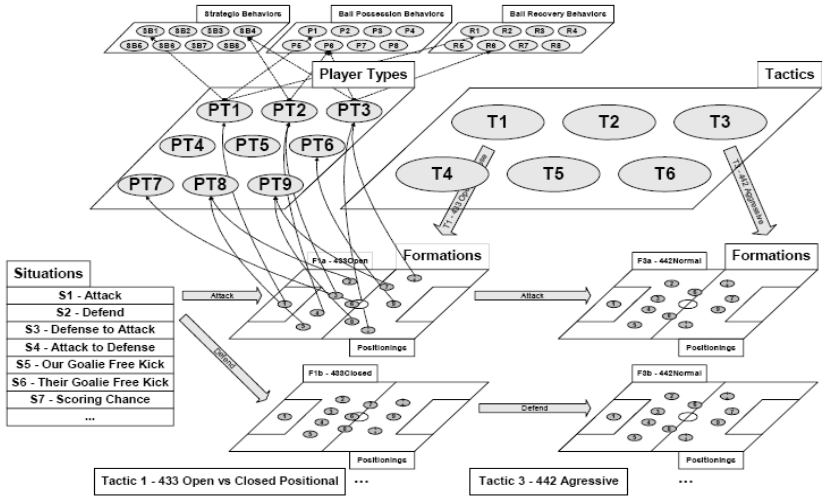


Fig. 1. Team strategy definition example

2.2 Dynamic Positioning

The goal of dynamic positioning is to let the agents to switch their positioning inside a given tactic and formation whenever that action leads to an improvement of the team global utility. Each agent has an allocated position inside the current formation that changes dynamically with the competition specific situation. Due to the dynamic positioning Situation Based Strategic Positioning for Coordinating Homogeneous Agents and role exchange mechanism agents do not have fixed positioning inside the formation.

For example, agent 2 can be at positioning 2 at a given time and at positioning 9 a few moments later.

There are many different implementations of Dynamic Positioning including the approach of FCPORTUGAL which was based on SBSP but here we will discuss the newest approach of Dynamically Positioning presented by ZJUBase team in Robocup2005 (Hao et al., 2006).

ZJUBase dynamic positioning method used NURB curves (Schneider 1996) to describe the strategic movement of players at a certain moment. In this method a parametric function on the balls position is defined to calculate the players' position. Here an important motivation for employing the NURB curves is the ability to control smoothness and the convenience. For example, Some spots B^i arbitrarily placed as control points, and the curve drawn with a NURB function, as shown in Fig. 2-a. Then in Fig. 2-b, The point B7 moves which its motion makes changes on curvature. So in order to get the required curve we only need to place and adjust the control points. This feature enables us to build a graphical editor and get much easier to adopt the positioning strategy. The applied function can be expressed by equation 1.

$$Q(u) = \frac{\sum B_i \omega_i N_{i,k}(u)}{\sum \omega_i N_{i,k}(u)} \tag{1}$$

Where B^i is the projection of one of four-dimensional control point and the ω^i is its weight. Equation 2 shows the N (u) function.

$$N_{i,0}(u) = \begin{cases} 1 & t_{i \le u < t_{i+1}} \\ 0 & otherwise \end{cases} \tag{2}$$

$$N_{i,k}(u) = \frac{(u - t_i) N_{i,k-1}(u)}{t_{i+k} - t_i} + \frac{(t_{i+k-1} - u) N_{i+1,k-1}(u)}{t_{i+k+1} - t_{i+1}}$$

Here it is the conventional notation for the i'th knot in the knots vector [Fig. 2] and k is the order of the curve. In this function these parameters are determinate. The three equations are based on a NURB curve. To learn more about the NURB curves, please refer to some related books or papers, such as (Schneider 1996).

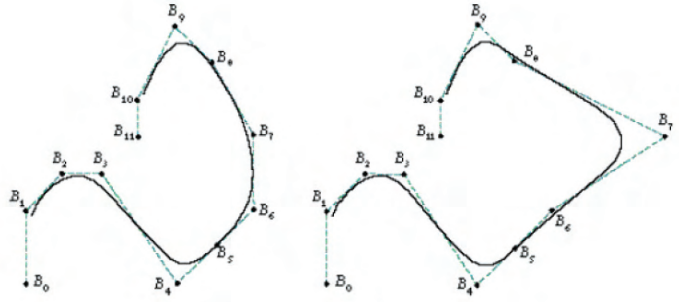


Fig. 2. Defining a curve with control points (Comes from (Schneider 1996))

2.3 Positioning using machine learning

For everyone with a background in artificial intelligence it is not a surprise to see that machine learning approaches are able to get good results in different complex aspects of robot soccer. Here we will describe a reinforcement learning approach which implemented by the BrainStormers team (Riedmiller et al., 2005) at 2D league of robocup2005 competition:

The key idea behind this approach is to learn a central value function V (s) for all players that describe how is a desirable situation. In other words, V (s) is a mapping from a state s to a value in [−1, 1]. A value close to 1.0 indicate that this situation is close to success (goal), a value near -1.0 means that it is very probable to loose the ball in that situation.

In this approach a situation consists of ball position and velocity plus the position of all attacking teammates and all defending opponents. The number of teammates and opponents that are used in the state representation for the value function has to be fixed beforehand.

The learning is done in epochs. In the beginning, the value function is initialized randomly so the players pursue a random strategy. Now the players play according to that strategy until five successful trajectories have been collected. If a trajectory was unsuccessful (e.g. loss of the ball) it is also stored. An example set E consisting of situations and rewards is generated from these five successful plus maximal five unsuccessful trajectories. The terminal state S_n of a trajectory gets a reward of 1.0 ($V(S_n) = 1.0$) if it is a successful terminal state and a reward of -1.0 ($V(S_n) = -1.0$) otherwise. The reward of the other states in the trajectory depends on the distance from the terminal state. Let $[S_1, S_2, S_3, \ldots, S_n]$ be the states encountered on a trajectory. The equation 3 computes these states' value.

$$V(s_i) = decay^{(n-i)} * V(s_n) \quad i=1,\ldots,n-1 \tag{3}$$

The action set for a player without the ball is very simple and consists only of moving to different positions relative to the current player position.

It is clear that such players' arrangement yield to players arbitrary moving, to handle this condition Riedmiller et al. like SBSP used the concept of home position.

A player without ball can choose an action from the following actions types (Actions that moves the player out of his home area is not allowed):

– go in one of eight directions from current position.

- go to one of eight positions around the players' home position.
- go to home position.

3. Centroidal Voronoi Diagrams

Since 1908 when Voronoi Tesselations were formally defined and studied by Russian Mathmatician Georgy Voronoi, these diagrams have found numerous applications in various fields of study including Physics, Biology, Chemistry, etc. To see a list of application ,see (de Berg et al., 2000) .
One of the applications of Voronoi Tessellations can be in Robocup Positioning.
Consider that you are given the positions of a set of agents and these agents are to cover the field in order to have some sort of control on whole the field. This control can be defined as a reasonable distance to the ball if the ball is put randomly on some point. To handle this situation, it is useful to partition the field into areas of influence in a way that each agent is responsible to cover one area. The area assigned to each agents can be defined as the set of the points to whom the agent is the nearest agent. This leads to the formal definition of Voronoi Diagram:
Defenition 1:
Given a set of points $\{S_1, S_2, .., S_n\}$, the Voronoi cell V_i corresponding to the point S_i is defined as the set: $Vi = \{ X \mid |X - Si| < |X - Zj| \text{ for } j = 1, \ldots, n, \; j \neq i \}$. The points $\{\mathbf{S}i\}^{n}_{i=1}$ are called generators or sites and the resulted tessleation is called Voronoi Tesselation or Voronoi Diagram.
Modeling the agent's positions as the Voronoi Sites has some advantages that would be discussed later. First we review some easy facts about Voronoi Diagrams. You can find most of the proofs in (de Berg et al., 2000).
Observation 1:
V_i (Voronoi Cell corresponding to the i'th site) is the intersection of n-1 lines and hence, an open convex polygon region bounded by at most n-1 vertices called *Voronoi Vertices* and at most n-1 edges called *Voronoi Edges*.
Observation 2:
Since Voronoi Tesselation applies on whole the plane, some Voronoi Cells would be unbounded, however in most of the applications it is usefule to define a *bounding box* containig all sites and Voronoi Vertices. (Fig. 3)

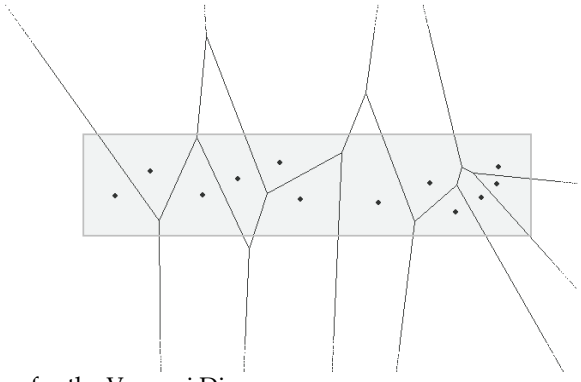


Fig. 3. a bounding box for the Voronoi Diagram

Observation 3:
Ω(nlogn) is a lower bound for computing the Voronoi Diagrams. This can be shown by reducing the problem of sorting to the problem of computing Voronoi Diagrams. Also there are some algorithm achieving this bound, among them the most popular is *Fortune Algorithm.* Fortune Algorithm uses a sweep line method for computing Voronoi Diagram. However for the problem of positioing in Robocup, since there are just a small number of sites, the time complexity is not a matter and a simpler algorithm can be more effective.
Observation 1 provides a very simple method for computing the voronoi diagram in a bounded box. For computing the cell of agent *i*, It is enough to start with the entire box and cut it n-1 times with bisectors of site i with n-1 other sites. (Fig. 4)

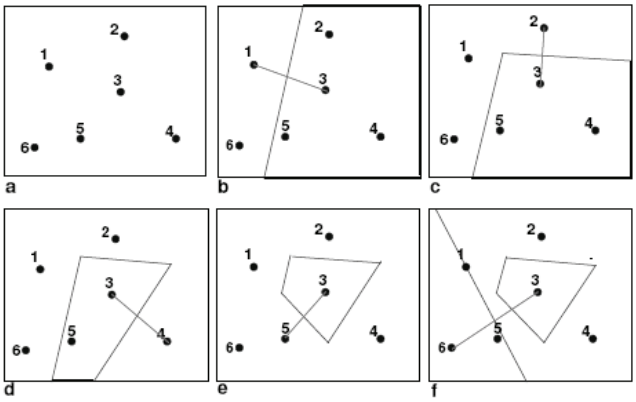


Fig. 4. A simple Algorithm for Computing Voronoi Diagram

3.1 Centroidal Voronoi Diagrams

Centroidal Voronoi Diagram are an interesting type of Voronoi Diagrams having several applications in to problems in image compression, finite difference methods, distribution of resources, cellular biology, statistics, and even the territorial behavior of animals. For a detailed discusision of these applications See (Du et al., 1999).
In this section we study Centroidal Voronoi Diagram as a rather ideal dynamic formation of agents in the field.
The *centroid* of an object *X* in 2-dimensional space is the intersection of all lines that divide *X* into two parts of equal moment about the line. Informally, it is the "average" of all points of *X*.
The x-coordinate of the centroid of an object can be calculated as the integral $\int x.f(x)\,dx \,/\, \int f(x)\,dx$, where $f(x)$ is the vertical extent of the object at abscissa x. In this way it is rather easy to find the centroid of an object.
The geometric centroid of a physical object coincides with its *center of mass* if the object has uniform density. So some times we refer the 'centroid' as the 'center of mass' or shortly 'center'.
A *centroidal Voronoi diagram* is a Voronoi tessellation whose generating points (sites) are the centroids (centers of mass) of the corresponding Voronoi regions. Note that to define centroids for Voronoi Cells, all of them need to be bounded. As a result Centroidal Voronoi Diagrams are defined on bounded boxes.
Fig. 5 shows two samples of Centroidal Voronoi Diagram (from Du et al., 1999).

As the core property of the centroid, we can say that the centroid of an area as a set of points minimizes the weighted sum of the Euclidean distances from the points to any point in the plane. (Abdi. 2007)
As a result, the centroid of the Voronoi Cell is a good position that an agent can take. In this way it can have better control on its cell. For example, putting the ball in a random position in the cell, the expected value of the distance to ball is mimized in the centroid. So trying to achieve a Centroidal Voronoi Diagram can be a good idea for the positioing of agents.

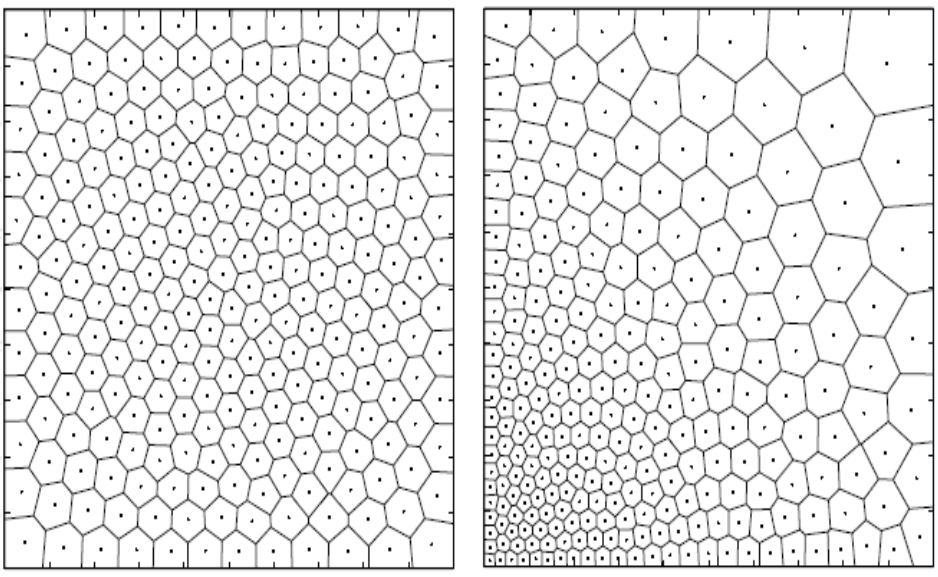


Fig. 5. Two samples of Centroidal Voronoi Diagrams (with 256 sites)

3.2- Lloyds Method

There exist both probabilistic and deterministic approaches for achieving Centroidal Voronoi Diagram. One of the determinstic methods is *Lloyd*'s Method. [See (Du et al., 1999) for probabilistic approaches]. This method was first presented by Stuart Lloyd in (Lloyd. 1982). There are some variations on Lloyd's method; However generally it can be described as follow:

0. Select an initial set of *n* points (Voronoi Sites).
1. Construct the Voronoi Diagram associated with the points.
2. Compute the mass centroids of the Voronoi regions found in step 1; these centroids are the new set of sites.
3. If this new set of sites meets some convergence criterion, terminate; otherwise, return to step 1.

The tremination procedure in step 3 is very depnedant on the specific applicaton.

Fig. 6 show the behaviour of Lloyd's methods[1]. As you can see, this method can be described as a set of *relaxations rounds*. After each round, the points are left in a slightly more even distribution: closely spaced points move further apart, and widely spaced points move closer together. Although the final shape of the diagram is a function of initial formation of points, the result has always an even distribution.
Note that defining a termination procedure is important since after each round the method convergence rate decreases[2]. (See Fig. 6)

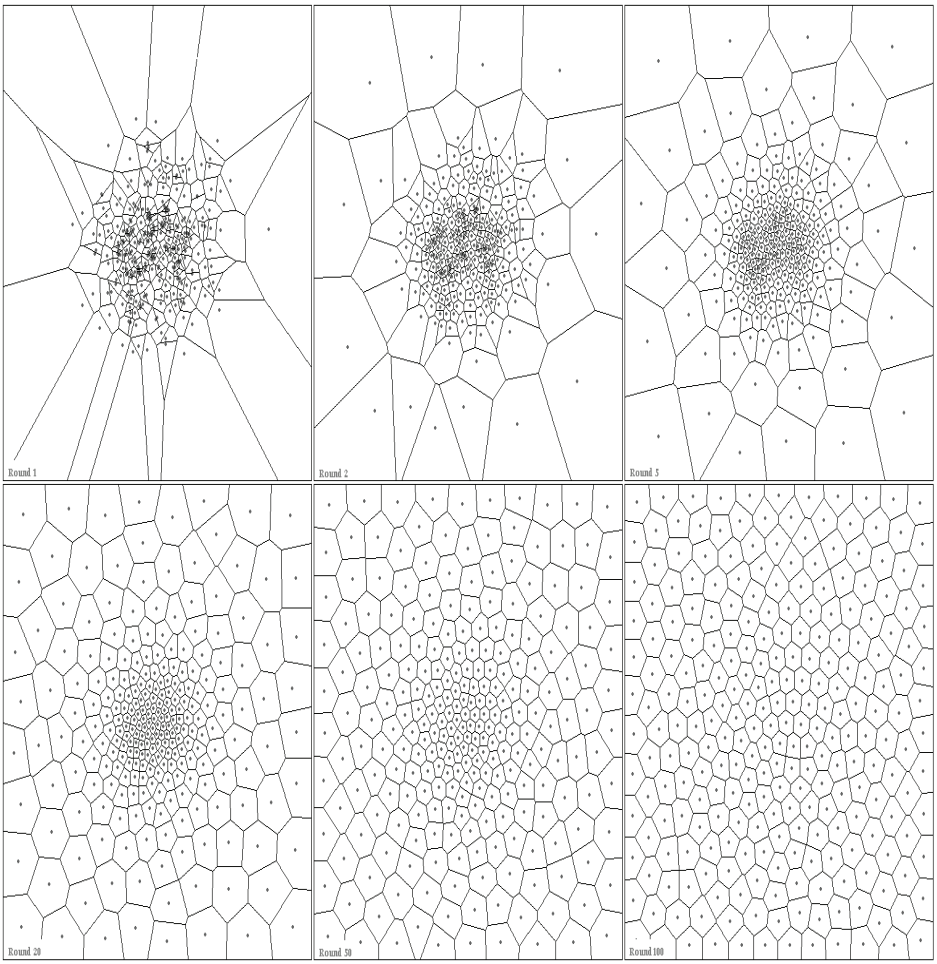


Fig. 6. Applying Lloyd's method on a set of 256 points. You see rounds 1, 2, 5, 20, 50, 100. The points are initially distributed normally in range [0..1] with mean=0.5 and variance=0.1

[1] The code for generating these slides is available upon request.
[2] The 1-dimension version of the Lloyd's method is proved to converge. The 2-dimension version is also conjectured to converge but no proof exists yet.

4. Dynamic Positioning based on Voronoi Cells (DPVC)

As mentioned before Dynamic Positioning based on Voronoi Cells is a Positioning method which models the agents (team mates) as the Voronoi Sites and uses a variation of Lloyd's method[3] to slightly distribute them in the field. Such modeling has the advantages of a dynamic positioning; for example there is no need for defining home positions for agents. Also as we will see it is very easy to apply futures of the soccer world to this model.

In DPVC, like the original version of Lloyd's method, in each sense (relaxation round), agents construct their Voronoi Cells and the center of such cell. Then, each agent should be replaced by the center of its cell, however since the agents movement is continuous they may not be able to get their center in one sense. So rather than 'putting' the agent into its center, we force the agent toward the center of its cell. Such force would be applied to the agent through a force vector called *Voronoi Vector*.

Fig. 7 shows the movement of agents (following a set of senses) while applying Voronoi Vectors. Like the original version of Lloyd's algorithm, applying Voronoi Vectors on agents causes them to repulse their team mates and cover the free spaces in the field. Also as you can compare Fig. 6 and Fig. 7, the convergence rate of this method is not worst than the original version of Lloyd's method[4].

It is believed that this variation of Lloyd's method, i.e. relaxing the diagram by approximating sites (agents) toward their cells' centers, converges to a centroidal Voronoi Diagram. Here we can define the convergence criterion to be a threshhold on the size of Voronoi Vector, i.e. we stop the algorithm if the size of Voronoi Vector of all agents gets smaller than a constant value ε.

Fig. 8 show the convergence of agents in Robocup soccer simulation.

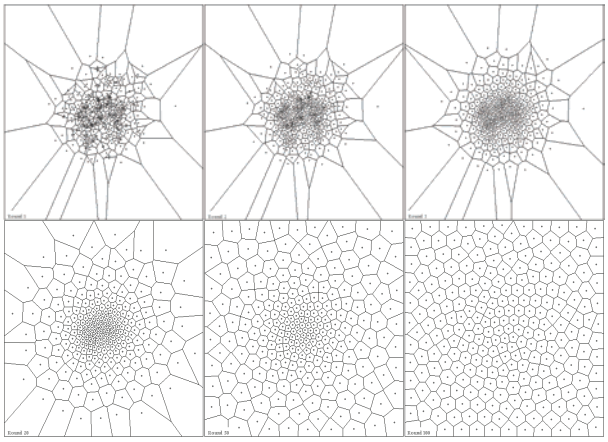


Fig. 7. applying DPVC on the same set of points of Fig. 6. The points have continuous movement. You see rounds 1, 2, 5, 20, 50, 100. The agents' velocity is 0.01 per round

[3] For the first time that DPVC was appeared in (Dashti. 2006) the authors were not aware of Lloyd's Method. However because of the similarities between two methods, it is better to regard the backbone of DPVC as a variation of Lloyd's method.

[4] Obviously the convergence rate of DPVC deeply depends on the (maximum) velocity of agents per round.

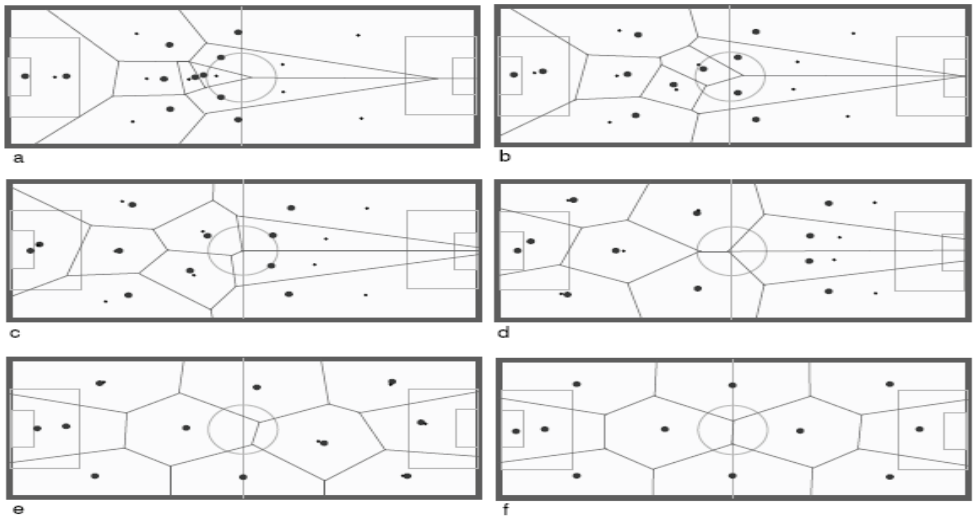


Fig. 8. Applying DPVC (without strategy) in Robocup Soccer Simulator Server 3D

4.1. Applying Strategy

Up to now what studied was a *plain* version of DPVC (Fig. 8) in which the Voronoi Vectors are just functions of the positions of team mate agents. We can easily apply other futures like opponents' positions, agents' batteries, team strategy, etc. to implement the team's strategy:

Opponents' positions

Considering game situation, it is wise to distribute some agents in the open spaces between opponents to make passes more successful. Adding opponents' positions to the Voronoi sites, the Voronoi Vectors would be constructed in the way of making agents move toward opponent team's openings. In this way agents would have a kind of repulsion to both opponents and teammates. (Fig. 9)

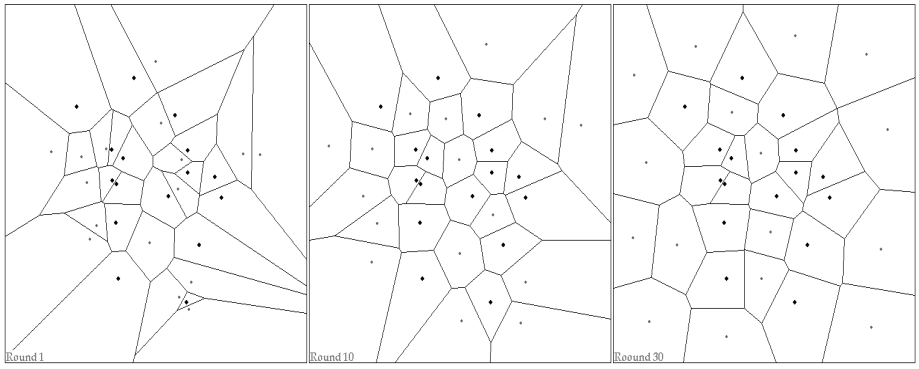


Fig. 9. Applying Opponents positions in DPVC. Opponent sites are bolded and fixed during the relaxation rounds. You see rounds 1, 10, 30. The team mates get positions in the openings between opponents

Agents' Stamina:
The *additively weighted Voronoi Diagram*[5] is a generalization of Voronoi Diagram in which a *weight* is assigned to each Voronoi site and while measuring distances to a site, the weight of the site is added to the usual Euclidian distance. Fig. 10 shows a sample of additively Voronoi Diagram. As you can see the sites with smaller weights would have smaller cells. Also note that the boundaries between cells are segments of circles rather than straight lines.
Since the agents with smaller cells would have smaller Voronoi Vector, we can model the agents' stamina by the weight of Voronoi sites, in this way agents with lower battery would have smaller Voronoi Vectors and can save stamina in this way.

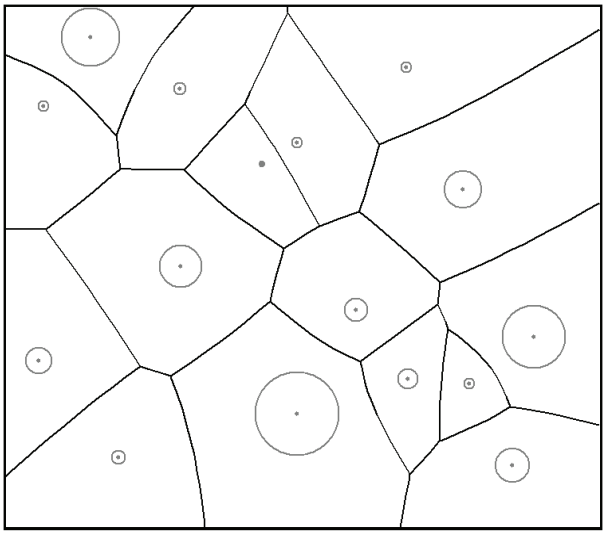


Fig. 10. weighted Voronoi diagram for a set of 16 sites. The radius of disks shows the weight of the sites

Concentration of agents
The Voronoi Bounding box is mainly defined as the whole filed. However Considering the game situation, it is wise to shrink this box to concentrate agents in an appropriate area. For example as Fig. 11 shows restriction on the width of box causes agents concentrate in an extreme side of the field which can be interoperated as a more defensive (or offensive) formation[6]. Also it is easy to see how this restricting on the width of box can be applied to form an Offside trap.
Note that if we use the same algorithm of section 3 for computing Voronoi Cells, the cells of all agents, even those that are out of the bounding box, would lie inside the box. So after several rounds all agents get into the box.

[5] There exist some other types of weighted Voronoi diagram such as "multiplicatively weighted Voronoi diagram", "additively weighted power Voronoi Diagram". See (Okabe et al., 2000) for more details.
[6] Getting an offensive of defensive formations can be a matter of ball position or other factors set by game strategy.

Other factors

It is desirable to apply all key factors of team strategy exclusively through Voronoi vectors (As we did about opponent's position, agents' batteries, Offside trap, and ball position). In this way the agent's movement would be much more stable and extra movements would be reduced. However it is not that easy to apply more complicated strategies just by Voronoi Vectors (for example in the case of collaborating with goalie or blocking opponents).

To handle these types of strategies, we need to define some extra force vectors called *Attraction Vectors*. These vectors are set by the team strategy considering the game situation and involve attractions to specific positions in the field. In this case the final force vector would be applied to the agent as a combination of the Voronoi Vector and Attraction Vectors[7].

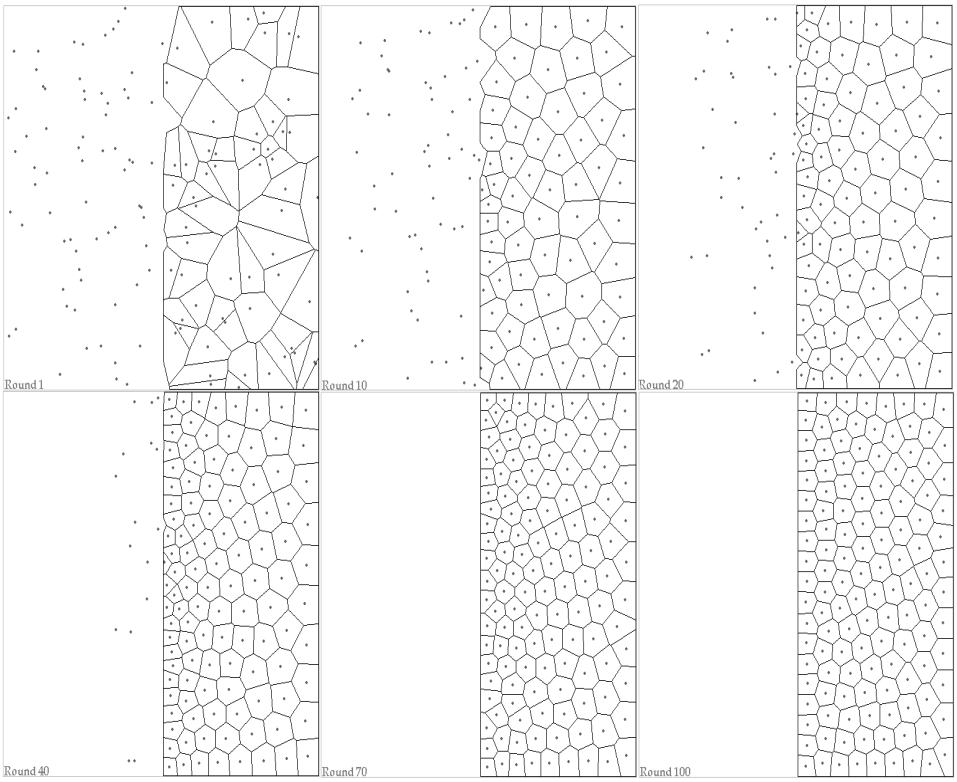


Fig. 11. Concentrating agents to one side of the field by restricting the Voronoi Bounding Box. You can see rounds 1, 10, 20, 40, 70, 100. In the beginning, there are 128 agents uniformly distributed in [0..1] × [0..1]. The bounding box is a [0..5] × [0..1] rectangle and the agents velocity is 0.01

[7] Here we do not go deep to the concept of attraction Vectors since they are very much dependant on the specific team strategy

5. Experimental Results

5.1- Statistical Experiments

Performance of a team not only depends on its positioning method but also depends on other decision modules and efficiency in implementation of agents skills. Accordingly it is difficult to define an appropriate criterion to evaluate the positioning method. In order to survey the applied positioning, we compared two similar teams using different positioning methods in Robocup soccer simulation 3-D. One of these teams uses SBSP as positioning method and the other uses DPVC. Since there is usually little density of players near corners of the field, we improved DPVC by restricting the initial Voronoi Cells of agents to a hexagonal surrounded by the entire field rectangle.

To compare these two positioning methods we prepared two experiments. In these experiments both teams play against Aria team the champion of Robocup 2004 3D competitions. In the first experiment the number of passable players around the ball when the ball is in possession of the testing team is measured.

Passable player is a player who has the opportunity to get the ball if it is passed. So being a passable player is a matter of player's distance from the ball. In our experiment a player is defined to be passable if its distance from ball is less than 20 meters. Figure 12 is a statistical diagram of passable players of the team. In Figure 12-a DPVC is used as the positioning method, whereas in Figure 12-b the positioning is based on SBSP.

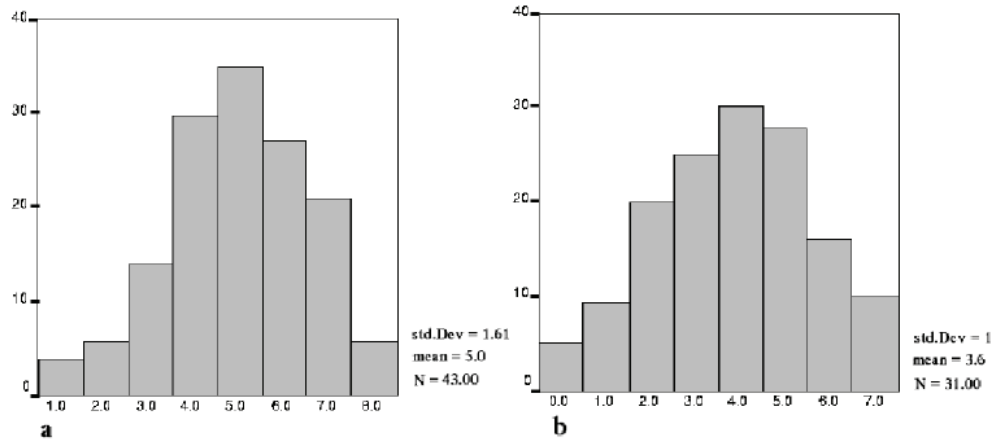


Fig. 12. The figure shows statistical distribution for average number of passable players of the team against Aria, using DPVC and SBSP methods for positioning. N is number of the team opportunities to pass the ball. Numbers in x-axis show number of passable players where numbers in y-axis show the times that these numbers occur. As it is shown while using DPVC in average there are five passable players whereas by using SBSP there are 3.6 passable players

In the second experiment both the team using DPVC and the team using SBSP are ran against Aria 10 times. Table 1 reports results of these two series of tests. Records of this table show parameters defined to compare the influence of each method of positioning on success of the team.

	DPVC	SBSP
Average number of passable players	4.09	3.36
Ball in control of the team	58.31%	56.92%
Ball in own half	18.80%	22.15%
Ball in own penalty area	1.47%	1.26%
Ball in opponent penalty area	23.02%	19.98%

Table 1. Results when two similar teams using different positioning methods (DPVC and SBSP) play against Aria team

5.2 Experiments at Robocup World Championship

For the first time DPVC was implemented by UTUtd (Dashti et al., 2006) at Robocup world championship 2005 at Osaka. UTUtd after playing 21 matches with 13 wins 6 draws and 2 losses reached the 5'Th place among 32 teams in the 3D simulation league. Next year at Robocup world championship in Bremen DPVC was implemented by Virtual Werder (Lattner et al., 2006). In this implementation of DPVC players had little attraction to ball and had a defensive Formation. Virtual Werder team could rank up to 8 between 32 teams after 5 wins, 17 draws and only 1 loss. As it is seen below the number of losses in VW team is much lower than the second and third team.

Rank	Team Name	Won	Draw	Lost
1	FCPortugal3D	19	2	0
2	WrightEagle	10	7	4
3	ZJUBase	12	6	3
8	VW3D	5	17	1

Table 2. Results from Robocup world championship 2006

References

Abdi, H. (2007) Centroid, center of gravity, center of mass, barycenter. In N.J. Salkind (Ed.): *Encyclopedia of Measurement and Statistics*, Sage, 1412916119, Thousand Oaks, California.

Dashti, H.; Aghaeepour, N.; Asadi, S.; Bastani ,M.; Delafkar, Z.; Disfani. F.M.; Ghaderi , S.; Kamali ,S.; Pashami, S. & Siahpirani ,A. (2006). Dynamic Positioning Based on Voronoi Cells (DPVC), *Proceedings of RoboCup international Symposium 2005,* pp. 219-229, 3-540-35437-9, Osaka, Japan, July 2005, Springer Berlin Heidelberg.

De Berg, M;. van Kreveld, M.; Overmars, M. & Schwarzkopf, O. (2000) *Computational Geometry:Algorithms and Applications*, Springer-Verlag, 3-540-65620-0, Berlin Heidelberg New York.

Du, Q.; Faber, V. & Gunzburger, M. (1999). Centroidal Voronoi Tessellations: Applications and Algorithms, *SIAM Journal on Applied Mathematics,* vol 41, no.4, pp.637-676, 1095-712X.

Hao, J.; Xinfeng, D.; Dijun L.; Yifeng, Z.; Yang Z., (2006) ZJUBase 3D - Team Description 2006, *The 10th RoboCup International Symposium,Bremen, Germany, http://www.nlict.zju.edu.cn/nlictrobocup/download/zjubase_3d_3dd_2006.zip*.

Lattner, A. D.; Planthaber, S.; Rachuy, C.; Stahlbock, A.; Visser, U.; Warden T., (2006) Virtual Werder 3D 2006 Team Description, *The 10th RoboCup International Symposium, Bremen, German, http://www.tzi.de/fileadmin/resources/publikationen/tzi_berichte/TZI-Bericht-Nr._36.pdf.*

Lloyd, S. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129-137, ISSN: 0018-9448.

[a] Reis, L. P.; Lau, N.; Oliveira, E. C., (2001) Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents, From Robocup to Real-World Applications, *Springer's Lecture Notes in Artificial Intelligence,* Vol. 2103, pp. 175-197, Berlin.

[b] Reis, L. P.; Lau N., (2001) FC Portugal Team Description: Robocup 2000 Simulation League Champion, In *The 4th Robocup International Symposium*, Melbourne, Australia, *Springer's Lecture note in Computer Science*, Volume 2019, pp. 29, 2001.

Riedmiller, M.; Gabel, T; Knabe, J.; Strasdat H., (2005) Brainstormers 2D – Team Description 2005, *The 9th Robocup International Symposium, http://panmental.de/papers/BSTeam05.pdf*.

Schneider, P. J, (1996) NURB Curves: A Guide for the Uninitiated. *The journal of Macintosh technology*, Issue 25, June 1996.

3

Non-monotonic Reasoning on Board a Sony AIBO

David Billington, Vladimir Estivill-Castro, René Hexel, and Andrew Rock
Griffith University
Australia

1. Introduction

Robots are today a reality. Moreover, robots have moved from assembly lines to being around human beings. Mobile autonomous robots are now a common sight in Korean airports. Other notable examples are LEGO's Mindstorms and Spybotics, who not only have a massive penetration in the toy market, but have penetrated the research and academic environment (Wallich, 2001). Robots are also being sold commercially as companions, or used as museum guides (Thrun *et al.*, 1999), and even as the long awaited vacuum cleaner (Kahney, 2003). The expectation that robots would be around us inspired Isaac Asimov to write "I Robot" as part of a series of books and to develop the character Susan Calvin who enunciated the Three Laws of Robotics:

1. A robot may not injure a human being, or, through inaction, allow a human to come to harm.
2. A robot must obey orders given to him by human beings except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

To follow these rules, a robot would need to reason about actions and their potential effect. Reasoning is a fundamental capability of intelligent systems and much progress has been made (Marek & Truszczynski, 1993; Rich & Knight, 1990) (this is also illustrated by the 4 chapters dedicated to uncertain knowledge and reasoning in (Russell & Norvig, 2002), at present the most widely accepted textbook in Artificial Intelligence and 57th most cited computer science publication ever). Most notably, for intelligent and robotic systems it is essential that such reasoning be capable of withdrawing some conclusion in the light of new evidence (including the negation of what used to be considered a fact). This is called non-monotonic reasoning.

This chapter will describe how a Sony AIBO performed on board non-monotonic inferences on two settings present on the RoboCup competition (Veloso *et al.*, 1998). Robotic soccer is the most challenging endeavor from the perspective of multi-agent technology (Wooldridge,

2002). We argue that non-monotonic reasoning is useful in even the dynamic,[1] inaccessible,[2] adversarial and non-deterministic[3] environment of robotic soccer, where reactive systems have received much attention. We also present an example for the RoboCup@Home setting. We achieve this common sense behavior by an implementation of Plausible Logic (and some algorithm fine-tuning) in C++.

We will commence the discussion by applying Plausible Logic inferences to make sense of the sightings in the configuration of the 2005/2006 4-legged league field of play. In particular, we show that we can analyze the objects reported by the vision module in a frame (or in a sequence of frames) and determine which were phantom sightings and which could actually be valid sightings. This assists localization as it dynamically selects proper inputs (landmarks). Localization means that the software in the robot must gather information from its sensors and arrive at a reasonable (and accurate) conclusion about its location and orientation. We argue that there is a role to be played by reasoning when localizing, with a loose analogy to when people use previous knowledge to explore an environment they have some information about. Our problem is different from a pure SLAM problem. Here, we assume some previous knowledge of the environment, so we can reason about and contrast our observations with our prior knowledge.

There are many algorithms to deal with the error in odometry as well as errors in other inputs for localization (vision or laser sensors). These usually fall into three main categories (the family of Kalman Filters (KF), the family of Markov Models (MM) and the family of Monte Carlo (MCL) localizers). We do not advocate their elimination. In fact, we use the Monte Carlo Localization approach for localizing Sony AIBO robots in the 4-legged league of RoboCup. However, we introduce non-monotonic reasoning as a filter before the localization process takes observations as inputs.

We believe our approach offers an alternative to the problem of *data fusion*. For data fusion, probabilistic models are favored over reasoning with logic models. For example, for combining information from several sources, their reliability is modeled using probabilities and "reasoning with uncertainty" is performed using general models that include applications to sensor fusion (Haemmi & Hartmann, 2006, and references therein). As sensors become more sophisticated, and as entire modules on board a robot collect information about the environment, reasoning is essential to integrate and comprehend such sources of information (some could come from observations by teammates). Non-monotonic reasoning would be most effective when the information is contradictory. In fact, contradictory information is actually the common case. At the sensor level, an odometry sensor will usually be in disagreement with the distance computed by using projective geometry and trigonometric equations from the images of a digital camera, and these will also exhibit differences with an infrared-range sensor. Information from teammates has a lag in time.

---

[1]The environment is dynamic if it will evolve in the time gap between the sensors collecting information and the agent performing an action (Wooldridge, 2002).

[2]Information about the entire environment may not be possible to collect (Wooldridge, 2002).

[3]An environment is non-deterministic if an action may not have the expected outcome (Wooldridge, 2002), like a skid because the surface is smoother than anticipated.

We propose to use non-monotonic reasoning to accept the inconsistent information and resolve it to obtain the most plausible interpretation of the state of a robot and its environment. Identifying this state is clearly a crucial initial step towards making a decision and then acting. We also want some assessment of the likelihood of that state for the decision making process. The influential works by Brooks (Brooks, 1991) have lessened the interest in using symbolic/logic approaches. We argue here that a computable non-monotonic logic has a role to play.

Our systems have been implemented and operated by the Mi-PAL team in RoboCup (2005, 2006, and 2007). The current robotic platform is the Sony AIBO robot.

2. Background on Plausible Logic

Non-monotonic reasoning (Antoniou, 1997) is the capacity to make inferences from a database of beliefs and to correct those as new information arrives that make previous conclusions invalid. Although several non-monotonic formalisms have been proposed (Antoniou, 1997), *Plausible Logic* (PL) (Billington & Rock, 2001; Rock & Billington, 2000) is currently the only one with an efficient non-looping algorithm (Billington, 2005). Another very important aspect of PL is that it distinguishes between formulas proved using only factual information and those using plausible information. PL allows formulas to be proved using a variety of algorithms, each providing a certain degree of trust in the conclusion. Because PL uses different algorithms, it can handle a closed world assumption (where not telling a fact implies the fact is false) as well as the open world assumption in which not being told a fact means that nothing is known about that fact. The β algorithm for PL uses the closed world assumption while the π algorithm uses the open world assumption.

If only factual information is used, PL essentially becomes classical propositional logic. However, when determining the provability[4] of a formula, the algorithms in PL can deliver three values (that is, it is a three-valued logic). The proving algorithms terminate assigning the value +1 to a formula that has been proved and -1 to a formula that has been disproved. It assigns the value 0 when the formula cannot be proved and attempting so would cause infinite recursive looping.

In PL all information is represented by three kinds of rules and a priority relation between those rules. Strict rules, denoted by the strict arrow → and used to model facts that are certain. For a rule $A \rightarrow l$ we should understand that if all literals in A are proved then we can deduce l (this is simply ordinary implication). A situation like *Humans are mammals* will be encoded as $human(x) \rightarrow mammal(x)$.

Plausible rules $A \Rightarrow l$ use the plausible arrow ⇒ to represent a plausible situation. If we have no evidence against l, then A is sufficient evidence for concluding l. For example, we write *Birds usually fly* as $bird(x) \Rightarrow fly(x)$. The intent is to record that when we find a bird we may conclude that it flies unless there is evidence that it may not fly (like knowing it is a penguin).

Defeater rules $A \sim\!> \neg l$ mean that if A is not disproved, then it is too risky to conclude l. An example is *Sick birds might not fly* which is encoded as $\{sick(x), bird(x)\} \sim\!> \neg fly(x)$. Defeater

[4]Provability here means determining if the formula can be verified/proved.

rules prevent conclusions that would otherwise be too risky. This could happen in a chain of conclusions from plausible rules.

Finally, a priority relation > between rules $R_1 > R_2$ indicates that R_1 should be used instead of R_2. In this chapter we actually demonstrate the expressive power of this aspect of the formalism. For example from

$$\begin{aligned} \{\} &\rightarrow quail(Quin) && \textit{Quin is a quail} \\ quail(x) &\rightarrow bird(x) && \textit{Quails are birds} \\ R_1{:}bird(x) &\Rightarrow fly(x) && \textit{Birds usually fly} \end{aligned}$$

one would logically accept that *Quin* usually flies. From the knowledge base

$$\begin{aligned} \{\} &\rightarrow quail(Quin) && \textit{Quin is a quail} \\ quail(x) &\rightarrow bird(x) && \textit{Quails are birds} \\ R_2{:}quail(x) &\Rightarrow \neg fly(x) && \textit{Quails usually do not fly} \end{aligned}$$

we would reach the correct conclusion that Quin usually does not fly. But what if both knowledge bases are correct, that is both rules R_1 and R_2 are valid. We see that R_2 is more specific than R_1 and so we add $R_1 > R_2$ to a knowledge base representing the beliefs of a robot that knows both. Then PL allows the agent to reach the proper conclusion that Quin usually does not fly, while if it finds another bird that is not a quail, the agent would accept that it flies.

Note that the Three Laws of Robotics are an example of how humans describe a model. They define a general rule, and the next rule is a refinement. Further rules down the list continue to polish the description. This style of development is not only natural, but allows incremental refinement. Indeed, the knowledge elicitation mechanism known as *Ripple Down Rules* (Compton & Jansen, 1990) extracts knowledge from human experts by refining a previous model by identifying the rule that needs to be expanded by detailing it more. The models presented here are each a progressive refinement of the previous one.

3. Plausible Logic for Localization

Non-monotonic reasoning has long been considered too complex for real-time environments. Visual robot localization in the 4-legged league places particularly stringent demands on the processor. Video camera systems typically operate at a rate of 30 frames per second, which allows only about 30 ms to perform full image recognition, feature extraction, and consistency verification. Moreover, poor lighting conditions can make color calibration extremely difficult. More often than not, vision systems make errors in object recognition (in particular, they may occasionally miss the landmarks for localization or report non-existent objects as visible). In this section we present our first application of PL. We use it to make sense of the sightings in a scene before they are used as landmarks for localization.

3.1 The Problem

The most well known family of techniques to interpret the input provided by a sensor with some noise is derived from the *Kalman Filter* after the 1960's publication by R.E. Kalman describing a recursive solution to the discrete-data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman Filter has been the subject of

extensive research and application, particularly in the area of autonomous or assisted navigation.

In robot localization, alternative techniques have emerged. Most notably, grid-based Markov localization and Monte Carlo localization (Fox *et al.*, 1999; Gutmann & Fox, 2002; Thrun *et al.*, 2001). These techniques are based on a paradigm that still uses probability distributions. The manipulation of the probabilistic representation is slightly different across these schemes. While the Kalman filter (KF) uses some mathematically defined probabilistic model (usually multivariate Gaussian distributions), the Markov model (MM) represents a distribution as a histogram (one could say Kalman is using parametric statistics while Markov localization uses non-parametric statistics). On the other hand, Monte Carlo localization (MCL) represents the probability distribution by a population of weighted samples (also close to a non-parametric model of the distribution) but rather than representing the distribution by piece-wise values on a grid, Monte Carlo uses a population of cases. Fundamentally, the three approaches update the current belief using Bayes theorem to incorporate the knowledge from a sensor and to update the current belief. They use conditional probabilities to represent the prior knowledge and posterior knowledge of the state of the world. In an autonomous mobile agent, the belief is revised by an observation as well as by an action. While the non-parametric schemes seem better equipped to deal with some of the performance issues of Kalman filters, and resolve some data fusion issues, they still are not able to rule out inconsistencies. For example, a phantom object in a frame can create a bump in the distribution that will be removed after many new consistent observations.

In particular, an example is a frame where the vision module reports two objects as visible, even though it is not possible for these objects to appear together in the same frame. In the case of Robotic soccer for the 4-legged league, this would be illustrated by the vision system seeing the opponent's goal as well as their own goal within the same frame. The localization approaches need to estimate *Prob*(visible scene| $\vec{pos}$), where the visible scene is a description of all visible objects, and vector $\vec{pos}$ is the current belief. To avoid describing probabilities for all possible scenes, one approach is to regard some observations as independent and modify the current belief by the product of $Prob(See_front_goal \mid \vec{pos})$ and $Prob(See_back_goal \mid \vec{pos})$ (for example when seeing both goals). The problem with this is that because $\vec{pos}$, has significant error regarding the orientation and pan of the head of the Sony AIBO, both of these probabilities are unlikely to be zero in any reasonable sensor model. This would result in creating a local mode in the probability distribution (in Markov and Monte-Carlo models) while creating a significant enlargement of the covariance matrix for the spatial Kalman filter. In fact, we know it is impossible to see both goals in the same frame, that is, we know $Prob(See_front_goal \wedge See_back_goal \mid \vec{pos}) = 0$, for all postures $\vec{pos}$. However, we just indicated that representing $Prob(See_front_goal \wedge See_back_goal \mid \vec{pos})$ as a function of $Prob(See_front_goal \mid \vec{pos})$ and $Prob(See_back_goal \mid \vec{pos})$ is rather complicated.

So the alternative is to generate a database of cases for these situations where domain knowledge allows us to plug in suitable values. The problem with this approach is that these cases become not only a few, but a rather large number. It then becomes hard to

ensure that this database of facts is accurate (or complete, or consistent). Furthermore, we must ensure that we are using this database to rule out observations at the right time. Because of the modeling assumptions in localization algorithms, it is important that the observations from sensors be as reliable as possible (otherwise, the convergence is too slow or the artifacts to handle the kidnap problem introduce other high modes in the representation of the distribution). In RoboCup most teams participating in the competition perform rules of thumb that fall in the realm of classical logic (sensible sanity checks); that is, they will filter out observations from the vision system that indicate that opposite goals were seen in the same frame. These *inconsistent*[5] inputs are simply, partially or not at all, used for localization.

The situation becomes difficult to manage, as entangled with the localization code is a series of logical tests that check special cases. Some code filters the observations that are considered inconsistent. This *consistency* module rapidly becomes a large piece of software, hard to verify for correctness or completeness. Our first thesis is that such a filter of inconsistent observations is better handled by some logic. The second thesis is that such a logic should not only be capable of ruling out observations, but allow reasoning about them to provide informative inputs to the localization module.

We have experimented with other alternatives (Billington *et al.*, 2005) to model the field of RoboCup 2005/2006[6] for the 4-legged league. These usually result in a complex description of the potential inconsistent inputs. In particular, it is very likely that most imperative object-oriented or procedural (in the case of the Sony AIBO C++) implementations of this, will result in at least incomplete models, and more seriously, deliver inconsistent models as they are usually developed incrementally as deeper and deeper nesting of `if-then-else` statements. Our analysis reflects that also some logic approaches rapidly result in a large number of rules. We believe most competing teams in RoboCup do not have a complete set of rules for handling, for example when vision detects four landmarks in a frame two of which are phantoms (blobs from the audience or off-field objects which fit the landmark characteristics but appear to vision as landmarks because of calibration or very similar color). Most teams survive this because these cases of many phantoms in one frame are reasonably rare in the constrained environment of labs or competition venues. However, they do pose a very serious threat to the correctness of their overall play (moreover, such faults become extremely difficult to reproduce and detect). The point we are making is that even from the software engineering, software verification and validation point of view, we need a complete and correct logic theory of the consistency of the vision reports.

Our example here analyzes the challenge of imperfect vision reports. That is, in a single frame, the analysis of an image may actually perceive two blobs of yellow color and one of blue that are rectangular enough for all of them to be considered as goals. Again, any software/logic that rules out two rectangular blobs of yellow, perhaps on the basis that one is larger than the other, or one is above the field of vision, or one is next to green, is performing some reasoning based on domain knowledge. What we are arguing here is that if all those ways of ruling out sightings of landmarks are not concentrated in a single place

[5]We use here the word *inconsistent* for an observation that is in some way in contradiction to what we expect from our knowledge of the domain. Note that we will use inconsistent theory or incomplete theory in the usual sense used by logicians.

[6]Previous versions of the field are very similar.

represented in logic, then the software is very likely to have such rules in several modules, resulting in high coupling of these, and more seriously, in incomplete and inconsistent modeling of the reasons why some sightings are ruled out before they are used for localization. As the robots move to more realistic environments more reasoning is needed.

3.2 Modeling with Plausible Logic

We introduce the modeling of consistent sightings incrementally. We start with a simple example but the point is not only to help the understanding of using non-monotonic logic, but to illustrate that in PL higher-level models are introduced incrementally as extensions of the previous model. This process allows us to model the most important (and most likely) scenarios upfront, while refining the models to handle the complexity of more specialized and sophisticated cases later. The execution of PL proofs on the AIBO is abstracted so that upgrading the model does not represent reprogramming the C++ code. The implementation of PL not only provides the algorithms for obtaining proofs, but provides a logic programming language DPL (Rock) for presenting facts, and describing a theory.

3.2.1 Model 1

We first need to represent the domain knowledge. Each 2005/2006 4-legged league soccer field has fundamentally 6 landmarks for localization. These are two goals (one yellow and the other blue) and four posts. Each post has two colors, and pink is always one of these. The two posts near the blue goal have blue as one of the colors while the two posts on the yellow side have yellow. This color-coding allows the identification of landmarks for a robot as Front Goal (FG), Back Goal (BG), Left Post (LP), Right Post (RP), Right Back Post (RBP) and Left Back Post (LBP). We also take advantage of the fact that although in 2005 the field has been enlarged, there are still some scenes that can be ruled out. The horizontal angle of view is 56.9°, but to simultaneously see LP and RBP would require a view greater that 67.5°.

First, the facts about the world are presented by type declarations in the logic programming language as follows. There are two goals.

```
type GoalType = {FG, BG}.
```

There are four posts.

```
type PostType = {LP, RP, RBP, LBP}.
```

A landmark is either a goal or a post, that is $Landmark=GoalType \cup PostType$. In the programming language we have

```
type Landmark = GoalType + PostType.
```

The next step is to define the inputs as predicates. In general, any piece of information we can retrieve from sensors or messages from teammates can be modeled as an input. Each input is introduced in the logic model as an axiom and these inputs trigger (fire) the plausible assumptions that appear in the model description. It is cumbersome to write, for every axiom a, the two plausible rules $a \Rightarrow p$ and $\neg a \Rightarrow \neg p$, where p is the plausible assumption to be fired by a. A macro of DPL simplifies this.

```
$=declareInput$(a$,p$)${$#
 input{$+a}. {$+a} => $+p. {~$+a} => ~($+p). $}$#
```

Now, we can write the plausible assumptions (this is what vision reports). First, *See*(*x*) will evaluate to true if and only if the vision module reports a sighting of landmark *x*.

```
type See(x <- Landmark).
```

This first model provides correct results only if the vision module reports exactly one landmark or none. Now, we are in a position to describe consistency rules. By default, when vision does not report a landmark, we do not forward anything to localization. This is an easy default case. This is R_1:{} $\Rightarrow \neg Cs(x)$ while in the programming language DPL we write

```
R1: => ~Cs(x).
```

However, if vision reports a landmark, we believe it; since for only one frame we have no other information to rule this out. PL writes this as R_2: $See(x) \Rightarrow Cs(x)$ $R_2 > R_1$. Note the relationship between rules. Now, the DPL equivalent is

```
R2: See(x) => Cs(x).  R2 > R1.
```

This works because we also provide a statement that any frame that has two or more landmarks should be ignored. This completes the programming of this simple model.

3.2.2 The Path to Implementation

Initially, the only implementation available of PL was in Haskell, and it was unclear that this implementation, even if translated to C++ would be fast enough to operate in the time slot for processing a vision frame (which is the usual time slot for doing all computation without losing frames, and thus possibly even losing sightings of critical objects like the ball). While originally (Billington *et al.*, 2005, 2006) we enabled PL without running the inference engine on the Sony AIBO, it became clear that this had computability limitations. This chapter focuses only on the scenario where the inference engine is operating on board the Sony AIBO. In order for PL proofs to be developed on board we had to extend the logic programming language DPL by adding automatic production of C++ macros and automatic production of gluing code. Also, we developed a template method and an architecture that makes model loading a component-replacement process. The PL was made to run on-board the Sony AIBO using a C implementation of the inference engine. The C implementation also ran on MAC-OS and LINUX since it used standard C-language constructs.

Extensions to DPL enable generation of C++ glue code. Namely, we took the decision we would read or evaluate sensor input only once. We would store the outcome of evaluating a plausible assumption in a C++ Boolean variable. For vision, for example, the C++ expression `FG` has value `true` iff the front goal is visible. This is an input axiom of the description that will be asserted either positively or negatively.

```
$+declareInput$("FG"$,See(FG)$)
```

Similarly, we have five more declarations. These input axioms are the atoms the logic will talk about and can be given initial values by the sensors of the robot (or the modules that read those sensors).

```
$+declareInput$("BG"$,See(BG)$)   $+declareInput$("RBP"$,See(RBP)$)
$+declareInput$("LP"$,See(LP)$)   $+declareInput$("LBP"$,See(LBP)$)
$+declareInput$("RP"$,See(RP)$)
```

An outcome of this is gluing code that declares the necessary Boolean variables in the corresponding C++ module (creates statements of the form `bool FG;`).

We proceed now to describe the special template method. The special template method consists of three phases. We already alluded to the first phase `INIT_ALL_FALSE()`. This is implemented as a macro that creates the necessary definitions of C++ Boolean variables for all input axioms. For example, we saw that the programming language enabled the declaration of an input.

```
$+declareInput$("FG"$,See(FG)$)
```

In the C++ code, there is a Boolean variable corresponding to this sensory input. In more elaborate models we will have additional inputs that indicate if one landmark is to the left of another landmark. These will also become Boolean C++ variables. Moreover, `INIT_ALL_FALSE()` not only defines those Boolean variables, but sets their values to false (C++ statements of the form `FG=false;`). This is just a default initialization. In temporal models that use information from a previous frame, the values of input axioms in one frame are copied to a corresponding frame-indexed set of C++ Boolean variables.

The second phase evaluates all input predicates (i.e. collects all information from the sensors), and stores this in the C++ Boolean variables. Its implementation is a macro `UPDATE_ALL()` that queries the values of the input axioms for the current frame. Also, if there are predicates that refer to sensor values in previous time slots, they become updated. We will say more about this when we discuss a model that analyses sightings across consecutive frames. In the current example, the macro obtains the values of input axioms from the vision module. For example, a variable for the front goal previously initialized to false may now be set to true if vision has found a front goal in the current frame (the most recent vision report will be extracted and since it reports the front goal as visible the C++ executes `FG=true;`). It also provides a pointer to such an object so other attributes about the landmark can be evaluated, e.g. its perceived size or whether it is seen to the left or right of another landmark in that frame.

The last phase of the template method is the invocation of the inference engine. `PLACE_CS_ALL()` will use the inference engine to evaluate the expressions for which we requested outputs. When we are filtering localization landmarks, if a landmark is found to be consistent (evaluates to true), the information on the landmark sighting will be forwarded to the localization module (or any other module that may benefit from it, such as the behaviour to kick when the front goal is visible). In particular `PLACE_CS_ALL()` will have as many `if` statements as landmarks, each with an expression that is a call to the inference engine. For any term in the model we want to ask its value, we can make a call to the inference engine (for example, is the sighting of the front goal consistent?). The C

implementation returns one of the three values of PL. For those landmarks found consistent, the information on them is forwarded to localization.

For the analysis of each frame, we have a module named `Consistency`. The C++ code of the template method `Consistency::Run` is executed every time a new frame arrives. In the filtering for localization example, the vision module is reporting about landmark sightings in each frame. The template method runs and verifies such reports.

```
void  Consistency::Run()
{INIT_ALL_FALSE(); UPDATE_ALL(); PLACE_CS_ALL();}
```

The three macros in the template method have the responsibility of implementing the three phases. The three macros in the template method are defined in a file named `ConsistencyMacros.h` that provides the glue code to the Mi-Pal architecture for the soccer playing robots as well as glue code for a visual testing tool. The code in `ConsistencyMacros.h` is computer generated, and depends slightly on the model to be executed on the Sony AIBO.

For the particular case of the Model 1 just presented, testing (evaluating) if the front goal is consistent is a call to the inference engine. If the front goal was not seen in this frame, then `INIT_ALL_FALSE()` would have set the variable to false and `UPDATE_ALL()` would not have changed FG's value, so no landmark sighting is forwarded to localization. However, if the front goal was visible, `UPDATE_ALL()` would have set FG to true and the `if` statement in `PLACE_CS_ALL()` would fire (because the engine would have used the logic rules of the model to prove a consistent sighting), resulting in localization receiving the sighting information about the front goal.

3.2.3 Higher Level Models

Originally, we developed the simple Model 1 for validation of the entire concept of a non-monotonic logic implemented on a Sony AIBO. We now introduce progressively more sophisticated models. We present a model that handles the consistency cases when vision reports 0, 1, or 2 landmarks in a frame. The type declarations regarding the landmarks are the same as before, but we need to describe the domain in a bit more detail. In particular, we use *Opp*(*x*,*y*) to mean *x* is opposite *y*. Also *Opp*(*x*,*y*) if and only if *Opp*(*y*,*x*). This appears in the programming language as

```
type Opp(x <- Landmark, y <- Landmark - {x}).  default ~Opp(x, y).
Opp(FG, BG).   Opp(RP,  LBP).   Opp(RBP, LP).
Opp(BG, FG).   Opp(LBP, RP).    Opp(LP,  RBP).
```

Also, we want to define relative positioning on the soccer field. The predicate *LR*(*x*,*y*) means landmark *x* is to the left of landmark *y*, and there are only 0 or 1 landmarks between them. The following are facts about left-to-right placements.

```
type LR(x <- Landmark, y <- Landmark - {x}).  default ~LR(x, y).
LR(LP, FG).  LR(RP, BG).   LR(FG, RBP).  LR(RBP, BG).
LR(LP, RP).  LR(BG, LP).   LR(RP, RBP).  LR(RBP, LBP).
LR(FG, RP).  LR(LBP, FG).  LR(BG, LBP).  LR(LBP, LP).
```

We are now in a position to use inputs from vision (using the same macro to declare them). Declarations for *See*(*x*) and axioms relating this predicate to the C++ expression `FG` are as before. What is new in this model is that we now use that vision reports if one landmark appears to the left of another. Plausible assumption *SeeLtoR*(*x*,*y*) means vision reports seeing landmark *x* to the left of landmark *y*.

```
type SeeLtoR(x <- Landmark, y <- Landmark - {x}).
```

Also for efficiency of the computation of proofs, we can use rules that simplify the setting. A macro for an axiom such as `LP_FG` that fires *SeeLtoR*(*LP*,*FG*) allows us to not consider cases where `LP_FG` is asserted but either of `LP` or `FG` is not. In the programming language we define a macro call `$+declareSeeLtoR$(`*x*`$,`*y*`$)` that declares an input axiom *x_y*, rules to fire *SeeLtoR*(*x*,*y*), and then specifies the (possible) cases to ignore.[7]

```
$=declareSeeLtoR$(x$,y$)${$#
 $+declareInput$("$+x_$+y"$,SeeLtoR($+x,$+y)$)
  ignore {"$+x_$+y",~"$+x"}.  ignore {"$+x_$+y",~"$+y"}. $}$#
```

Then, we only need to use this macro, for all possible pairs of landmarks, for example

```
$+declareSeeLtoR$(LBP$,LP$).
```

For the new model, the first two rules are the same as before. Namely, nothing is to be forwarded to another module unless it is seen. But now we add that if vision reports two landmarks we know to be opposites, then we believe neither (irrespective of whether vision reports one to the left of the other).

```
R3: {See(x), See(y), Opp(x,y)} => ~Cs(x).  R3 > R2.
```

If vision reports two objects out of left-to-right order, then we also believe neither.

```
R4: {See(x), See(y), SeeLtoR(y,x), LR(x,y)} => ~Cs(x).
R4: {See(x), See(y), SeeLtoR(y,x), LR(x,y)} => ~Cs(y).  R4 > R2.
```

The complete rules for **Model 2** are expressed in the programming language as follows:

```
R1: => ~Cs(x).  R2: See(x) => Cs(x).  R2 > R1.
R3: {See(x),See(y), Opp(x,y)} => ~Cs(x).  R3 > R2.
R4: {See(x),See(y),SeeLtoR(y,x),LR(x,y)} => ~Cs(x).
R4: {See(x),See(y),SeeLtoR(y,x),LR(x,y)} => ~Cs(y).  R4 > R2.
```

Note the non-monotonic aspect of the model. In particular, the inference engine may reach the initial conclusion that nothing is to be forwarded to the localization module (by R_1) but then conclude that there is a landmark sighting to be forwarded (because of R_2). However, it may change that conclusion in light of R_3.

[7]While the `ignore` statements do not influence the plausible rules directly, they serve the purpose of declaring that the given combinations of inputs can be ignored.

If we want to add rules that use other aspects of the information from the sensors in the report from vision, we can also achieve this. We illustrate with rules to report a post over a goal, or the larger of two goals. We can now revise **Model 2** to **Model 2a** to use information on objects size or type, for example. We may want to report a post over a goal even if perceived in the wrong left-to-right order.

```
R1: => ~Cs(x).  R2: See(x) => Cs(x).  R2 > R1.
R3: {See(x),See(y), Opp(x,y)} => ~Cs(x).  R3 > R2.
R4a: {See(x),See(y),SeeLtoR(y,x),LR(x,y)} => Cs1(x,y).  R4a > R2.
R5: {Cs1(x,y),Post(x),Goal(y)} => Cs(x).
R5': {Cs1(x,y),Post(x),Goal(y)} => ~Cs(y).
R6: {Cs1(x,y),Post(x),Post(y),BigSmall(x,y)} => Cs(x).
R6': {Cs1(x,y),Post(x),Post(y),BigSmall(x,y)} => ~Cs(y).
```

The predicate *Cs*1(*x*,*y*) means only one of *x* and *y* is consistent with the domain knowledge. We now have that when two objects on the scene are a post and a goal and we have concluded one is inconsistent, then we prefer the post (because it is harder to confuse an object identified with two colours). The last rule says that the larger of two inconsistent posts is to be forwarded as input for localization since it is harder to perceive a large phantom post.

There is a case with 3 landmarks in a frame where Model 2 could be refined. To describe this new refined model we have to state some more facts about the environment. We use *Adj*(*x*,*y*,*z*) to say *x* is left of and next to *y*, and, *y* is left of and next to *z*.

```
type Adj(x <- Landmark , y <- Landmark - {x},
z <- Landmark - {x, y}).  default ~Adj(x, y, z).
Adj(LP,FG,RP). Adj(FG,RP,RBP). Adj(RP,RBP,BG).
Adj(RBP,BG,LBP). Adj(BG,LBP,LP). Adj(LBP,LP,FG).
```

Let us consider the case when we see three objects *x*, *y*, and *z*, known to be adjacent from left to right, but we see *x* on the wrong side of both *y* and *z*. While we do not need to revise our opinion about *x* being inconsistent (by R_4) the mutual consistency of *y* and *z* are grounds for overriding the conclusion by R_4. This leads to an extension that we name **Model 3**.

```
R5: {See(x),See(y),See(z),SeeLtoR(y,z),SeeLtoR(z,x),Adj(x,y,z)} => Cs(y).
R5: {See(x),See(y),See(z),SeeLtoR(y,z),SeeLtoR(z,x),Adj(x,y,z)} => Cs(z).
R5 > R4.
```

Similarly, if *z* is the one out of order, then believe *x* and *y*.

```
R5: {See(x),See(y),See(z),SeeLtoR(z,x),SeeLtoR(x,y),Adj(x,y,z)} => Cs(x).
R5: {See(x),See(y),See(z),SeeLtoR(z,x),SeeLtoR(x,y),Adj(x,y,z)} => Cs(y).
```

The rules that complete this model are as follows.

```
R6: {See(x),See(y),See(z),SeeLtoR(x,z),SeeLtoR(z,y),LR(x,y),LR(y,z),
     Opp(x,z)} => Cs(x).
R6: {See(x),See(y),See(z),SeeLtoR(x,z),SeeLtoR(z,y),LR(x,y),LR(y,z),
     Opp(x,z)} => Cs(y).
R6 > R3.  R6 > R4.
```

We omit the last model that handles even 4 landmarks in the same frame. But we believe this progression of models and the illustration of their design suffices.

3.3 A First Evaluation

The implementation of these PL models has been evaluated in two directions. The effectiveness of the approach was demonstrated in an ERS-7 Sony AIBO in the lab and in the actual RoboCup competitions. We evaluated the results on the robot with a `telnet` connection that displays the ID of the objects reported by vision (see Figures 1 and 2).

Figure 1 creates a scene where a non-moving Sony AIBO would have a vision module where the left post is correct, but the goal and right post are inverted. We can see in the figure the telnet connection that portrays the image captured in the robot as well as the text that reports which landmark sightings are being forwarded for localization. For Figure 1, Model 2 provides the correct result. Namely, we can forward to localization the left post, but neither the goal nor the right post should be forwarded to the localization module.

We believe it is remarkable that the PL description for Model 2 (which only aims at writing rules for frames with zero, one or two landmark sightings) obtains correct conclusions for many settings in which three sightings or more occur. This again reflects the power of modeling with PL as opposed to modeling without it. The PL is analyzing the pairs within the triplet in sight. And while two of the pairs are consistent (those involving the left post), the pair involving the goal and the right post indicates both of these are inconsistent.

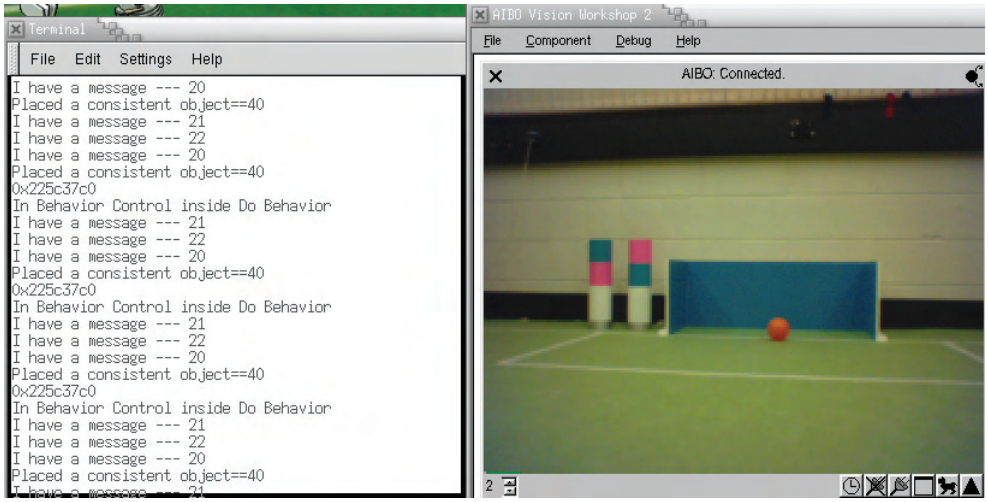


Fig. 1. The left post is correct, but the goal and right post are inverted

Not all settings with 3 landmarks are correctly identified with Model 2. Figure 2 is a setting where Model 2 rules all landmark sightings as inconsistent. In this case, all objects are involved with another object to form a pair seen in the incorrect left to right order with respect to the domain knowledge. While Model 2 rules this setting as inconsistent Model 3 correctly identifies that the left post and the goal constitute a pair seen in the expected order and thus it is likely that the right post is the phantom.

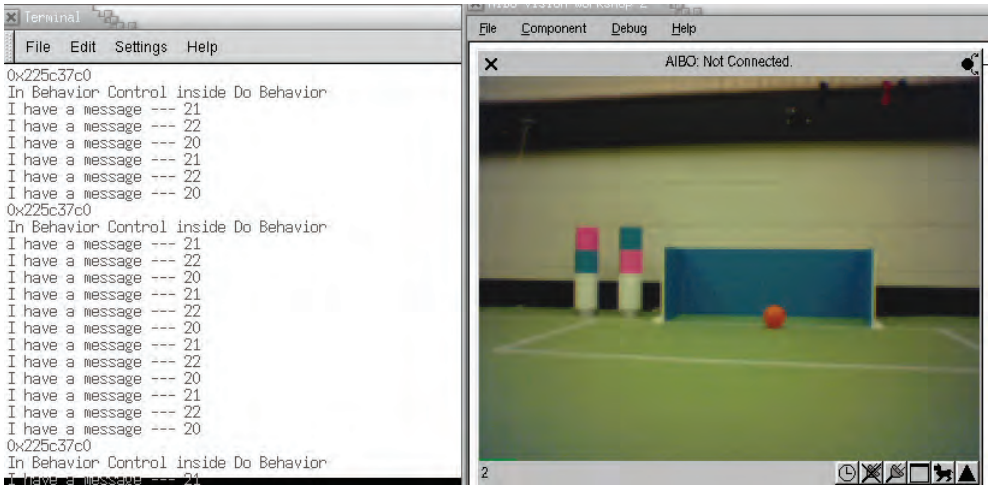
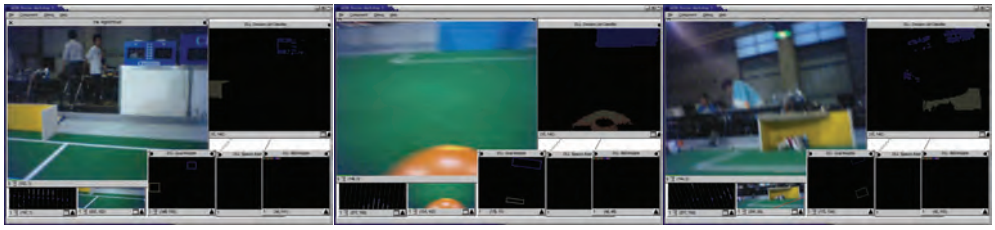


Fig. 2. The left post and the goal are in correct order, but the right post is not

The following figures show images at the RoboCup-2005 venue in Osaka where the consistency module filtered phantom objects for localization. Figure . 3 shows the processing by vision system on board the Sony AIBO. We have enlarged the captured image on board, then the blobs of color as the second largest and the objects reported by vision appear on three screens on the bottom right corner. The left most of these bottom images displays the sightings for goals.



(a) The blue timer appears as a goal with the yellow goal. (b) Phantom goal caused by yellow pixels on the ball. (c) A window appears as a blue goal above the real goal.

Fig. 3. Examples of competition situations where we see opposite goals in the same frame

Figure . 3 (a) shows that the blue match score and timer appear as a goal on a frame with the yellow goal. While our vision system has an analysis for filtering objects above the field of vision, the fact that the Sony AIBO has a head with three degrees of freedom and has legs that during pursuit of the ball make positions and angles of vision that cannot always rule this case out. Figure . 3 (b) shows that phantom sightings occur even with the regular color-coded objects in the field. The ball has enough yellow pixels to be confused with a yellow goal against a blue goal. Figure . 3 (c) shows another case where natural lighting and off the field objects result in phantom sightings. In this case, a window registered enough blue pixels to be reported as a blue goal on a frame that spots the yellow goal as well.

With the aid of a GUI simulator (description to follow) and the telnet connection, the models

were evaluated for all possible configurations of phantom and real sightings that involve up to three landmarks, and even in some cases 4 or 5 landmarks. Some examples of the outcomes are shown in the Figure 4. We invite the reader to attempt to decide what the reliable sightings are before exploring the results produced by the models.

The results are as follows. In Figure 4 (a) only BG and RBP are consistent. For Figure 4 (b), BG and RBP are the consistent objects while only RBP is consistent for Figure 4 (c). For these three previous cases, with Model 3 both `Cs_LBP` and `¬Cs_LBP` are -1. For Figure 4 (b) both `Cs_RP` and `¬Cs_RP` are -1. In Figure 4 (d) nothing is consistent while for Figure 4 (e) only RBP and RP are consistent.

(a) RBP BG LBP RP (b) RP RBP BG LBP (c) RP RBP LBP (d) RBP BG FG LP (e) RBP BG FG RP

Fig. 4. Interesting cases for the static models

3.4 Using A Model of Time to Improve Localization

Our previous model illustrated reasoning based only on the current reading from the sensors. It is natural that decisions on agents may be based not only on the information on the current state of the system, but also on data retrieved in the past. To illustrate how we can accomplish reasoning about the current and previous states of the environment we show a temporal expansion of the previous spatial DPL model. In a temporal model, the objects may have been visible in the previous frame or in the current frame. So we model this by considering that vision now reports sightings with respect to a time step or a frame (i.e. the predicate is now *See*(*x*,*f*)).

```
type Frame = {PF, CF}. type See(x <- Landmark, f <- Frame).
type SeeLtoR(x <- Landmark, y <- Landmark - {x}, f <- Frame).
```

Sightings may be transient (did not last across consecutive frames) or persistent (the object is in both frames).

```
type Tra(x <- Landmark).   R1: {} => ~See(x,f).   R2: {} => ~Tra(x).
R3: {See(x, PF), ~See(x, CF)} => Tra(x).
R3: {~See(x, PF), See(x, CF)} => Tra(x).  R3 > R2.
type Per(x <- Landmark).   R4: {} => ~Per(x).
R5: {See(x, PF), See(x, CF)} => Per(x).   R5 > R4.
```

Nothing is consistent unless we get at least a transient or a persistent sighting.

```
type Cs(x <- Landmark). R6: {} => ~Cs(x).
R7: {Tra(x)} => Cs(x). R7 > R6. R8: {Per(x)} => Cs(x). R8 > R6.
```

Seeing two opposite landmarks is grounds for inconsistency (even persistently or transiently).

```
R9: {Opp(x, y), Per(x), Per(y)} => ~Cs(x). R9 > R7. R9 > R8.
R10: {Opp(x, y), Tra(x), Per(y)} => ~Cs(x).
R11: {Opp(x, y), Tra(x), Tra(y)} => ~Cs(x). R10,R11 > R7.
```

What does it mean for two objects to be in a transient left-to-right order? This happens if vision sees the objects in the previous frame in that order but does not see them in the current frame in that order, or they are seen in the current frame in that order but they were not seen in the previous frame in that order.

```
type TraLtoR(x <- Landmark, y <- Landmark - {x}).
R14: {} => ~TraLtoR(x,y).
R15: {SeeLtoR(x,PF,y,PF),~SeeLtoR(x,CF,y,CF)} => TraLtoR(x, y).
R15: {~SeeLtoR(x,PF,y,PF),SeeLtoR(x,CF,y,CF)} => TraLtoR(x, y).
R15 > R14.
```

However, objects are persistently seen in a left-to-right order if the sighting of that relationship happened in the previous and current frame.

```
type PerLtoR(x <- Landmark, y <- Landmark - {x}).
R16: {} => ~PerLtoR(x,y).
R17: {SeeLtoR(x,PF,y,PF), SeeLtoR(x,CF,y,CF)} => PerLtoR(x,y).
R17 > R16.
```

Finally, seeing landmarks out of order is grounds for inconsistency. But we only overwrite those rules that may have suggested consistency.

```
R18: {LR(x, y), Per(x), Per(y), PerLtoR(y, x)} => ~Cs(x).
R18: {LR(x, y), Per(x), Per(y), PerLtoR(y, x)} => ~Cs(y).  R18 > R8.
R19: {LR(x, y), Tra(x), Per(y), TraLtoR(y, x)} => ~Cs(x).
R19: {LR(x, y), Per(x), Tra(y), TraLtoR(y, x)} => ~Cs(y).  R19 > R7.
R20: {LR(x, y), Tra(x), Tra(y), TraLtoR(y, x)} => ~Cs(x).
R20: {LR(x, y), Tra(x), Tra(y), TraLtoR(y, x)} => ~Cs(y).  R20 > R7.
```

3.5 Implementation of Temporal Models

For the temporal model, the robot executes a variant of the template method (named `Run()`). Again, the template method is executed every time a prompt from the environment demands an action and we want to do some reasoning before the action. For the localization example, arrival of a frame and its analysis by the vision module are prompts for reasoning about the sightings before sending results to the localization module. However, this variant involves other macros.

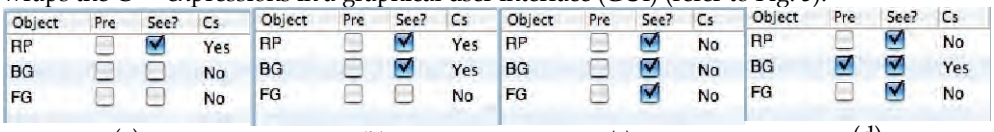
```
void  Consistency::Run()
{INIT_ALL_FALSE(); UPDATE_ALL(); CHECK_NEW_LANDMARKS();
PLACE_CS_ALL(); COPY_ALL_BOOL(); }
```

As explained earlier, the building blocks of `Run()` are procedures defined by computer generated glue code macros. The `INIT_ALL_FALSE()` macro implements the first phase. It gives definitions of C++ Boolean variables for all defined inputs and sets all Booleans

corresponding to inputs of the current frame to false. `UPDATE_ALL()` queries the reports of all the sensors in the current status of the environment, in our case obtains from the vision module reports for the current frame. We have a new intermediate phase `CHECK_NEW_LANDMARKS()` that ensures all information from sensors is labeled with its time step identifier. A sensor reading now is labeled with subindex 0, but for the previous fame the index is -1. Indexes are shifted when this macro is executed. In our case, this ensures that sightings are for the current frame and that previous sightings also have the correct fame numbers relative to the current frame. As before, `PLACE_CS_ALL()` will evaluate the output expressions using the inference engine. If a landmark is found consistent, it will forward the sighting to the localization module (or any other module that may benefit from it, like the action to kick when the front goal is visible). `PLACE_CS_ALL()` will have as many `if` statements as outputs/proofs are requested. For example, testing (evaluating) the `Cs_FG` macro, corresponds to asking the inference engine if we have a consistent sighting of the front goal. Finally `COPY_ALL_BOOL()` shifts all the current Boolean values in C++ Boolean variables to the Boolean variables corresponding to previous frames (so the previous frame values are correctly set for the next execution of the template method `Run()`). We avoid keeping the reports and evaluating predicates regarding vision reports on previous frames.

3.6 Evaluation of the Temporal Model

The entire architecture, and the models, were also validated in a Graphical User Interface (GUI) simulator. That is, the `roboconsistency.h` files produced for a model as output by DPL can be used by the GUI simulator as well. A user interacting with the GUI simulator can set up the vision reports (for example, set up a scenario where the front goal and front post are visible and the front post is to the right of the front goal). The GUI simulator then indicates which of these landmark sightings are regarded as worth forwarding to the localization module. This simulator facilitates the debugging of the entire architecture without having to execute the consistency module on the robot. This is particularly useful in the evaluation of temporal properties. On the robot, validation is particularly hard because sighting errors only occur sporadically at a frame rate higher than 25 Hz. Our simulator allows reproducible scenarios of what vision might report to the localization system and this was used to validate the correctness of the PL expressions resulting from a model. The inference engine is also the same in the simulator and the on-board execution module on the robot. In order to provide a way to consistently set and evaluate a scene, the simulator wraps the C++ expressions in a graphical user interface (GUI) (refer to Fig. 5).



(a) (b) (c) (d)

Fig. 5. (a) The simulator showing one particular object (RP) as visible in the current frame. (b) Two consistent objects in the current frame. (c) Three inconsistent sightings in the current frame. (d) The persistent back goal (BG) wins over the temporary sightings of the right post (RP) and the front goal (FG)

The user of the simulator places landmarks on rows, and the succession of rows represents

the order in which these objects are seen (with top to bottom representing left to right in the field of vision). The first column shows the landmark. The next two columns allow the user to select what the visibility state is for the previous (`Pre`) and current (`See?`) frames. The rightmost column shows the output of the consistency module (`Cs`) after performing its reasoning. Furthermore, the GUI allows the dragging and dropping of objects to change the order, as well as addition (`Add`) and deletion (`Delete`) of landmarks.

Fig. 5 (b) shows two consistent sightings. Even though the two landmarks RP and BG were not visible in the previous frame, they are consistent with each other, allowing them to be forwarded on to the localization module. In fact, this is the same result that a traditional value domain reasoning system would obtain. This is also true for Fig. 5 (c) where we can see three objects that are inconsistent with each other. Since all the objects only occur within one single frame, the only conclusion that can be drawn is that nothing is consistent in that scene. Once information varies over time, a richer belief about the environment can be formed. Fig. 5 (d) shows the same scenario as Fig. 5 (c), but this time the temporal properties of the visible objects vary. The back goal that was visible in the previous frame as well as the current frame is given precedence over the right post and the front goal that were only visible in a single frame.

Temporal and spatial tests can be combined as in Fig. 6 (a). Objects that are consistent in either space or both space and time are ruled as being consistent in the world view of the system. Only inconsistencies that persist over both space and time will force the system to conclude that nothing is consistent (Fig. 6 (b)).

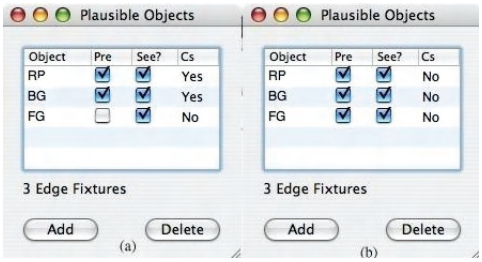


Fig. 6. Whether an `Object` is visible is indicated by `See?` for the current frame and by `Pre` for the previous frame. Column `Cs` shows whether the corresponding object is concluded to be consistent. (a) The two persistent landmarks (RP and BG) are consistent with each other, but inconsistent with the front goal (FG). (b) Nothing is consistent in this view

3.6.1 Evaluation on the Robot

We also have analyzed the effectiveness of the approach in a Sony AIBO ERS-7 in the lab. Fig. 7 shows a lab setting where we can rapidly produce opposite goals in a frame and immediately after block one goal, or the other. In the log, we found sequences where the robot is seeing only the front goal and reports it as consistent. When the back goal appears as well, for that first frame, the front goal remains consistent and the back goal is labeled inconsistent (note that in the discussion of the KF, MM, and MCL localizers we indicated that the frame with both goals becomes not usable). If the back goal persists with the front goal for one more frame, then both goals are now labeled inconsistent (note that the model

can easily be adjusted if a different effect is desired besides having two consecutive frames with both goals to rule them out). If the front goal drops out, then the back goal in the previous frame and the back goal in the current frame both become consistent. We have no space here to discuss more examples of the versatility of the modeling here, but using the RP also helps since the blue goal is in the right left-to-right order with respect to the post.

(a) Both goals are visible. (b) The blue goal is covered. (c) The yellow goal is covered.

Fig. 7. Lab Examples. A setting that allows sequences of frames where two goals are reported by vision. Goals can be covered and uncovered quickly

4. Plausible Logic for the Referee

One of the most crucial aspects of soccer is the offside rule (rugby, hockey and many others have similar variants). Typically, the rule is a source of much debate, and almost every soccer fan forgets one or more of the exceptions when first presenting the rule to a novice. The rule represents an interesting resource for the defense, and its comprehension is crucial for the attacking team. Naturally, soccer referees (main and assistant) are mostly judged by their ability to officiate the rule.

If robotic players are to participate in a competition in 2050 and defeat the human world champion they would need to reason about these types of rules. Moreover, we can foresee that an artificial agent can enforce the rules in leagues like the simulation league or even the Sony AIBO competition as there are now video analyzers that recognize players, landmarks and the ball (Ruiz-delSolar *et al.*, 2006).

The official rules of the game, as per the FIFA web site indicate that Rule 11 corresponds to the off-side rule.

> *It is not an offence to be in an offside position. A player is in an offside position if he is*
>
> - *nearer to his opponents' goal line than both the ball and second last opponent*
>
> *A player is not in an offside position if he is*
>
> - *on his own half of the field of play or*
> - *level with the second last opponent or*
> - *level with the last two opponents*
>
> *A player in an offside position is only penalized if, at the moment the ball touches or is played by one of his team, he is, in the opinion of the referee*
>
> - *interfering with play or*
> - *interfering with an opponent or*

- *gaining an advantage by being in that position.*

There is no offside offence if a player receives the ball directly from

- *a goal kick or*
- *a throw-in or*
- *a corner kick.*

We now describe this with a model using PL with the programming language DPL. We will mimic closely the official wording, although simpler equivalent models can be described more succinctly. We define the objects we need to talk about as the last two opponents and the ball.

```
type Last2Opponents = {Opp1, Opp2}.
type Objects = {Ball} + Last2Opponents.
```

We also need to describe the possible types of activities for a player and the possible ball transfers.

```
type Plays = {IfPlay, IfOpp, TkAd}.
type Transfers = {GlKk, ThwIn, CnrKk}.
```

The robotic referee would need to deduce from its sensors whether a player is level with the second last opponent.

```
type lvl(x <- Last2Opponents).
```

Also, using its sensors, it must identify the involvement of the player in play and the types of transfers that result in no offside.

```
type play(x <- Plays).
type xfer(x <- Transfers).
```

The default situation is that a player is not committing an offense, as per the official rules above. Note that we are naming rules with more meaningful names that just a rule identifier.

```
NoOffence: {} => ~offsideOffence.
```

The remaining rules are now clear from the similarity with the FIFA's Rule 11.

```
Offence: {offsidePosition, active} => offsideOffence.
Offence > NoOffence.

NotOffside: {} => ~offsidePosition.

Offside: {nearr(x) | x <- Objects} => offsidePosition.
Offside > NotOffside.

OwnHalf: {ownHalf} => ~offsidePosition.  OwnHalf > Offside.

lvl: {lvl(x)} => ~offsidePosition.  lvl > Offside.
```

```
NotActive: {} => ~active.

Active: {play(x)} => active.  Active > NotActive.

Transfer: {xfer(x)} => ~offsideOffence.  Transfer > Offence.
```

5. Plausible Logic for the RoboCup@Home

RoboCup@Home is a league that concentrates on real-world applications for robotics. It has a strong focus on interaction between autonomous robots and humans, aiming at the development of applications that can assist humans in everyday life (van der Zant & Wisspeintner). We believe that such applications will become prevalent in the near future and will be an integral part of our lives in areas such as public transport, housework, care, and medicine. We have explored human-robot interfaces for assisting learning of blind children (Bartlett *et al.*, 2003) and e-mail between blind adults (Estivill-Castro & S., 2006). As originally proposed (van der Zant & Wisspeintner), RoboCup@Home anticipates many applications of robots assisting elderly humans with situations like emergencies.

We have presented, as part of the Open Challenge in RoboCup 2007, the use of rules in DPL for a scenario where an elderly lady (Grandma) lives alone at home. While she does not require constant care, raising an alarm in an emergency or if something unexpected happens can be of vital importance. We now illustrate the constructing of these rules for our scenario. We use vision as the main source of input and information about the environment since RoboCup@Home settings can hardly have specific sensors for robots. Again, the suggested methodology for building a model is to introduce the rules incrementally, adding consideration of possibly visible objects one at a time.

We presume that the raising of an alarm could be forwarded to a behavior module on the robot. Thus, *alarm* means an alarm should be raised about Grandma's welfare. By default, no alarm should be raised. *Default*: {}⇒~*alarm* The frames analyzed by vision have attached a frame identifier *f*. $F^{t',t}$ denotes the sequence of frames between two points in time, t and t'. The variable *now* denotes the point in time when a decision for an alarm is being made. For a short time in the past, say half an hour, we use *short*, but *long* is a long time, say 8 hours. The variables *long* and *short* could represent a collection of tunable parameters, for those rules that use them. Again, sightings from the sensors are plausible assumptions and *See*(*x*,*f*) is true if the vision module reports that object *x* is visible in frame *f*. Sorry Grandma, but to your robotic helper you are just an object.

A prolonged absence of *Grandma* is reason to raise an alarm. PL represents this as a definition for what *absence* means (*absence* = $\forall f \in F_{now}^{long}$ ~*See*(*Grandma*,*f*)), one plausible rule *Absence*: {*absence*}⇒*alarm*, and an instance of the priority relation *Absence* > *Default*. In DPL this can be expressed as follows.

```
Default: {} => ~alarm.
Absence: {absence} => alarm.  Absence > Default.
```

Grandma is likely to be in trouble (suffered a fall) if not standing, that is *Horizontal*. If Grandma is horizontal, then Grandma is by necessity visible. Our implementation can

assume that the analysis of a frame by vision can not assert *Horizontal*(*Grandma,f*) without also asserting *See*(*Grandma,f*) over matching frame ranges. A fall is likely if Grandma is *Horizontal* for a short time. Thus, formally, we need again a definition (*lying* = $\forall f \in F_{now}^{short}$ *Horizontal*(*Grandma,f*)), two refining rules (*Lying*: {*lying*} ⇒ *fall*, and *Fall*: {*fall*} ⇒ *alarm*), and one more instance in the priority relation *Fall*>*Default*. This can be expressed by the following DPL rules.

```
Lying: {lying} => fall.

Fall: {fall} => alarm.  Fall > Default.
```

With the simple model so far, we can already handle two cases that might be cause for concern and raise an alarm. However, in a real-world scenario, it may be perfectly valid for Grandma to lie down and rest. It is likely that when Grandma is lying down on her bed, that she may be resting. We now model this refinement. We assume that the robot can sense *Over*(*x,y,f*), that means that object *x* is over object *y* in frame *f*. *Over*(*x,y,f*) implies *See*(*x,f*) and *See*(*y,f*). There is no possibility that Grandma can be absent and over her bed, so rules *Absence* and *OnBed* can never conflict. Mathematically, *onBed* = $\exists f \in F_{now}^{short}$ *Over*(*Grandma,Bed,f*). The refinement is introduced with a rule that indicates the exception *OnBed*: {*onBed*} ⇒ ~*alarm*. And the priority over the rule that raises an alarm when lying *OnBed* > *Lying*. The corresponding DPL rules are as follows.

```
OnBed: {onBed} => ~fall.  OnBed > Lying.
```

However, if Grandma stays in bed for too long and is not getting up, this may still be cause for concern. Therefore, if Grandma is horizontal for a long time, raise an alarm regardless of where she is. We specify what it is to be lying for too long as *LyingLong*: {*lyingLong*} ⇒ *notGettingUp*, where *lyingLong* = $\forall f \in F_{now}^{long}$ *Horizontal*(*Grandma,f*). Then, we have an exception to the exception with the rule *NotGettingUp*: {*notGettingUp*} ⇒ *alarm* and another instance of the priority relation *NotGettingUp* > *Default*. So we can extend our DPL model once more to take the amount of time that Grandma has been lying down for into account.

```
LyingLong: {lyingLong} => notGettingUp.
NotGettingUp: notGettingUp => alarm.  NotGettingUp > Default.
```

Grandma is likely to be okay if the long time she is lying is at night and on the bed. At night you cannot see the sun. Grandma is not likely to get up at dawn, and may go to bed before dark. All we know is that bedtime overlaps nighttime. This leads to the following definition *nighttime* = $\exists f \in F_{now}^{long}$ ~*See*(*Sun,f*). We model the new revisions by another rule *Nighttime*: {*nighttime*} ⇒ ~*notGettingUp*. This is a revision on when it is OK to be lying too long *Nighttime* > *LyingLong*. The following DPL code handles this scenario.

```
Nighttime: {nighttime} => ~notGettingUp.  Nighttime > LyingLong.
```

To continue with this illustration, we ask the question what happens if Grandma is not home alone. Perhaps we want to raise an alarm if a stranger is looming over Grandma. For purposes of this scenario, a stranger is anyone other than Grandma. The presence of a stranger is not alarming unless Grandma is horizontal. We define the condition our sensors

can detect as *looming* = $\exists f \in F_{now}^{short}$ *See*(*Stranger*,*f*) $\wedge$ $\forall f \in F_{now}^{short}$ *Horizontal*(*Grandma*,*f*). Then, we define the rule *Looming*: {*looming*}$\Rightarrow$*alarm*. And place it in the hierarchy with *Looming*>*Default*. This can be expressed in DPL as follows.

```
Looming: {looming} => alarm.  Looming > Default.
```

To better illustrate how these rules work, Figure 8 summarizes the relationship between the rules shown above. To cope with more complex scenarios, additional rules can be created and priority relations can be added to resolve potential conflicts between the existing set of rules and the newly added set.

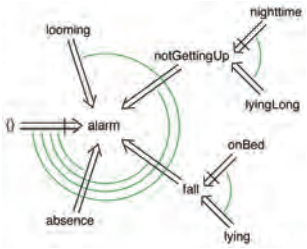


Fig. 8. Grandma's rules represented graphically. A priority is represented by an arc. The anticlockwise end beats the clockwise end

5.1 Implementation and Evaluation

The DPL model for Grandma's helper was implemented and evaluated in a similar fashion to the robotic soccer models introduced earlier. Using the GUI simulator, all combinations of inputs were tested to verify that the response (*alarm* or no *alarm*) was consistent with the model's discussion. An instance of the simulator using the complete set of rules is shown in Figure 9.

The user can set the conditions as perceived by a virtual vision module. If the user changes the inputs, the engine attempts to prove the *alarm* output. The "Proof" output column for *alarm* is set to +1 if an alarm should be raised and to -1 if no alarm should be raised. An output of 0 would indicate that the current situation cannot be decided either way (because a proof always leads to loop). This cannot (and did not) occur with the given set of rules. The "Negation" output column shows the results of proofs of ~*alarm*.

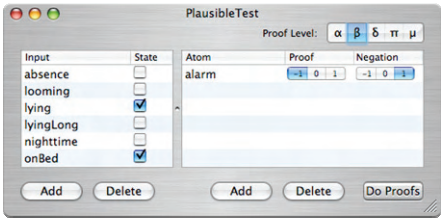


Fig. 9. Evaluation of the Grandma-Alarm model in the Plausible Test GUI simulator. The state of the DPL inputs is set on the left side of the screen while the output (the proof result from the engine whether *alarm* should be set or not) is shown on the right hand side

The same model and the DPL proof engine were then used on an ERS-7 Sony AIBO for the RoboCup@Home open challenge. We used the same vision module of our robotic soccer code. Any person dressed in blue is easily recognized using the code that recognizes the blue goal, while we set the bed yellow so we reused the modules for identifying the yellow goal. An orange circle passes for the sun, and thus the code for the module to recognize balls was reused. Any other object, when visible, was perceived as a stranger.

While most of the robotic software used for the league (such as vision, the behavior module, and the networking code) could be re-used some changes needed to be made to accommodate the Grandma scenario. A new module is introduced with the template method and the phases we discussed before. The template method executes every time a frame is ready and analyzed by vision. This new module forwards alarm signals to other modules (a behavior module or a network connection if an *alarm* takes the value true for the current frame). Naturally, some of the definitions must be translated into concrete detectable sensor information. Thus, some C++ code needs to be produced. For example, in a soccer game, the goal would never be expected to be in a vertical position. Since it does make a difference whether Grandma is standing upright or is lying in a horizontal position, a method was added that would determine whether the dimensions of the blue object (if reported as visible by the vision module) were horizontal (wider than its high) or vertical (higher than its width). Similarly, if both goals (i.e., both Grandma and her bed) were visible, C++ code needs to be added to compare their relative positions to provide the information whether Grandma is lying on the bed or not. C++ code needs to be added for the other terms defined in PL that demand information from processing sensory input. To determine if Grandma had been lying down for long, we developed new C++ code that counts the number of frames that Grandma had been seen in a horizontal position.

6. Conclusion

One of the most satisfying aspects of our implementation is that it has proven efficient enough to be running on board a Sony AIBO while in competition in the soccer 4-legged league or in RoboCup@Home. Table 1 shows the CPU-timings on board an ERS-7 running our C++ implementation with three different models and two situations. It may be surprising that while the robot was in the playing state, chasing the ball and executing kicks, the execution is faster than while standing as a goalie. However, the standing situations have usually on average 2 landmarks per frame. But while playing, frames with 2 objects in sight are less frequent. In all cases, the inference engine is executed six times per frame, to verify if each of the landmarks is consistent.

Model	Activity	Phase 1 and Phase 2	95 % Confidence Interval	Phase 1, Phase 2, and Phase 3	95 % Confidence Interval	Net Phase 3
4	Chasing	749 µs	± 7 µs	931 µs	± 8 µs	182 µs
4	Standing	1,438 µs	± 31 µs	1,687 µs	± 35 µs	249 µs
3	Standing	407 µs	± 15 µs	622 µs	± 17 µs	215 µs
2	Standing	209 µs	± 13 µs	371 µs	± 17 µs	162 µs

Table 1. CPU-times for the 3 phases of our template method on a Sony AIBO ERS-7

7. References

Antoniou, G. (1997). *Nonmonotonic Reasoning*. MIT Press, Cambridge, Mass. ISBN 0-262-01157-3.

Bartlett, B.; Estivill-Castro, V.; S., Seymon & Tourky, A. (2003). Robots for pre-orientation and interaction of toddlers and preschoolers who are blind. In Roberts, J. & Wyeth, G., editors, *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, Brisbane, Australian Robotics and Automation Association Inc, Queensland Centre for Advanced Technologies (QCAT). CD-ROM (paper 13.pdf) ISBN 0-9587583-5-2.

Billington, D. & Rock, A. (2001). Propositional plausible logic: Introduction and implementation. *Studia Logica*, 67:243–269. ISSN 1572-8730.

Billington, D.; Estivill-Castro, V.; Hexel, R. & Rock, A. (2005). Non-monotonic reasoning for localisation in robocup. In Sammut, C., editor, *Australasian Conference on Robotics and Automation*, Sydney, Australian Robotics and Automation Association. ISBN 0-9587583-7-9.

Billington, D.; Estivill-Castro, V.; Hexel, R. & Rock, A. (2006). Using temporal consistency to improve robot localisation. In Lakemeyer, G. & Sklar, E., editors, *International RoboCup Symposium*, Bremen, Germany, Springer-Verlag Lecture Notes in Computer Science. Volume 4434, pages 232-244, 2007. ISBN 978-3-540-74023-0.

Billington, D. (2005). The proof algorithms of plausible logic form a hierarchy. In Zhang, S. & Jarvis, R., editors, *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, volume 3809, pages 796–799, Sydney, Australia, Springer Verlag Lecture Notes in Artificial Intelligence. ISBN 3-540-30462-2.

Brooks, R.A. (1991). Intelligence without reason. In Myopoulos, R. & Reiter, R., editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595, San Mateo, CA, ICJAI-91, Morgan Kaufmann Publishers. Sydney, Australia. ISBN 1-55860-160-0.

Compton, P.J. & Jansen, R. (1990). A philosophical basis for knowledge acquisition. *Knowledge Acquisition*, 2(3):241–257. ISSN 0001-2998.

Estivill-Castro, V. & S., Seymon. (2006). Mobile robots for an e-mail interface for people who are blind. In Lakemeyer, G. & Sklar, E., editors, *International RoboCup Symposium*, Bremen, Germany, Springer-Verlag Lecture Notes in Computer Science. Volume 4434, pages 338-346, 2007. ISBN 978-3-540-74023-0".

Fox, D.; Burgard, W. & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligent Research*, 11:391–427. ISSN 1076-9757.

Gutmann, J.-S. & Fox, D. (2002). An experimental comparison of localization methods continued. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, v 1, pages 454–459, Lausanne, Switzerland, IEEE. ISBN 0-7803-7398-7.

Haemmi, R. & Hartmann, S. (2006). Modeling partially reliable information sources: A general approach based on Dempster-Shafer theory. *Information Fusion*, 7:361–379. ISSN 1566-2535.

Kahney, L. (2003). The new pet craze: Robovacs. Wired Magazine. June, 16th; visited Septenber 10th, 2003, www.wired.com/news/technology/0,1282,59249,00.html.

Marek, V.W. & Truszczynski, M. (1993). *Nonmonotonic Logic: Context-Dependent Reasoning*.

Springer Verlag, Berlin. ISBN 0387564489.

Rich, E. & Knight, K. (1990). *Artificial Intelligence*. McGraw-Hill Higher Education, NY, second edition. ISBN 0070522634.

Rock, Andrew. The DPL (decisive Plausible Logic) tool. Technical report, (continually) in preparation, available at www.cit.gu.edu.au/ ,˜arock/.

Rock, A. & Billington, D. (2000). An implementation of propositional plausible logic. In Edwards, J., editor, *23rd Australasian Computer Science Conference*, volume 22(1) of *Australian Computer Science Communications*, pages 204–210, Canberra, IEEE Computer Society, Los Alamitos. ISBN 076950518X.

Ruiz-delSolar, J.; Loncomilla, P. & Vallejos, P. (2006). An automated refereeing and analysis tool for the four-legged league. In Lakemeyer, G. & Sklar, E., editors, *International RoboCup Symposium*, Bremen, Germany, Springer-Verlag Lecture Notes in Computer Science. in press.

Russell, S. & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Englewood Cliffs, NJ, second edition. ISBN 0130803022.

Thrun, S.; Bennewitz, M.; Burgard, W.; Cremers, A.B.; Dellaert, F.; Fox, D.; Hahnel, D.; Rosenberg, C.R.; Roy, N.; Schulte, J. & Schulz, D. (1999). MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz, 23rd Annual German Conference on Artificial Intelligence*, volume 1701, pages 14–26. Springer Verlag Lecture Notes in Computer Science. ISBN 3-540-66495-5.

Thrun, S.; Fox, D.; Burgard, W. & Dellaert, F. (2001). Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128:99–141. ISSN 0004-3702.

van der Zant, T. & Wisspeintner, T. Robocup X; a proposal for a new league where robocup goes real world. www.robocupathome.org.

Veloso, M.; Uther, W.; Fujita, M.; Asada, M. & Kitano, H. (1998). Playing soccer with legged robots. In *Proceedings of IEEE/RSJ Intelligent Robots and Systems Conference (IROS-98)*, volume 1, pages 437–442, Victoria, Canada. ISBN 0-7803-4465-0.

Wallich, P. (2001). Mindstorms - not just a kids toy. *IEEE Spectrum*, pages 53–57. ISSN 0018-9235.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons, NY, USA. ISBN 047149691X.

4

Color Classification and Object Recognition for Robot Soccer Under Variable Illumination

Nathan Lovell and Vladimir Estivill-Castro
Griffith University
Australia

1. Introduction

One of the biggest challenges for vision systems in mobile, autonomous robotics is to show adaptivity to changing visual circumstances. Many state of the art robotic platforms still use color segmentation as the basis for vision, even though this technique performs badly when the robot is removed from the exact lighting environment for which it was calibrated. Nevertheless, the very nature of a mobile platform with cameras on board, suggests that it will move and thus lighting conditions will change.

All variants of human soccer regularly use color-coding, but robotic soccer is currently limited to soccer fields with carefully controlled illumination conditions. Experiments in the RoboCup competition have shown how detrimental even the smallest lighting changes can be to platforms with cameras on board. Competitors calibrate their systems multiple times, even on the same field, simply because the ambient light in the room varies with the time of day. Small variations adversely affect the ability of agents to accurately assess the visual scene. Robotic soccer requires accurate visual information and our current systems simply do not provide this under variable or unknown illumination conditions.

Many attempts to resolve this issue involve a priori geometric information regarding the objects in the visual scene. For example, if I know that the only round thing that I expect to see is the ball, then I can identify the ball by its round shape —I do not need its color. Such techniques are useful but often require far more intensive computation. Robotic soccer is a real time adversarial, nondeterministic and inaccessible setting, on a platform with limited computational power. Therefore, some geometric image processing solutions are simply not suitable.

In this chapter, we describe our solution to this problem. Our solution is fast enough to run on the limited resources of a mobile system under the constraints of real-time image processing. Our technique is still basically a color segmentation process, so it runs in equivalent time to systems that are already used for robotic soccer. Faster processors or more reliable hardware (cameras/lenses) will result in more reliable and more robust systems under even larger illumination variations.

We do not attempt to exhaustively classify an entire color class for a single illumination condition. Rather we calibrate a core color class of, say, orange, that will remain orange under many illumination conditions. This sparse classification means that in no image will

every pixel that looks orange be classified as orange, however in every image some pixel that looks orange will be classified as orange. We can quickly find some pixels of any given class, independent of the illumination condition, and complemented with other techniques, we are able to identify objects.
Simple geometric shapes are usually easy to locate. A circle, for example, requires three edge points. We can thus use a special edge-detection algorithm to quickly find three points on the edge of the circle; starting from any of the pixels we have identified. A more complex shape may require a description of the entire edge. For this purpose we have developed a fast edge detection algorithm that can be used in conjunction with a border following algorithm so that we do not have to process all of the edges in the entire image.
A large proportion of the machine vision literature of recent times, both in RoboCup and elsewhere (Gunnarsson et al., 2005; Shimizu et al., 2005; Kak & DeSouza, 2002), has attempted to address the problem of dynamic illumination conditions. We present a viable solution here, provided that the conditions are not too widely variable. Our technique has been applied with success to the Four-Legged league in RoboCup. We have illustrated our ability to correctly identify objects on the field in real time in complex and dynamic lighting environments. The object recognition system is efficient for the RoboCup environment and is robust to changes in color intensity and temperature. It is based on a sparse classifier that is very accurate, but only on pixels that are at the core of each color class. It essentially refuses to make a decision for most other pixels, labeling them as unknown. The necessary ideas of image segmentation appear in Section 3 and the use of the sparse classifier appears in Subsection 5.1. Optimized edge-detection is the second pillar of our approach. Thus, Section 4 shows how we find the border of objects without performing complete edge detection. How everything comes together is described in Section 5. Detecting object boundaries in simple objects is discussed in Section 6, while Section 7 handles more complex objects. Accuracy of our methods is reported in Section 8. We offer final remarks in Section 9.

2. Pipeline for Color-Coded Environments

A vision pipeline is a sequence of techniques and algorithms applied in a pipeline architecture (Shaw & Garlan, 1996), where each step in the pipeline manipulates or analyses

Fig. 1. The Bruce et al image processing pipeline

image data. There are certainly many varied methodologies for image-processing solutions and it is also true that robotic vision systems reflect this variety depending on the specific application context of the system. However, across a variety of problem domains within mobile autonomous robotics, a similar pipeline has emerged as a standard not only as a

standard in RoboCup (Ogihara et al., 2005; Veloso et al., 2005; Chalup et al., 2004; Chen et al., 2003), but also in other robotics applications (Halme et al., 2000; Kak & DeSouza, 2002).

This pipeline was first described by Bruce, Balch and Veloso (Bruce et al., 2000) in the context of the RoboCup competition and is shown in Fig. 1. The first stage in the pipeline is image segmentation (b) where each pixel in the image is labeled as one of a set of color classes. Pixels that look, for example, blue, are labeled as belonging to the class blue. After the image is segmented, it is passed to a blob-former (c). Blobs are groupings of connected pixels that all belong to the same color class. Each blob can then be analyzed to determine its properties or relation to other blobs. This is the object-recognition stage (d). The Bruce et al. pipeline is certainly not the only pipeline used by mobile, autonomous robots. Indeed, even within the RoboCup competition we have seen many interesting and significant variations in recent years, such as the one in the German team code (Röfer et al., 2005). However, the

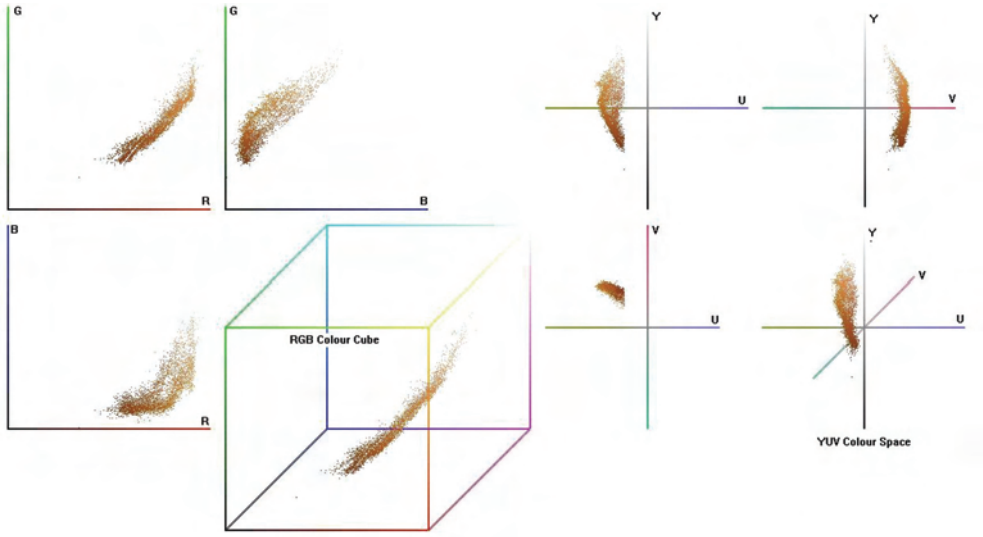


Fig. 2. All the (orange) ball pixels in Fig. 3 mapped to the RGB colur spac (left) and the YUV-colour space (right). Note that the shape in space defined as orange is not regular

Bruce et al. pipeline does possess certain advantages that make it an extremely popular choice. Firstly, it is very efficient — it requires only one pass over the raw data and one pass over color segmented data in order to complete the object recognition task. It is relatively easy to implement and there are third-party libraries available that implement some of its functionality. Another advantage is that it is also relatively easy to calibrate and to use in various different conditions. So significant are these advantages that even some competitive teams do not deviate far from the pipeline, despite its age (Chalup et al., 2004; Chen et al., 2003).

In the context of real-time autonomous robotic vision, color is one of the only object features that is sufficiently easy to distinguish in order to make the image processing fast enough to keep up with the frame rate of the camera. A pipeline similar to Bruce et al. can even be

used in military hardware in order for missiles to detect targets via the infrared spectrum (Shaik & Iftekharuddin, 2003).

3. Colour Classification

We describe a color classifier tailored for the first part of the pipeline, namely, image segmentation. The classifier described here is as fast as a lookup table, but considerably more compact than any other classifier available (it is under 4KB). Also, in contrast to other classifiers used at RoboCup, its representation is intuitive.
We may represent a color classifier as a function

$$\texttt{colour_class}: YxUxV \rightarrow \text{colors} \quad (1)$$

that given a triplet (y,u,v) produces a color class (a member of a discreet set of color classifications). This function may be easily represented as a single characteristic function for each member of colors. For example, let class_orange: Y×U×V→ {true, false} return true when the pixel (y,u,v) belongs in the class orange.
The set of all pixels in the color space that we would like to classify orange cannot be accurately described by any linear discriminator (refer to Fig. 2). This is because the orange area is not regular. This makes the individual characteristic functions difficult to define.

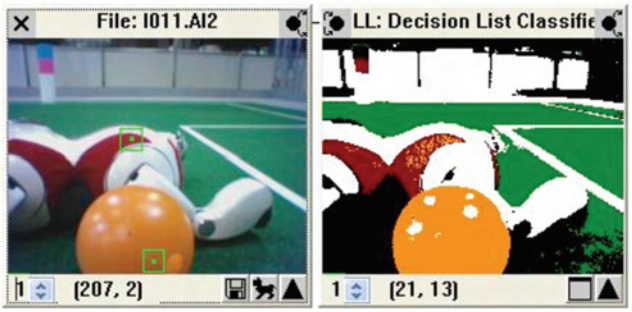


Fig. 3. The highlighted pixels both classify to orange because they have the same component (YUV) values. But the human eye clearly perceives one as red and the other as orange because of the context of the image

It is clear that a more complex knowledge-representation alternative would be required to exactly represent each color class. We certainly lose accuracy whenever we attempt to define the class orange by any linear discriminator. We are, however, quite unconcerned with this loss of accuracy. The reason why such accuracy is unimportant is illustrated in Fig. 3. The two highlighted pixels in the image on the left have exactly the same (y,u,v) tuple even though our eye clearly perceives one as red and one as orange due to the context of the image. Should that pixel be classified as orange or red? Clearly the human eye, to some extent, defines color by what it expects to see given the surrounding context of the image. An ideal classifier would therefore classify a pixel as class orange only when the predominant color of the surrounding shape is orange. The problem, of course, is that image

segmentation is usually the first step in object recognition (and such recursion could be CPU-costly). The first image analysis algorithm does not yet know the context of the object in which the pixel lies. For this reason, even a comprehensive look-up table (an artificial neural network, a support vector machine, or a decision tree), will classify pixels incorrectly. For this reason, we will be content with a classifier that recognizes the core class, where core means those values (y,u,v) that remains on the same class across a wide variety of illumination conditions. We will ignore pixels that could fall into more than one class depending on surrounding context.

Thus a composition of linear descriptions will be a suitable representation. We start with some simple characteristic functions. A characteristic function of each color class can therefore be represented by the projection functions on each of the dimensions Y, U and V. For example, the three projections Y_{orange}, U_{orange} and V_{orange} can be used to approximate `class_orange` in the following way:

$$\texttt{class_orange}(y,u,v) = Y_{orange}(y) \wedge U_{orange}(u) \wedge V_{orange}(v) \tag{2}$$

where $\wedge$ stands for logical AND.

Each of the characteristic projection functions has a domain of 256 values and thus can be feasibly stored in a look-up array (of size 256 bits) that stores 1 if Y_{orange}, is true and 0 otherwise. Thus, there is a look-up table of 256 values for each of Y_{orange}, U_{orange} and V_{orange} which we store in 3 arrays. A characteristic `color_class` function can then be represented by a C++ bitwise AND-operation:

$$\texttt{color_class}(y,u,v) = \texttt{Y[y]\&U[u]\&V[v]}. \tag{3}$$

We can store the look-up for several characteristic functions in compact arrays of `int` type (rather than Boolean type) if they are of convenient width (32 bits is a convenient width because it represents an `int` data type on a modern 32-bit processor). We do this by putting the look-up for the first characteristic function in the first bit (left-most bit) of the value, the second function look-up in the second bit, and so on. If more characteristic functions are required, then we simply increase the size of the data type.

Algorithm 1 Decision List classification.

Require: Arrays `Y [255]`, `U[255]` and `V [255]` where the first bit in `Y[n]`, `U[n]` and `V[n]` is the look-up of the characteristic function for the `n`-th color class. The color tuple to classify is (y, u, v).

Ensure: The color class of (y, u, v).

```
1: val = Y[y] & U[u] & V[v]
2: count = 0
3: while !(val&0x01) && (count < 32) do
4: val=val>>1
5: count++
6: end while
```

However, there are typically few color classes. Thus, we can use more characteristic functions and consider them organized into a hierarchy. The left-most bit represents the highest decision rule. If a pixel is not already classified, we proceed to the next characteristic function, until one rule classifies it. The Machine Learning or Data Mining community would refer to this as a Decision List (Witten & Frank, 2000). This can be implemented with `efficient` shift operations that compute the discrete $\log_2$ of the result to determine the ID of the color class. This entire method of classification is shown in Algorithm 1.

This implementation of our algorithm is similar to that presented by Bruce et al. (Bruce et al., 2000). The innovation of our approach is that each color class may be represented in the array more than once because Algorithm 1 retrieves an index to the class, not the class itself. There are two advantages to this system over Bruce et al.. The first is that our algorithm allows us to discriminate non-rectangular regions in the color space using a decision list format. The second is that the representation of the calibration file is very easy to understand and edit (even by hand). This is advantageous when learning a classifier, validating the classifier, or inspecting the pipeline functionality with tools that link to the AIBO. The Bruce et al. algorithm does not permit more than one linear discriminator for each color class, and therefore does not permit non-rectangular color regions. This makes the Bruce et al. algorithm unsuitable for use in both of the calibration techniques (robust and sparse) that we will introduce shortly.

Algorithm	Average time on AIBo per frame (ms)	Amount of memory consumed (bytes)
Our method	1.71	3060
Complete look-up table	1.41	16777216 (16Mb)
k-Nearest Neighbours	Depends on k but very slow (1ms / k)	fairly small (4 bytes .k)
Support vector machines	1.93	262144 (256Kb compressed)

Table 1. Comparison of our Decision List classifier with other available methods

Our entire classifier is at most 3060 bytes in memory and runs very quickly. Table 1 compares our method with some of the other classifiers that are being used in the RoboCup competition. It is easy to understand why our technique is so much faster than the others. If classification is treated as a spatial problem (k-Nearest Neighbors), then the classifier is required to compute Euclidean distances that involve a square root operation. If it is treated as a decision- tree, then it may process up to 20 levels of conditional statements before a decision is reached. Our method has a runtime cost only marginally larger than the fastest possible solution of the look-up table.

In general, calibration for classification is a supervised learning task; given a set of sample pixels with known color class, derive a classifier to assign a color class to future pixel values. It is important to recognize that we may not encounter every possible (y, u, v) tuple in the training set, so the classifier must generalize. We consider two types of calibrations possible: *robust* and *sparse*. Let P be a training set of n images and $Orange_i$ be the set of pixels that we wish to classify as orange (for example) within the i-th image. Then an ideal robust classification for the class orange is:

$$\texttt{class_orange}\,(y,u,v) = \text{true} \Leftrightarrow (y,u,v) \in \bigcup_{i=1}^{n} Orange_i\,. \tag{4}$$

That is, if a pixel with values (y ,u, v) is recognized as orange in *any* of the images in the training set, then the classifier should label this pixel as class orange for any future images. Of course, an ideal robust classifier may not be possible (the same (y, u, v) tuple may be assigned to two different classes in the training set even within the context of the same image). An ideal classifier may also suffer from over-fitting (Witten & Frank, 2000). In practice, we weaken the condition by asserting that the classifier should label a pixel as class orange if it is recognized as orange *more often* than any other color class.
By contrast, an ideal sparse classification for the class orange is:

$$\texttt{class_orange}\,(y,u,v) = \text{true} \Leftrightarrow (y,u,v) \in \bigcap_{i=1}^{n} Orange_i\,. \tag{5}$$

That is, we label a pixel (y, u, v) as orange only if it is recognized as orange in *all* of the images in the training set. Again, sometimes it is necessary to weaken this condition. In practice we label a pixel as orange, if it is recognized as orange in *most* of the images in the training set.
There are both benefits and drawbacks to each of these two calibration methods. Because we are aiming for versatility to illumination conditions, we take advantage of sparse classification.
Each of our characteristic projection functions, as described above, is capable of storing any pattern of 255 bits. However, for the task of calibration we restrict this to a continuous block of 1's any where within the domain of the function. We label the lowest positive bit for a particular projection (y, u or v) and class (*COLOR*) as $Min_{proj,COLOR}$. Similarly we label the highest positive bit $Max_{proj,COLOR}$. This means that each characteristic projection function is essentially testing the clause

$$(\,Min_{proj,COLOR} \leq x\,) \wedge (\,x \geq Max_{proj,COLOR}). \tag{6}$$

Therefore each characteristic function may be written

$$\begin{aligned} \texttt{class_COLOR}\,(y,u,v) \;=\; & (\,Min_{Y,COLOR} \leq x\,) \wedge (\,x \geq Max_{Y,COLOR}) \\ & \wedge (\,Min_{U,COLOR} \leq x\,) \wedge (\,x \geq Max_{U,COLOR}) \\ & \wedge (\,Min_{V,COLOR} \leq x\,) \wedge (\,x \geq Max_{V,COLOR}). \end{aligned} \tag{7}$$

It is therefore this representation, in the form of a decision list (Witten & Frank, 2000) that we expose to the user. A typical calibration file has the following format:

```
Colour_ID_1 Min_Y Max_Y Min_U Max_U Min_V Max_V
Colour_ID_2 Min_Y Max_Y Min_U Max_U Min_V Max_V ...
```

Of course, we are limited in the number of characteristic functions we can apply by the size of the selected data type, as explained above. In our case we are limited to 32 characteristic functions which, of course, may be increased if a larger data type is used. The results of applying one such characteristic function can be seen in the screen-shot in Fig. 4. Here the Y, U and V channels are calibrated separately to produce the overall characteristic function.

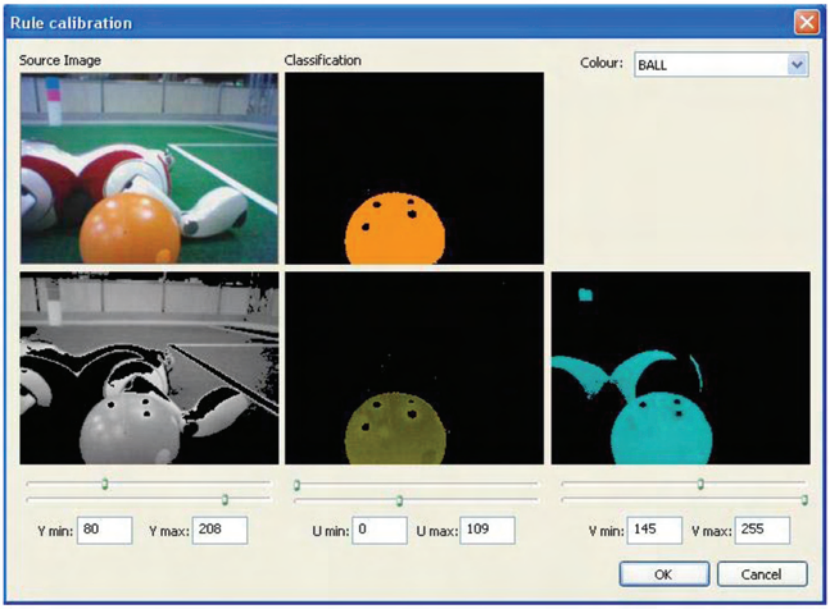


Fig. 4. We may calibrate for each colour by separately examining the Z, U and V components and finding the projections for each characteristic function. This image shows our manual calibration tool allowing the user to select the *Y* (grey-scale in the image), *U* (yellow-scale in the image) and *V* (blue-scale in the image) projections for the class orange separately

Although the rule format restricts us to linear discrimination within the color space, significant flexibility is achieved by the decision list format as Fig. 5 illustrates. Image (a) represents the class orange that we want our classifier to learn. In Image (b) the bounding rectilinear area minimally surrounds the class orange. This is the optimal linear discriminator that completely contains the class, but it does not give a very good solution. There is a large area that our classifier will label orange that is not orange. By using a decision list format we can do much better. We first find a characteristic function for the two shaded areas in Image (c), and label these pixels 'not orange' (i.e. unknown). The while loop in Algorithm 1 will only continue evaluation until a characteristic function returns true. Therefore, if the shaded areas in (c) are tested before the rule in (b), then pixels within them will not be classified as orange even though the linear discriminator in (b) would have classified them so. In this way it is possible to obtain relatively good classifications of colors. Of course, if we are training a sparse classifier instead of a robust one, we will not be interested in a box that completely surrounds the color class. Instead we will want a discriminator that contains the core of each class. In this case linear discriminators are more than adequate. Of course we may require more than one characteristic function to adequately describe the core of each color class (Image (d) in Fig. 5).

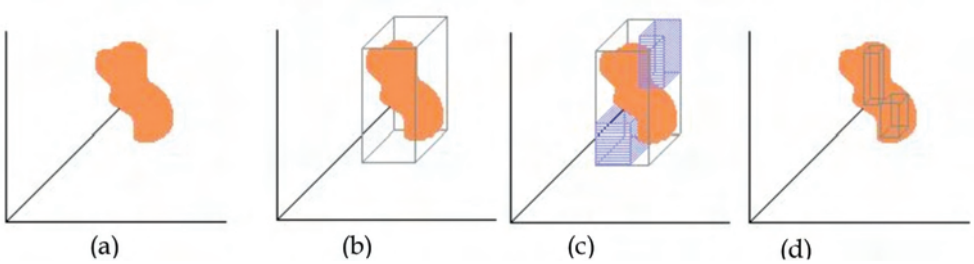


Fig. 5. (a) Colour classes cannot be described by orthonormal rectangular regions in the color space. Therefore (b) classification by linear discrimination is typically bad because it labels many pixels that are not in the class. (c) We may use linear discrimination in combiantion with decision lists to do a better job. If the shaded regions are labelled not in the class, and are higher in the list, pixels within them will not be labelled as belonging to the class. (d) More than one charcteristic function can be used for each colour class. This is particularly useful for sparse classification where only the core of each color is required

4. Optimized Edge Detection

Edge detection is often ignored in dynamic computer vision applications due to the high runtime cost associated with sliding a processing window over an entire image. We have obtained (Lovell, 2007) very optimized methods for edge detection based on effective methods such as Canny's and Sobel's. While our optimizations considerably improve the performance of the Sobel's algorithm, and the resulting algorithm is an order of magnitude faster than Canny's, but it is still computationally expensive. Here we focus on a second alternative that enables us to delay edge detection until it is required (late edge detection). By delaying the edge detection phase we have found it possible to use edge detection as a fundamental tool in our image-processing pipeline because only the areas of the image where edge detection is required will be examined for edges.

4.1 Late Edge Detection

The challenge is to identify interesting sections of the image where edge detection would be useful, rather than apply the difference and window testing of Canny's or Sobel's methods to every pixel of the image. Edge detection is usually an early step in the image-processing pipeline so it is unlikely that we will have a large amount of contextual data on which to base such a decision. But, assuming that we can identify some points inside interesting objects, it is then possible to use edge detection to locate the edges of that object and use them for feature extraction. For now, we will simply assume that we have identified $p = (x, y)$ as a pixel that is contained within an object for which we need to find the edges. We discuss here two techniques depending on the amount of edge information that is required.

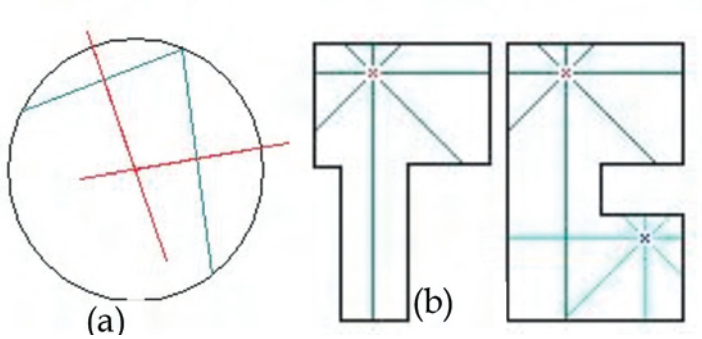


Fig. 6. (a) Vectorization of a circle does not require all the edge points to be known. Any three points on the boundary of the circle can be used to loate its centre by the perpendicular bisectors method as shown in this figure. (b) Patial edge detection on a convex shape (such as the left one in (b)) may not produce enough points to correctly identify and parameterize the shape. If the shape is not convex, further sample points may be needed, or complete edge detection may be requried

Full or complete late edge detection is a technique we use when a complete description of the edge of the object is required. This renders traditional edge detection on a relevant object within the image. However, it is not always necessary to find the complete edge of each object. For example, we are only required to know three points on the edge to find the correct parameterization of a circle (see Fig. 6). In this instance, we use a partial late edge detection that can find n points on the boundary without actually tracing the entire boundary. The difference between the two algorithms is illustrated in Fig. 7. We describe the partial edge detection first, because the complete one will build on some of these techniques.

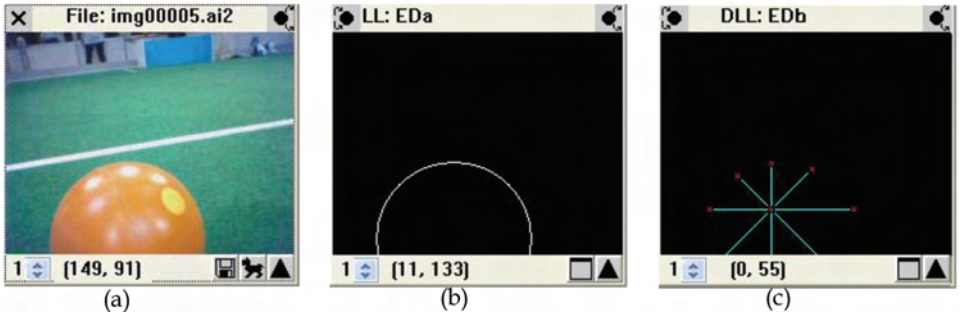


Fig. 7. Full Late Edge Detection on an image (a) results in (b) a full list of pixels that compose the edges in the image. (c) Partial Late Edge Detection locates only a subset of boundary points for an object

4.2 Partial Late Edge Detection

The idea of partial edge detection is that, given some seed point $p = (x, y)$ that we know to be within the boundary of an object, we wish to find a set E of n pixels that lie on the boundary.

To do this we cast n evenly spaced rays out from p, examining each pixel on the ray as we come across it. Each pixel is compared with its neighbors to check the gradient change in intensity. By examining the pixels along a ray in this way, we are essentially performing a Sobel's gradient comparison. Edges are therefore detected well when they are approximately orthogonal to the ray — which is most of the time if the shape is convex.
Of course, if the shape is not convex then we may not be able to gather enough information to parameterize it in this way (see Fig. 6 (b)). In this case, casting rays from a second point p_2 (or more) may sometimes be sufficient for parameterization. More complex shapes will require our second method. By using the Manhattan distance on the color space, we have been able to successfully handle edge detection even in blurry images (Lovell, 2007).

4.3 Complete Late Edge Detection

Sometimes it is not adequate to know only a sample of the boundary points. For example, for vectorisation we require an entire list of spatially connected edge pixels that represent the boundary of an object. Although our method for this full late edge detection is slower than a partial edge detection, it is still significantly faster than traditional edge detection that must be performed on the whole image.

Let $B(p, I) \rightarrow p'$ be a standard border following algorithm that, given a pixel p on the border of an object in a raster image with borders marked I, returns the next pixel around the border of an object p'. Our late edge detection algorithm is then defined by Algorithm 2.

Algorithm 2 Full late edge detection.
Require: A source pixel p that is within the spatial boundary of an object to identify in image (with no marked edges) I.
Ensure: The complete list of pixels E that make up the boundary of the object.
1: Trace any ray from p to find an edge as described in Section 4.2 and label this pixel s.
2: Let the current pixel be c.
3: **while** $c \neq s$ do
4: Apply any edge detection window to locate borders around c.
5: $c = B(c, I)$
6: **end while**

Of course in Line 4 we may use Sobel's window (or optimizations of it (Lovell, 2007)). We show some images in Fig. 8 that illustrate the results of this algorithm. The edge detection is complete in that a full list of pixels that compose the border of a particular object are discovered, but the algorithm does not need to examine any unnecessary pixels to do this. Irrelevant sections of the image are never examined because the algorithm uses a border follower.

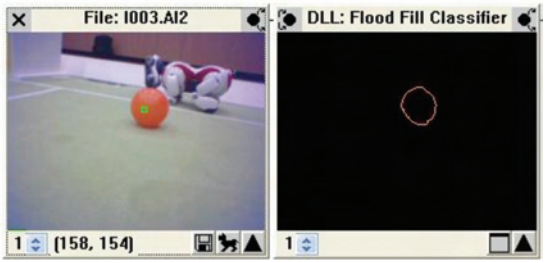


Fig. 8. A full Late edge Detection on a single object within an image reveals the boundary of that object without examining any unrelated areas of the image. In this figure we show a full edge detection on the ball, starting from the pixel indicatedy the green in the source image

One of the problems associated with this technique arises if the edge detector is unable to form closed contours. This issue can be somewhat avoided by a sensitive calibration (that is, one that forms thick edges). However, occasionally we will be forced to abandon an attempt at identifying the edge of the object. By bounding the pixels in the edge of each object we can abort unsuccessful attempts.

4.4 Border Following.

There are many standard border following algorithms that we can apply as B in Algorithm 2, all of which are extremely fast ($\Theta(n)$) on the number n of pixels in the border). We describe here one of the simplest for completeness. This is a standard algorithm that operates in 8-connected space, but it can be easily modified to work in 4-connected space. Let D (p, d) be a function that returns the next pixel from p in direction d. Let the directions be defined as in Fig. 9.

Fig. 9. Direction definitions for the border following method in Algorithm 3

5. Illumination Independence

We now introduce an efficient object recognition system that is robust to changes in color intensity and temperature. Our algorithm will use our edge-detection methods to obtain a description of the boundaries of objects. A list of features in the boundary of the object, plus its color, are usually sufficient for object recognition of landmarks and the ball in RoboCup and for many other object recognition tasks (Lovell, 2007).
Our algorithm will provide this list of features as a basis for object recognition in a wide

variety of illumination conditions without re-calibration. It can do this because it does not rely solely on color classification in order to form blobs. We present two variations of our method. One variation runs extremely quickly but is only able to find simple shapes. The other variation is slightly more processor-intensive but will recognize an arbitrary shape.

Algorithm 3 A simple 8-connected border following algorithm.
Require: Initial pixel p that lies on the border of an object in image (with edges marked) I. The direction dir returned from the previous call.
Ensure: The next pixel p' in a counter-clockwise direction around the border. The direction dir' to use in the next call.
1: Initialise $dir = 7$ on first call.
2: $p' = p$
3: **if** $dir\%2 == 0$ **then**
4: $dir = (dir + 7)\%8$
5: **else**
6: $dir = (dir + 6)\%8$
7: **end if**
8: **while** p' is not on border **do**
9: $p' = D(p', dir)$
10: $dir = (dir + 1)\%8$
11: **end while**
12: $dir' = (dir - 1)\%8$

It is well accepted that edge detection algorithms are far more robust to changes in the temperature and intensity of light than color based segmentation algorithms. This is easy to illustrate. Consider Fig. 10 which illustrates the effect of varying the illumination intensity (a) and temperature (b) of a common scene in RoboCup. Although the edge information in the images is not lost until the illumination levels become extremely low, the robust color calibration becomes useless quite quickly.

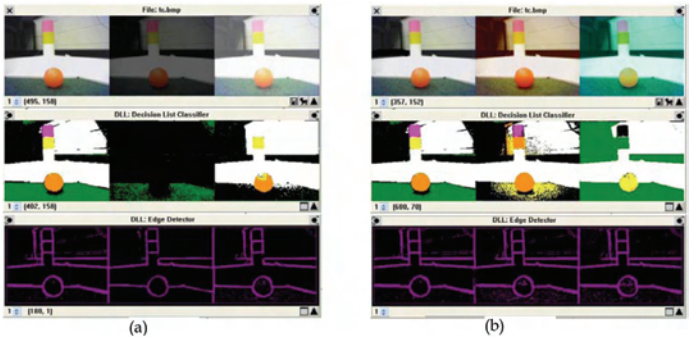


Fig 10. This figure shows the result of varying (a) illumination intensity and (b) color temperature on a common scene in RoboCup. The top rwo of images is the source image, the middle row is a robust classification and the bottom row is our edge detection routine. Note that although variation in illumination conditions is detrimental to color segmentation, it does not particularly affect edge detection

It is apparent then that edge detection, rather than color segmentation, is a better basis for object recognition systems if robustness to dynamic lighting conditions is a requirement. However, edge detection on its own will not yield sufficient information quickly enough. For example, to find the ball in the binary edge images in Fig. 10 we would need to run a circle detection algorithm such as the Hough transform. This would be far too slow for our purposes. Instead, our method works by combining edge detection with the sparse classification technique introduced earlier.

5.1 Building a Sparse Classifier

While it is extremely unlikely that a pixel-color classifier can be built for variable illumination conditions, we may, however, train a sparse one as long as there is not too much variation. Fig. 11 shows how this is possible. As the lighting conditions vary, the perceived color changes in a non-predictable way ((a) and (b)). However, as long as there is some overlap we may classify only the pixels in the overlap section as orange (c). We therefore have found a core class orange that contains pixels that are perceived as orange across both images. The classification itself (c) would be poor if we were to use it for object recognition purposes, but we do not wish to use it directly in this way.

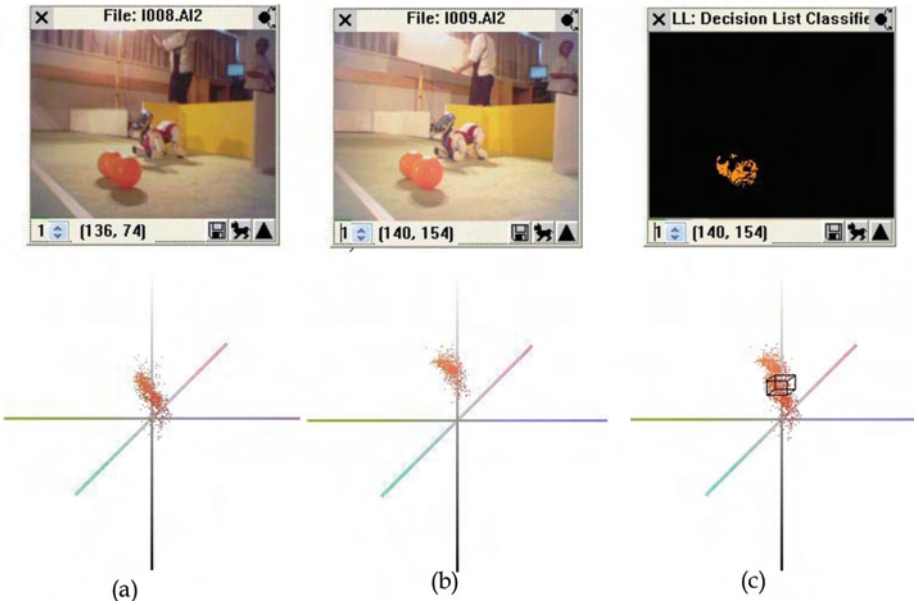


Fig 11. This figure shows the result of varying illumination conditions on the color space. For both images (a) and (b) a perfect calibration was made and every pixel that was labelled orange is plotted in the three dimensional YUV space. Notice that the location of the orange colour class shifts in the space as the illumination changes. We can find the core of the colour class orange (c) by only classifying pixels that are labelled orange in both images. This leads to a poor robust classification, but a good sparse classification

We see now how it is possible to train a sparse classifier for dynamic illumination conditions. We simply widen our set of training images to include many images from different lighting environments and, in the manner described above, train a sparse classifier to label only the pixels that are at the core of each color class. Of course if the lighting conditions are too variable then the intersection operation of the sparse classifier Equation (5) will yield an empty set. In this case we should be less ambitious with our illumination conditions.

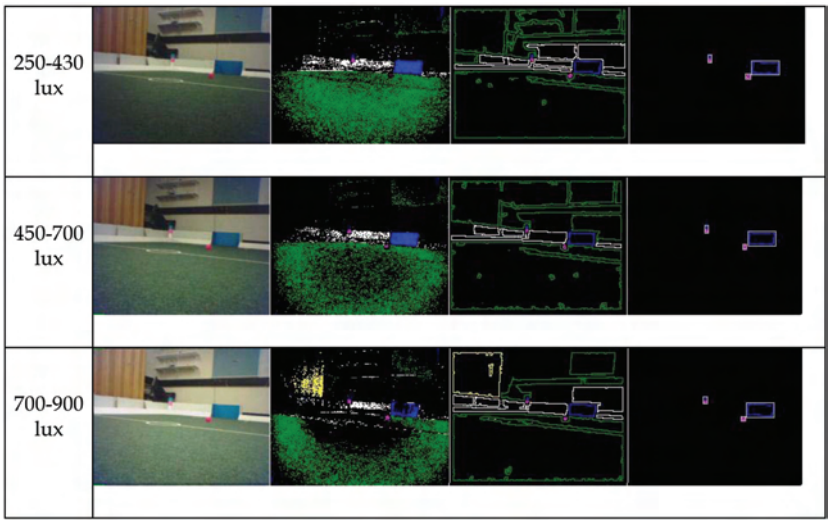


Fig. 12. The basic algorithm for illumination-independent object recognition. A sparse classification (colimn 2) is used along with a border-following algorithm (column 3) to locate regions within the image. These regions can be analyzed for objects of interest (column 4). Note that the system uses the same classification in each illumination condition with accurate results

5.1 Combining Edge Detection with Sparse Classification

One initial algorithm for illumination-independent object recognition is shown in Algorithm 4. In this algorithm we use the list of labeled pixels as seed points to find the border points of each object in the image. Although we can not be sure that every pixel that is part of, for example, the ball, will be labeled as orange, we can be certain that some of them will. Therefore to find the border points of an object we start at a seed point and iterate over pixels in any direction until a previously identified edge is found (Line 4). Once we find an edge we may border trace using the algorithm in the previous section (Subsection 4.4) to obtain the entire list of pixels in the border.

Algorithm 4 Basic illumination-independent object recognition.
Require: An image I.
Ensure: A list of the border points E_o for each object o in I.
1: `EdgeDetect`(I)
2: `SparseClassify`(I)
3: **for** all classified pixels p in I **do**
4: Iterate across pixels in any direction until an edge pixel e is found
5: **if** e does not belong to the border of a known object **then**
6: `BorderTrace`(e) to obtain a list of border points E
7: Assign new object o with edge E, E_o
8: **end if**
9: **end for**

This process renders an image segmentation that works in a similar fashion to other region-growing techniques (Wan, 2003) except that it has the advantage of being illumination-independent. Refer to Fig. 12 where each row in the table shows the process working under a different illumination condition (but with the same calibration file). The first picture is the source image for the row, the second shows the results of a sparse classification step, the third shows the result of Algorithm 4 and the final image shows the object recognition step. In each case the ball, goal and beacon are found correctly, despite the large variation in illumination.

	Component	Average time on AIBO
Our basic pipeline	Sparse classification	1.71 ms
	Edge detection	20.90 ms
	Border following	17.71 ms
	Object recognition	5.87 ms
	Time taken for 30 frames (1s of data)	1385.70 ms
The Bruce et al pipeline	Robust classification	2.42 ms
	Blob forming	9.13 ms
	Object recognition	6.21 ms
	Total per frame	17.76 ms
	Time taken for 30 frames (1s of data)	532.80 ms

Table 2.The runtime cost of basic illumination-independent object recognition is quite high compared to the Bruce et al. pipeline. A large component of this cost is the edge detection and border following steps; therefore, optimizing these steps will improve the runtime cost of our algorithm

The disadvantage of this method is the runtime cost. Refer to Table 2 for a breakdown in the runtime cost of this algorithm, compared to the Bruce et al. pipeline. Most of the execution time is taken in the edge detection step. This edge detection step uses a significantly optimized version of Sobel's edge detection, but remains the main runtime cost of Algorithm 4 (Lovell, 2007). However, in what follows we suggest specialized edge detection

algorithms, rather than a general box to include in the pipeline. The algorithms we will describe significantly improve the efficiency of Algorithm 4 to make it feasible for use in real-time robotic environments.

Notice that Algorithm 4 is not fast enough to analyze 30 fps on an AIBO. In fact, thirty frames analyzed at this speed take 1404.3ms, or just under one and a half seconds. Clearly, a large part of the cost of the algorithm is associated with the edge detection and border following steps. Therefore, we propose two modifications to Algorithm 4 in order to make it execute quickly enough to use in a mobile, autonomous robotic environment.

Both of our alternatives use our late edge detection technique (Section 4). By delaying the edge detection step until it is required, we can perform edge detection only on the sections of the image in which we need it.

6. Detecting Simple Object Boundaries

The first of our alternatives uses the partial late edge detection that we described in Section 4. There are several cases where we do not require a complete description of the boundary of an object in order to parameterize it, as occurs with most simple shapes. The ball in RoboCup is a good example because it is circular. We require only three points on the boundary of the ball in order to apply the geometrical technique of perpendicular bisectors to find the center and radius (Fig. 6). Therefore, we need only to cast three rays in different directions from a seed pixel p that we are sure is part of the ball. Of course, we may wish to cast more rays, or to start with more than one seed point, in order to check if we have actually found the ball or just some other object that also looks orange. This is an extremely fast ball-finding algorithm.

Algorithm 5 Efficient rectangle parameterization.
Require: A set P of points that lie roughly on the boundary of a rectangle of unknown aspect and size but known orientation, θ.
Ensure: The parameters of the rectangle R as four corner points – p_{tl}, p_{tr}, p_{bl} and p_{br}.
1: **for** all $p \in P$ **do**
2: Rotate P by angle $-\theta$
3: **end for**
4: Find the bounding, rectilinear rectangle of P and assign to R.
5: Let $maxS_1 = 0$, $maxS_2 = 0$
6: **for** all x in R **do**
7: Let S = the sum of points that lie on or near x
8: **if** $S > maxS_1$ **then**
9: $maxS_2 = maxS_1$
10: $maxS_1 = S$
11: **end if**
12: **end for**
13: Let $x_1 = \text{MIN}(maxS_1, maxS_2)$, $x_2 = \text{MAX}(maxS_1, maxS_2)$
14: Let $maxS_1 = 0$, $maxS_2 = 0$
15: **for** all y in R **do**
16: Let S = the sum of points that lie on or near y
17: **if** $S > maxS_1$ **then**
18: $maxS_2 = maxS_1$
19: $maxS_1 = S$
20: **end if**
21: **end for**
22: Let $y_1 = \text{MIN}(maxS_1, maxS_2)$, $y_2 = \text{MAX}(maxS_1, maxS_2)$
23: R = { $p_{tl}=(x_1, y_1)$, $p_{tr}=(x_2, y_1)$, $p_{bl}=(x_1, y_2)$, $p_{br}=(x_2, y_2)$}
24: Rotate R by angle θ around the origin

There are other shapes as well that can be found in this way. We present in Algorithm 5 our fast algorithm for determining the parameters of a rectangle provided the angle of orientation is known a priori. If the angle of the horizon in the image is known, then this algorithm can be applied to find (for example) the beacons or goals in RoboCup or a great many rectangular shaped objects in the real world.

We use the method outlined above to find a set P of points that are on the boundary of the rectangle. The points are then rotated in space so that the rectangle is aligned with the axes (Line 2). Each vertical column and horizontal row is then examined to find the lines on which the most points in P lie (Lines 6-21). These lines are labeled as the sides of the rectangle (Line 23). Finally the four corners are rotated back to the frame of reference of the image (Line 24). If the aspect ratio of the rectangle is also known a priori, the algorithm adjusts the rectangle based on the side that had the weakest support value (S). Similar algorithms may be employed to detect any regular polygon.

If enough original sample points are chosen in the initial set P of points, then the algorithm is quite tolerant to noise in the image and even eliminates points where the edge has been determined incorrectly (see Fig. 13 in the next section).

	Component	Average Time on AIBO (ms)
The Bruce et al pipeline	Robust classification	2.42
	Blob forming	9.13
	Object recognition	6.21
	Total per frame	17.76
	Time taken for 30 frames (1s of data)	532.80
Our pipeline using only simple object recognition	Sparse classification	1.71
	Edge point detection	6.71
	Object recognition	1.64
	Total per frame	15.35
	Time taken for 30 frames (1s of data)	460.50

Table 3. Byusing our partial Late Edge Detection in conjunction with the sparse classification, our object recognition pipeline is not only illumination-independent, it executes faster than the Bruce et al. pipeline

6.1 Runtime Performance of Simple Object Detection

In Table 3 we examine the runtime performance of simple object recognition. Note that the two object recognition components in Table 3 should not be directly compared. Much of the work that happens in object recognition in the Bruce et al. pipeline is performed in the shape recognition phase in our pipeline. The most expensive components of this pipeline are edge-point detection algorithms and shape recognition, but the pipeline is still a dramatic improvement on the basic pipeline in Algorithm 4. Indeed, this pipeline is not only

illumination-independant, it is faster than the Bruce et al. pipeline. We could potentially analyze 60 fps at this speed.

7. Detecting Complex Object Boundaries

The second of our alternatives uses the full late edge detection that we described in Section 4.3. The basis of this method was that a point on the boundary of an object would be found in the same way as above, casting a ray from a seed pixel *p* until we reach the edge of the object. We noted in Section 4.3, particularly in Fig. 6, that it might not be enough to simply cast rays from seed pixels to determine the parameters of a complex shape.
We therefore proposed Algorithm 2, a combination of a boundary-following algorithm and a partial late edge detection that could be used to discover and trace the boundary of an object in linear time.

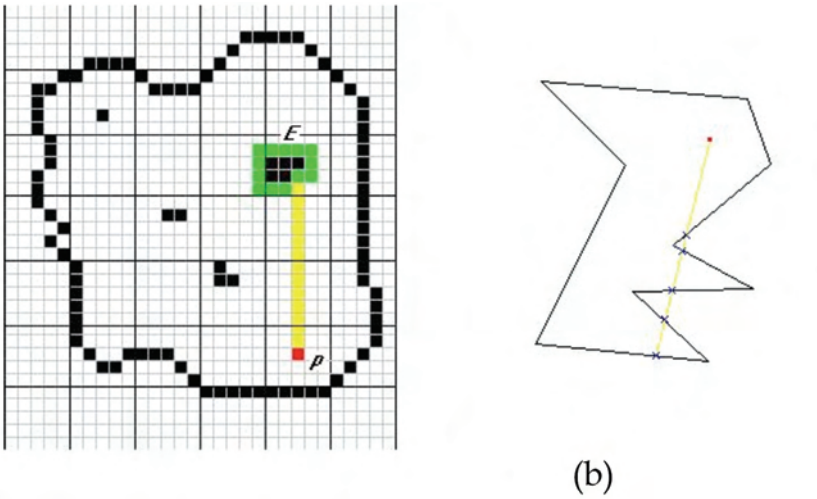


Fig. 13. (a) One of the main roblems with our method is that islands can be found and traced instead of the border of the object. (b) We use a standard polygonal interior test to detect this situation

Algorithm 2 will, under most conditions, return a list *E* of pixels that represents the boundary of the object in which the pixel *p* resides. Due to noise in the image, on occasion, it is possible for *E* to locate an island, rather than the object. Refer to Fig. 13 (a) where the yellow pixels illustrate the ray cast from *p* (the red pixel) and the green pixels indicate the edge, *E*, that has been located. The correct boundary of the object has not been found in this case due to noise in the image.
We may use the standard polygonal inclusion test (O'Rourke, 1998) to detect this situation (Fig 13 (b)).
A ray cast from any point to a point on the edge of a polygon cuts the polygon an odd number of times if that point is inside the boundary of the polygon. We wish to see if our original point *p* is inside the polygon represented by *E*. Therefore the algorithm is simple. As

E is constructed (Algorithm 2), we count the number of times a pixel $q = (x, y)$ is placed in E with $q_x = p_x$ and $q_y < p_y$. If this number is odd then the original point p is inside the polygon E, otherwise it is outside.

Another problem that may be encountered due to noise in the image is that the edge detector does not return a closed contour: that is, there may be gaps in the list of pixels that comprise the edge of the object. If the edge detector is calibrated to be sensitive, (that is, for thick edges), then this is not a frequent problem (Lovell, 2007). When the anomaly occurs it is possible to detect it by bounding the number of pixels that may comprise the edge of an object. In this case, we discard the edge entirely and the object is missed in that frame.

Once the boundary has been correctly determined, it is usually useful to vectorise it. This can be done in linear time (Lovell, 2007). Once a vectorization has been obtained there are several useful and fast analysis algorithms (Lovell, 2007).

7.1 Runtime Performance of Complex Object Detection

Most object recognition tasks will blend a mixture of objects that can be detected using simple object recognition, and objects that must use complex object recognition. Table 4 shows the runtime cost of such a system. Images in these tests were from the RoboCup domain and had a mixture of balls, beacons and goals (representing simple objects) and opponent AIBOs (representing complex objects).

	Component	Average Time on AIBO (ms)
The Bruce et al pipeline	Robust classification	2.73
	Blob forming	8.99
	Object recognition	14.71
	Total per frame	26.43
	Time taken for 30 frames (1s of data)	792.90
Our pipeline	Sparse classification	2.51
	Edge point detection	7.98
	Simple shape recognition	5.06
	Complex shape recognition	10.44
	Object recognition	1.34
	Total per frame	27.33
	Time taken for 30 frames (1s of data)	819.30

Table 4. The final runtime cost of the two pipelines including analysis of both simple and complex objects. Our pipeline executes less than 1ms/frame slower than the Bruce et al. pipeline but provides an illumination independent object recognition system

We make several observations on Table 4, particularly with respect to the introduction of

complex objects (AIBOs). In order to analyze images containing other AIBOs (Lovell, 2007), we must identify their skeleton so the processing that was done on blobs (in the Bruce et al. pipeline) and on complex shapes (in our pipeline) is enough to identify a complete and ordered list of pixels along the border of the AIBOs' uniforms. We do not, however, include a full AIBO recognition algorithm in either pipeline.

This processing represents one extra step for the Bruce et al. object recognition component than for our pipeline. This is reflected in the slower processing time for object recognition for the Bruce et al. pipeline than was seen in Table 2 and Table 3. Our object recognition has no extra overhead because this work is done in the complex shape recognition component.

Different sample images were used for this set of tests (so that AIBOs could be included) so minor variations from the data in the above tables are to be expected. Classification is marginally slower in these images because there are more potential colors to classify and therefore a longer decision list.

We see from this table that our pipeline is essentially equivalent in speed to the Bruce et al. pipeline, being less than 1% slower. Therefore, we have managed to provide an illumination-independent object recognition system for minimal extra cost.

	Object	Occurrences in 300 Images	Recognized (simple)	Accuracy (%)	Recognized (complex)	Accuracy (%)
Stationary Camera (300 Images)	Blue Goal	123	117	95.12%	109	88.62%
	Yellow Goal	119	114	95.80%	107	89.92%
	P/B Beacon	57	56	98.25%	51	89.47%
	B/P Beacon	83	79	95.18%	76	91.57%
	P/Y Beacon	34	31	91.18%	29	85.29%
	Y/P Beacon	62	57	98.39%	54	87.10%
	Orange Ball	212	203	95.75%	192	90.57%
	Red AIBO	87	-	-	63	72.41%
	Blue AIBO	112	-	-	86	76.79%
Moving Camera (200 Images)	Blue Goal	97	86	88.66%	73	75.26%
	Yellow Goal	86	76	88.37%	68	79.07%
	P/B Beacon	52	45	86.54%	31	59.62%
	B/P Beacon	35	29	82.86%	23	65.71%
	P/Y Beacon	39	34	87.18%	27	69.23%
	Y/P Beacon	41	34	82.93%	29	70.73%
	Orange Ball	156	139	89.10%	111	71.15%
	Red AIBO	49	-	-	31	61.22%
	Blue AIBO	54	-	-	29	53.70%

Table 5. The accuracy of our object-recognition system in a set of 500 images: 300 taken from a stationary camera, and 200 contain some degree of blur. The images are taken over a variety of lighting conditions

8. Accuracy of our Method

We have collected 500 images that contain scenes that may be expected in a typical RoboCup game. The image database is divided into two sections. The first 200 of the images are taken from a moving AIBO (and thus contain varying degrees of blur), the other 300 are taken from an AIBO standing still. The images vary in lighting condition (200-1500 LUX, with dynamic shadows) though they are similar enough that a suitable sparse classifier has been found (see Section 5.1).

Table 5 shows the accuracy of our system. There are several things to note here. Firstly, the system performs well — especially given the variable lighting conditions. Even in images taken from a moving camera the system performs well enough to use as the primary sensory input of a soccer-playing robot.

The second thing to notice is that we may determine the extent of the closed contour problem (Section 7) by comparing the accuracy of our system on simple objects with the accuracy achieved when we detect these same objects using our algorithm for complex objects. Typically, we only fail to detect objects due to this problem in less than 10% of images taken from a stationary camera. This accuracy is sufficient for complex shape recognition like AIBO posture recognition [Lovell, 2007]. We fail to detect objects approximately 25% of the time when the images contain blur. This is to be expected — complete edge detection in blurry images remains a difficult problem. Table 5 shows only the positive accuracy (that is, objects in each image that were correctly identified). There were an insignificant number of false positives — less than 10 for all objects in all images — however, this result is not significant to the discussion in this chapter. It is also possible to rule out false positives (Lovell, 2007).

9. Conclusion

We have discussed an approach to object recognition inspired on the vision analysis pipeline. However, the linear organization of the pipeline is a model that propagates the decision of each filter, and therefore, it propagates mistakes. It is natural that after one pass over the pipeline, feedback from the result to one or several of the filters would improve the overall result. For example, finding a large circular orange ball may facilitate finding yellow goals and red AIBOs in the image since now we have information of where the ball is. Also, we have illustrated that not every pixel in the image most be processed by the entire pipeline and thus we can avoid processing some regions of the image.

These remarks suggest two avenues for expanding our work. First, we can have some areas of the image significantly advanced on stages of a vision pipeline whose results may be input to other areas or early filters. Second, running the pipeline with parameters that emphasize speed but coarse results may enable further later executions of the pipeline on the same image with feedback information and adapted parameters. Having a very fast and robust pipeline here means that as CPU-speeds increase, we can run them very effectively. Notice that as resolution of the frames increases linearly, the number of pixels increases quadratically, similarly, as the frame rate increases linearly, the number of pixels that needs analysis increases quadratically. Executing a fast pipeline like ours will enable more reliable and robust systems under even larger illumination variations as we move into faster processors and more reliable hardware.

10. References

J. Bruce, T. Balch, and M. Veloso. (2000). Fast and inexpensive color segmentation for interactive robots. In *Proceedings of the International Conference on Intelligent Robots and Systems*, (2061–2066). IEEE Computer Society Press. ISBN: 0-7803-6348-5.

S. Chalup, R. Middleton, R. King, L. Li, T. Moore, C. Murch, and M. Quinlan. (2004) The NUbots' team description for 2004. In *Proceedings of RoboCup 2004 – Robot Soccer World Cup VIII*, Lisbon, Portugal, pages CD–Rom Proceedings. Springer-Verlag. ISBN: 3-5402-5046-8.

J. Chen, E. Chung, R. Edwards, N. Wong, E. Mak, R. Sheh, M. Kim, A. Tang, N. Sutanto, B. Hengst, C. Sammut, and W. Uther. (2003). A description of the rUNSWift 2003 legged robot soccer team. In *Proceedings of RoboCup 2003 – Robot Soccer World Cup VII*, Padua, Italy, pages CD–Rom Proceedings. Springer-Verlag. ISBN: 3-5402-2443-2.

K. Gunnarsson, F. Wiesel, and R. Rojas. (2005) The color and the shape: Automatic on-line color calibration for autonomous robots. In *Proceedings of RoboCup 2005 – Robot Soccer World Cup IX*, Osaka, Japan. Springer- Verlag. ISBN 3-540-35437-9

A. Halme, K. Koskinen, V-P. Aarnio, S. Salmi, I. Leppnen, and S. Ylnen. (2000). Workpartner – future interactive service robot. In *Proceedings of the Millennium of Artificial Intelligence Conference, 9th Finnish Conference on Artificial Intelligence*, pages CD–Rom Proceedings. Finnish Artificial Intelligence Society. ISBN: 9-5122-5128-0.

A. Kak and N. DeSouza. Robotic vision: What happened to the visions of yesterday? In *Proceedings of the 16th International Conference on Pattern Recognition*, (839–847). IEEE Computer Society Press, 2002. ISBN: 0-7695-1695-X.

N. Lovell. (2007) *Machine Vision as the Primary Source of Input for Mobile, Autonomous Robots*. PhD thesis, Griffith University, Nathan, 4111, QLD, Australia, 2007. Available www.cit.gu.edu.au/˜s2130677/PhDthesis/nLovell.pdf.

Y. Ogihara, Y. Shibata, H. Najima, K. Kii, K. Oda, and T. Ohashi. (2005) Asura: The kyushu united team 2005 in the four-legged robot league. In *Proceedings of RoboCup 2005 – Robot Soccer World Cup IX*, Osaka, Japan, pages CD–Rom Proceedings. Springer-Verlag.

J. O'Rourke. (1998). *Computational Geometry in C*. Cambridge University Press, U.K.

T. Röfer, R. Brunn, S. Czarnetzki, M. Dassler, M. Hebbel, M. Jungel, T. Kerkhof, W. Nistico, T. Oberlies, C. Rohde, M. Spranger, and C. Zarges. (2005) GermanTeam 2005. In *Proceedings of RoboCup 2005 – Robot Soccer World Cup IX*, Osaka, Japan. Springer-Verlag. ISBN 3-540-35437-9

J. Shaik and K. Iftekharuddin. (2003). Automated tracking and classification of infrared images. In *Proceedings of the 2003 International Joint Conference on Neural Networks*, (1201–1206). IEEE Computer Society Press. ISBN: 0-7803-7899-7.

M. Shaw and D. Garlan. (1996). *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, U.S.A. ISBN: 0-1318-2957-2.

S. Shimizu, T. Nagahashi, and H. Fujiyoshi. (2005) Robust and accurate detection of object orientation and id without color segmentation. In *Proceedings of RoboCup 2005 – Robot Soccer World Cup IX*, Osaka, Japan. Springer- Verlag. ISBN 3-540-35437-9

M. Veloso, S. Chernova, C. McMillen, P. Rybski, J. Fasola, F. vonHundelshausen, A. Trevor, S. Hauert, and R. Espinoza. (2005). Cmdash05: Team description paper. In

Proceedings of RoboCup 2005 – Robot Soccer World Cup IX, Osaka, Japan, pages CD–Rom Proceedings. Springer-Verlag. ISBN 3-540-35437-9

S. Wan. Symmetric region growing. (2003) *IEEE Transactions on Image Processing*, 12(9):1007 1015. ISSN: 1057-7149.

I. Witten and E. Frank. (200) *Data Mining – Practical Machine Learning Tools and Technologies with Java Implementations.* Morgan Kaufmann, U.S.A. ISBN: 1-5586-0552-5.

5

Towards Model-based Vision Systems for Robot Soccer Teams

Murilo Fernandes Martins, Flavio Tonidandel and Reinaldo A. C. Bianchi
Centro Universitário da FEI
Brazil

1. Introduction

Since it's beginning, Robot Soccer has been a platform for research and development of independent mobile robots and multi-agent systems, involving the most diverse areas of engineering and computer science. There are some problems to be solved in this domain, such as mechanical construction, electronics and control of mobile robots. But the main challenge is found in the areas related to Artificial Intelligence, as multi-agent systems, machine learning and computer vision. The problems and challenges mentioned above are not trivial, since Robot Soccer is dynamic, uncertain and probabilistic.

A computer vision system for a Robot Soccer team must be fast and robust, and it is desirable that it can handle noise and luminous intensity variations. A number of techniques can be applied for object recognition in the domain of Robot Soccer, as described by (Grittani et al., 2000).

The research of (Grittani et al., 2000) is based only on color information, as well as the research of (Weiss & Hildebrand, 2004) that uses color information to reduce the amount of information contained in each image frame through a called "relevance point filter".

Other researches uses the shape model of the objects to detect on the image, technique generally used in local vision systems. The research of (Gönner et al., 2005), for instance, detects the ball through it's shape model projected on the image, a circumference, but still uses color-only information to recognize the robots.

No matter which technique is used to solve the Robot Soccer computer vision challenge, it must be able to determine position and angle of the robots and the ball with maximum accuracy and minimal processing time possible, because the success of the strategy and control system depends on the information given by the computer vision system.

This chapter extends the work presented by (Martins et al., 2006a), which considers the use of a well known image segmentation technique – the Hough Transform – to locate the mobile robots and the ball on global vision images, taking advantage of the domain characteristics – the robots and ball shape. To implement the Hough Transform technique, which is in most cases implemented in robotic systems using special hardware, only an off-the-shelf frame grabber and a personal computer are used. A new approach to interpret the Hough space is proposed, as well as the method used to recognize objects, which is based on a constraint satisfaction approach.

The chapter is organized as follows: Section 2 describes the most commonly used color spaces in Robot Soccer computer vision systems – RGB and HSI. The Section 3 introduces the Hough Transform. Section 4 describes the system implement in details and Section 5 presents and discusses the results obtained. Section 6 concludes this chapter and presents suggestions for future work, as well as the work that has already been done to enhance the system.

2. Color Spaces – RGB and HSI

Color is the output of a vision system to the perception of different wave lengths reflected by the objects in an observation. Color spaces are representation models that define the primitives do describe the colors. Many image editor softwares work with different color spaces, including the RGB (Red, Green, Blue) and HSI (Hue, Saturation, Intensity) color spaces (Forsyth & Ponce, 2002). The RGB color space is characterized by a cube with R, G and B axis. The Fig. 1 illustrates the RGB cube. Notice that similar color can be represented by many RGB values, what makes hard to build a threshold color filter based on the RGB parameters.

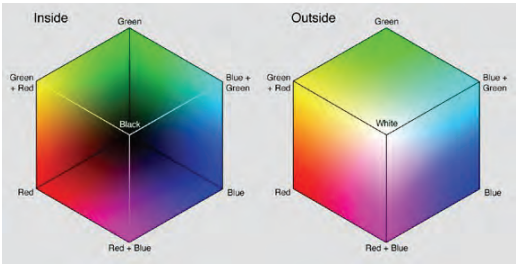


Fig. 1. The RGB cube representation

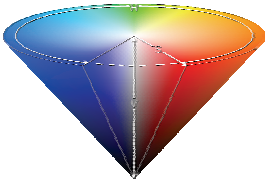


Fig. 2. The HSI conical representation

The parameters R, G and B represents three channels of colors red, green and blue, where, generally, each channel is represented by an 8-bit, implying in 255 different levels of color. The merge of the three channels results in the well known 24-bit true color.
The HSI color space represents a color by Hue, Saturation and Value, where Hue is the color, Saturation identifies how strong the color is and Intensity represents the luminosity intensity of the color. The Fig. 2 shows the conical HSI representation.

The HSI color space is represented by a cone, with a 0 to 360 degrees interval for Hue and a 0 to 100 per cent interval for Saturation and Intensity. Notice that, for any value of Hue, if Saturation is set to 0, then only colors in a gray scale level can be described.
As described by (Penharbel et al., 2004), the HSI color space is better to build a filter to separate colors independent of the luminosity than the RGB color space. The main challenge of using HSI color space is the need of converting the image pixels from RGB, which is tipically the format delivered by ordinary frame grabbers, to HSI. This conversion has a computational cost that can avoid the use of HSI color space and many researches tend to use the RGB color space on their filters, like the filter described in (Bianchi & Reali-Costa, 2000).

3. The Hough Transform

The Hough Transform (HT) (Hough, 1959) is one technique of image segmentation used to detect objects through models adjustment. This technique requires that an object class is determined and such class must be able to describe all possible instances of the referred object. The parameterization of an object class defines the form of this object, therefore, variations of color on the image, or even on the objects, do not affect the performance and efficiency of the HT. To detect objects on an image, the HT tries to match the edges found on the image with the parameterized model of the object.
The HT has rarely been used in robotic systems operating in real time and, when used, it generally needs specific hardware due to its computational complexity. In Robot Soccer, the HT is only used to locate the ball – but not the robots – as described in (Gönner et al., 2005) and (Jonker et al., 2000).

3.1 Circles Detection with Hough Transform

Circles are a very common geometric structure in Robot Soccer since all objects can be represented by one or more circles. The ball for instance is a sphere, but it becomes a circle when projected on the captured image. In FIRA Mirosot and RoboCup Small Size categories, the robots are identified through labels on their upper part. These labels can be of any form and must contain determined a priori colored areas to allow distinction among the robots of different teams. The label used in this work has two circles at 45 degrees with respect to the front of the robot, which complies with FIRA rules (Fig. 3). These circles have the same diameter of the ball used.

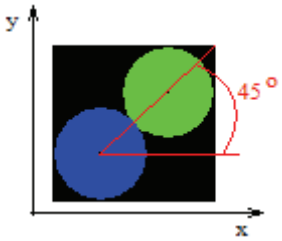


Fig. 3. Label used to identify the robots

Circles are parameterized by the triplet (x_c, y_c, r), which defines a set of equidistant points in r from the central point represented by the Cartesian coordinate (x_c, y_c). A circle can be parameterized using polar coordinates by the equations:

$$x = x_c + r\cos\theta \tag{1a}$$

$$y = y_c + r\sin\theta \tag{1b}$$

In this manner, for known values of the triplet (x_c, y_c, r) and varying the angle θ in all 0 – 360 interval, a complete circumference can be drawn. The space of parameters, also called Hough Space, is three-dimensional. In Robot Soccer, objects move in two dimensions since there is no depth variation of objects on the image, allowing a constant value for radius r to be employed. Thus, the space of parameters becomes bidimensional and it is represented by the point (x_c, y_c).

3.2 The Hough Space

To detect circles of constant radius, on an image that contains only (x,y) edge points, using the HT consists on determining which points belong to the edge of the circle centered in (x_c, y_c) and of radius r. The HT algorithm determines for each edge point of the image a set of possible centers in the Hough Space, set which will be defined iteratively by the variation of θ. Equations (1a) and (1b) become:

$$x_c = x - r\cos\theta \tag{2a}$$

$$y_c = y - r\sin\theta \tag{2b}$$

Fig. 4 demonstrates the algorithm execution for three points on a circle edge of the image on the left, and the respective Hough Space generated is shown on the right. Three circles of radius r drawn on the Hough Space, from 3 points on the edge of the circle with center (x_c, y_c) on the image, intersect themselves in only one point, which is exactly the central point of the circle on the image. Each edge point on the image generates a circle in the Hough Space. Each edge point of this circle in the Hough Space receives a vote. The greater the number of votes a point receives, the greater the probability of this point being a circle center. These points with greater probability are relative maximum of the Hough Space and they define the centers of existing objects on the image.

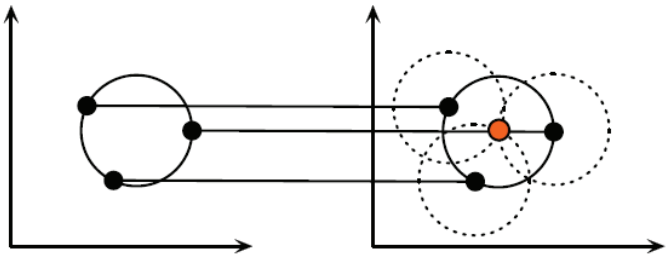


Fig. 4. Example of Hough Space generation

4. Description of the Implemented System

The system described in this section was developed to work in two different situations: first, when the two teams have the same kind of robot label on the top of them and second, when the opponent have a different label. In the first case, the system will recognize the opponent in the same way it recognizes its own players. In the second case, the detection of the opponent robots is done by the color information. Both cases are described below.

4.1 A Model-based Approach to Recognize the Objects

The implemented computer vision system has seven stages, as follows: image acquisition, background subtraction, application of edge filter, Hough Space generation, determination of high probability points to be centers of circles on the image and objects recognition (robots and ball).

4.1.1 Image Acquisition

The image acquisition system consists of an analogical camera with a composite video output and an off-the-shelf video frame grabber based on Bt-878 chipset. This equipment can acquire up to 30 frames per second. In this work, two image resolutions were used: 320x240 and 640x480 pixels, both with color depth of 24 bits. Thirty pictures were captured for each resolution. Fig. 5-left presents one of the images used.

4.1.2 Background Subtraction

As previously mentioned, only the edges of the image are relevant to the HT. Each point of the edge is an iteration of the HT algorithm. To optimize the performance of this algorithm, a simple method of background subtraction was used. It computes the difference between the image captured and a background image, without the moving objects. The background image is updated each frame time, using a method known as Running Average (Piccardi, 2004), according the equation below:

$$B_{i+1} = \alpha F_i + (1-\alpha) B_i \tag{3}$$

where the background image is represented by B, the captured image is represented by F and α is a learning rate used to determine how fast static objects become part of the background image. Although the method is simple, it is efficient for this application because the background does not suffer major modifications. The final image contains only the mobile objects, resulting in relevant edges only. The background image is presented on Fig. 5-center and the result of the background subtraction can be seen on Fig. 5-right.

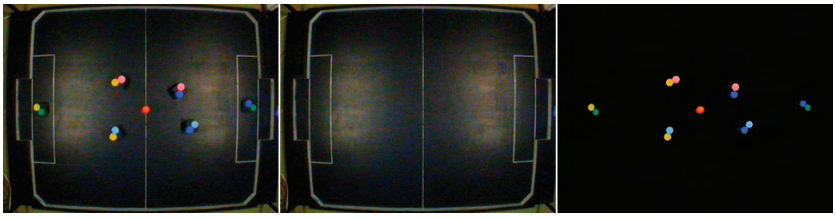


Fig. 5. (left) Captured image containing a ball and two complete teams of robots (center) background and (right) result of background subtraction

4.1.3 Canny Edge Filter

There are many different techniques capable of extracting edge information on an image, as described by (Forsyth & Ponce, 2002). The present work uses a well known technique for edge detection, the Canny filter (Canny, 1986), which produces binary images. Fig. 6-left shows the result of edges detection with the Canny filter on an image that the background was subtracted (Fig. 5-right) and was converted into gray scale (to lower the processing time).

4.1.4 Hough Space Generation

The Hough Space can be generated from the resultant binary image produced with the Canny filter. The generation of the Hough Space with the algorithm described in Section 2 is correct, but not efficient. Although this algorithm generates the Hough Space correctly, it does several redundant iterations to produce the points of possible circle centers. This redundancy happens because when varying the angle θ, it generates 360 centers points for each edge point with decimal precision. However, the digital images are composed of pixels located on a grid, where each pixel position is an integer number. Therefore, the use of decimal precision is irrelevant and redundant.

To eliminate redundancy, an algorithm for circle drawing proposed by (Bresenham, 1965) was used. This algorithm determines points of a circle using only integers, through the addition of a value determined a priori. Moreover, the algorithm takes advantage of points symmetry in a circle: points position is computed in only one octant and, by symmetry, the points of the other 7 octants are determined, without repetition of previously drawn points. In this way, the processing time for the generation of the Hough Space is minimized, allowing the use of applications in real time. The Hough Space generated for the edge points of the image in Fig. 6-left can be observed in Fig. 6-right and Fig. 7 shows the same Hough Space in a 3D view, where the Z-axis represents the probability of a point being a center of an existing circle on the image.

4.1.5 Circles Determination from the Hough Space

After the Hough Space is generated, the following step is to find out the points that received more votes in the space in order to detect the possible circle centers (x_c, y_c), with r kept constant. It is possible to consider r constant because the distance between the camera and the field is greater than the field dimensions, and the radius variation is less than 5%. This fact also implies in no significant distortion in the images.

This stage is the one that presents greater problems in terms of circle detection. As any edge point generates a set of points (in a circle) that receives votes in the Hough Space, there might be misrepresenting votes producing false relative maximums.

The implemented algorithm verifies whether a voted point reached a minimum number of votes – determined a priori – as the Hough Space is being generated. If a point exceeds this threshold, it is stored only once in a vector of possible centers. At the end of the Hough Space generation, this vector stores the number of votes for each point that exceeded the minimum number of votes.

To guarantee that all points representing real circle centers on the image are in the vector, a low minimum number of votes is defined. But, because of this low threshold, there might be false relative maximums in this vector. Another problem is that, due to the nature of the HT

and the Canny filter, a circle in the image may generate several possible centers, lying close to each other.

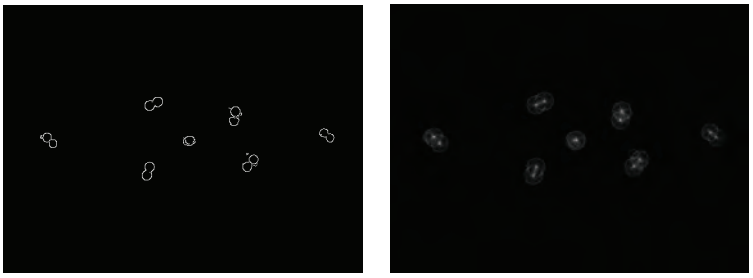


Fig. 6. (left) Result of the application of the Canny filter on Fig. 5-right and (right) Hough Space generated for the edge points, where brighter points have more votes

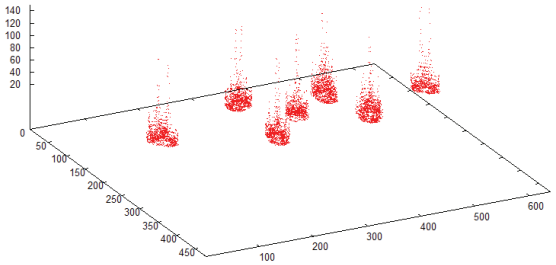


Fig. 7. A 3D view of the Hough Space shown on Fig. 6 (right)

The first step to separate the false center from points where real circle centers are located is to order the points in the vector by the number of votes received using the Quicksort algorithm. After this ordination, the point in the first position in the vector represents the global maximum and is considered a circle center and is inserted in a new vector, the vector of centers.

As the real center of a circle can be defined as the point that was voted the most, and overlapping between two circles do not occurs, all the points that the Euclidean distance to the first center is less $2r$ can be removed from the vector of possible centers. After this, the second position will represent the second maximum and can be considered as a center, and so on.

The algorithm continues until iterations reach a maximum number of circles determined, or until the end of the vector of possible center is reached. This algorithm results in a vector with points distant enough from each other to be considered different circles.

4.1.6 Object Recognition

This stage is the only one that considers color information and image illumination. Therefore, to successfully recognize objects it is necessary to distinguish them, what can only be done when the main colors, defined by the competition rules and different for each team, as well as the secondary colors, used in order to differentiate the robots of the same team, are known.

To define the colors in the image first the mean color of each circle is computed, using a 5 x 5 mask centered in the points found during circle detection. Then, these colors are converted from the RGB to the HSI color space. As mentioned in Section 2, in this color space, pixels are described by the triple (Hue, Saturation, Intensity). The Hue component describes the color of the pixel. It ranges from zero to 360 degrees, where zero degree is red, 60 yellow, 120 green, 240 blue and 300 degrees magenta. The Saturation component signals how white color is present, and the Intensity component represents illumination. In this color space, illumination variations do not modify the hue value of a pixel.
Using this color space, it is easy to define the colors in the image: the mean colors of the circles are ordered by the Hue component using the Quicksort algorithm. As the colors in the HSI color space are always in the same order and at least 30 degrees apart, and the number of circles of each color is known a priori, it is very easy to define the colors of the objects. For the circles in Fig. 5-right, the following colors were found: one orange circle (Hue = 21), three yellow (H = 45, 48 and 50), two green (H = 154 and 160), two cyan (h = 200 and 207), three blue (all at 223) and two pink (H = 348 and 354). The histogram of the Hue component of the same image is presented in Fig. 8.

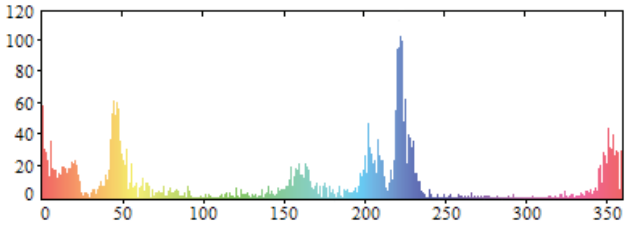


Fig. 8. Histogram for the Hue component of the pixels present in Fig. 5-right

Now that the color of each circle is known, deciding which circle is the ball and which ones are parts of the same robot can be done by solving a problem of constraint satisfaction. According to (Russell & Norvig, 2002) a constraint satisfaction problem is "a special kind of search problem that satisfies additional structural proprieties beyond the basic requirements form problems in general". In this kind problem, the states are defined by the value of a set of variables and the goals specify constraints that these values must obey. In the robot recognition problem, the constraints are that the two circles that are in the robot identification label must be at a fixed distance, $2r$. Another constraint is that each circle of a primary color must be matched with one circle of a secondary color.
To identify which circles belongs to each robot, the algorithm searches in the vector of centers which circles are of a primary color (blue or yellow): this circles are defined as roots of three trees. After defining the roots, the algorithm searches in the vector for circles that are at $2r$ from the primary color circles and adds them as child nodes of the corresponding tree.
If all robots are located distant one from another, this procedure will result in three trees with only one root and one child, defining a robot. Having a center for each labeled circle and the position of the circles known, the algorithm determines the global position (x, y) of the robot on the image and its direction angle in relation to the axis x. As the ball is the only object that can be orange because this color cannot be used for any another object, any orange circle is considered a ball and there is no need to construct a tree to recognize balls.

However, there might be a situation where robots are close to each other, or even touching each other, as in Fig. 9-left. In this case, instead of a tree for each robot, the algorithm will build one tree with three child nodes. It will also build two trees with one child node, as expected (Fig. 9-right). In this case, the algorithm needs to remove child nodes from the tree with three child nodes. To do this, first nodes that are not of a secondary color are removed (it might be another robot's primary color or the ball). And second, the algorithm removes from the wrong tree the circles that are of a secondary color which are already represented in a correct tree. The algorithm will stop when all the trees are correct, representing one robot. The final output of the implemented system can be observed in Fig. 10-right.

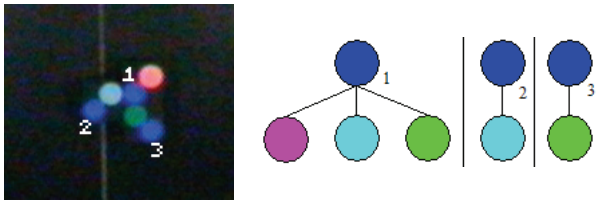


Fig. 9. (left) Image with three robots touching each other and (right) the trees built by the algorithm

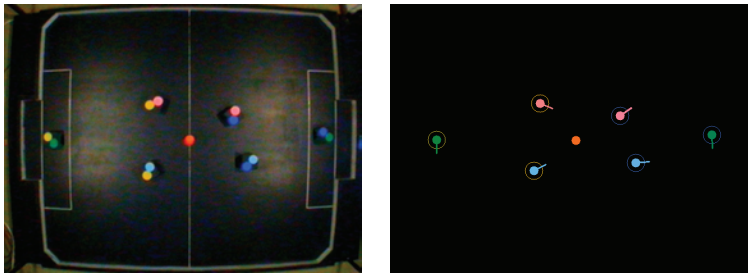


Fig. 10. (left) Captured image containing a ball and two complete teams of robots and (right) result of the execution of the system, showing the computed position of each object

4.2 A Color-based Approach to Recognize the Objects

To detect the opponent robots, the color information is given a priori, but there's no information about the shape of the labels used by opponent. For this case, an approach to recognize the robots by their color, instead of recognizing by the shape model, is used.

This implementation is described in details in (Martins et al., 2006b) and follows the proposal of (Bianchi & Reali-Costa, 2000), where line segments are traced when a pixel of the given color is reached. The system first defines, from the histogram of the image resulting of background subtraction (Fig. 8), the range of the opponent color and then a sparse pixel sampling is done. Notice that the sparser the pixel sampling is, the faster the algorithm will be, but it may imply in an inefficient recognition because this sampling may not reach all opponent's color blobs.

When a pixel of desired color is reached, horizontal and vertical line segments are traced from this pixel position until a non-desired color pixel is reached. The first line segment to be traced is the horizontal, followed by the vertical segment, which is traced from the mid-

point of the horizontal segment. For the process of tracing horizontal and vertical segments, was given the name "cross-processing" (Martins et al., 2006b).
When the cross-processing is applied to a circle, only one iteration is necessary to determine it's center, but when the cross-processing is applied to other shape models or irregular geometric shapes, more iterations are required to determine the center of this object. The cross-processing runs, recursively, *n* times for each pixel of desired color, where the iteration *m* starts from the mid-point of the vertical segment achieved at the last *m-1* iteration.
The cross-processing is executed many times for the same object, because the sparse pixel sampling may result in many pixels of desired color from the same object blob. Each cross-processing execution for each pixel of desired color generates a possible center point.
Some geometric shapes, as the one illustrated on Fig. 11, can have many possible centers after the complete sparse pixel sampling. To determine the center of the object, a mean between the possible center points is computed. This mean is a spatial mean where each dislocates the center to it's nearby. The more a region of the blob has possible center points, the nearer from this region will be the final center computed. The possible center points considered to compute the mean are those which have an Euclidean distance of less than a threshold defined a priori. The Fig. 11 illustrates how the mean computation approximates the possible center points and converges to the center of the object.
Computing the mean allows that, even on noisy images, the objects can have their centers determined. The Fig. 12-right illustrates a noisy image containing objects with different shapes and their centers determined (red point). Notice that there are objects which the centers computed are not the real centers, but these centers remain into the object blob.
This approach doesn't ensure that the real center of the objects will be achieved, but an estimated center close to the real center, no matter the shape of the object, allowing the recognition of the opponent robots for obstacle avoidance and strategy issues, for instance.

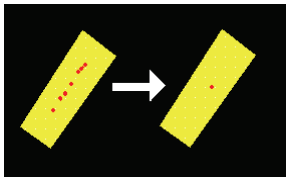


Fig. 11. The cross-processing resulting in the possible center points and the mean computation to determine the center of the object

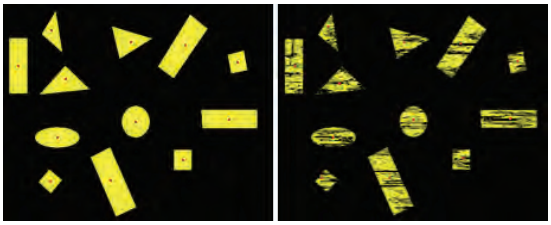


Fig. 12. (left) center detection of objects with different shapes in a standard image (right) center detection on the same image with noise

5. Experiments and Results

The HT and cross-processing implementations were made in C++ and the computer vision library OpenCV (Intel, 2007) was used. The results were obtained in a computer with an Intel Pentium 4 Hyper Threading processor running at 3.2 GHz. The program was executed under Windows XP, configured for priority in real time.

5.1 Execution Times

As the goal was to use the system for an application in real time, the performance evaluation considers as being an acceptable maximum time for the algorithm execution the time interval of an image acquisition. The acquisition systems commonly used in Robot Soccer, as the described in this work, are able to acquire 30 pictures per second. Therefore, the maximum time available for processing is 33 ms.

Table 1 presents the performance results for the HT implemented system, showing the amount of time needed for each processing stage. The values presented are the average of the execution of the system with 30 different images, in two different resolutions (320x240 pixels and 640x480 pixels, both 24 bits NTSC color images). The images used in this test contained six robots and two balls. In each image, the objects are in a different position, spread randomly over the entire field, including the corners. The difference between the sum of the stages times and the total time is small and can be considered rounding error. This results show that the implemented system is capable of recognizing objects not only in real time, but allowing 13 ms for other processes, as strategy and robots control.

As previously mentioned, all adjustable parameters of the system were kept constant for all experiments described in this work. For images with a resolution of 640x480, the radius *r* was set to 8 pixels, while the minimum number of votes was set to 16. For images with a resolution of 320x240, the radius *r* was set to 4 pixels and the minimum number to 10. The thresholds of the Canny filter were defined off-line: the low threshold used was 75 and the high threshold was 150.

Task	*Image size*	
	320x240	640x480
Background subtraction	0,8412 ± 0,001	5,1667 ± 0,003
Color conversion + Canny filter	1,6163 ± 0,002	7,8435 ± 0,005
Hough Space generation	1,4621 ± 0,006	6,3683 ± 0,02
Circle centers determination	0,0334 ± 0,001	0,0267 ± 0,001
Objects recognition	0,0031 ± 0,0001	0,0023 ± 0,0001
Total time	4,0674 ± 0,009	19,4083 ± 0,03

Table 1. Execution Times (in milliseconds)

For the cross-processing the parameter *n*, the number of iterations, was set to 5, with a pixel sampling interval of 10 pixels for lines and columns, clustering the possible centers with a Euclidean distance of less than the threshold set to 70 pixels. The time measured for images with resolution of 320x240 pixels was 3,037 ms with a standard deviation of 0,0354 and 5,892 ms with a standard deviation of 0,0672 for the 640x480 resolution images.

5.2 Light Intensity Variation Robustness

To verify the robustness of the system in respect to light intensity variation, a second experience was performed: again, 6 robots and 2 balls were placed in a random position of the field and then the light intensity was slowly changed from 300 Lux to 1300 Lux. This experiment was repeated 10 times, each time placing the objects in a different position, randomly chosen. To be able to compare the system described in this paper, the same experiment was performed with a color based system that uses threshold and blob coloring techniques to find the robots, calibrated at 1000 Lux (Penharbel et al., 2004).

The result of these experiments is presented in Fig. 13. It can be seen that, while the system proposed in this work is robust to light intensity variation, detecting all objects in all the trials, the color based system only performed well when light intensity was near 1000 Lux. This experiment also indicates that color noise do not affect the system. As it is model-based, different illumination in the same image may change the color of an object, but, nevertheless, will not affect the system capability to compute the position of any object. Finally, the system was tested while controlling the robots during a real game, with the robots moving in all positions of the field, presenting the same performance as in the two experiences described above.

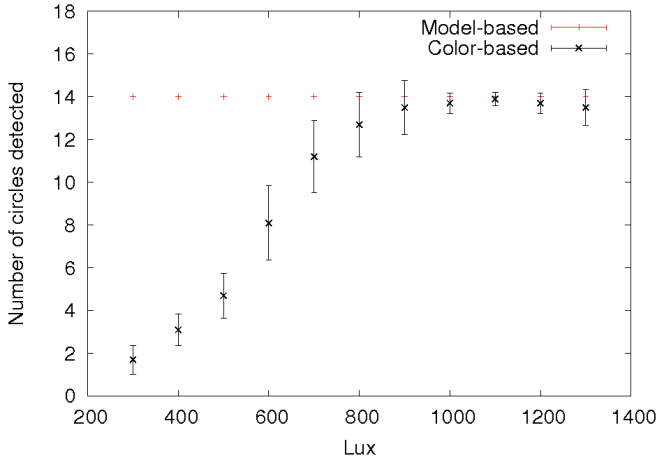


Fig. 13. Number of detected objects versus light intensity variation for the model-based system proposed in this chapter and the color-based system implemented by (Penharbel et al., 2004)

5.3 A Specific Case in which the System May Fail

The system is able to recognize all the robots in any case but one. After the tests, a very specific case was observed, which is very difficult to occur in a Robot Soccer match, where the system may fail. The robots arrangement in the image is one that even the human vision is unable to distinguish the robots with certain. The Fig. 14 illustrates two image sequences of the system processing steps where it may fail and not recognize the robots. Each line shows an example with the captured image, followed by the Canny filter, the Hough Space generated, the circles detected and the objects recognized.

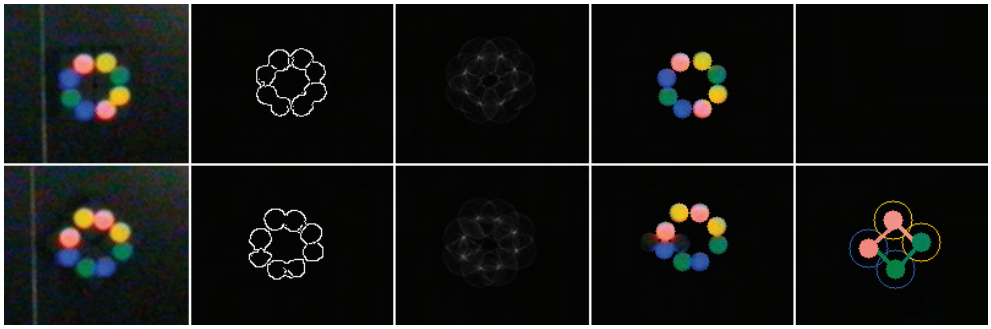


Fig. 14. Specific case where the system may fail, not detecting the robots

On the first image sequence (upper line) of Fig. 14 the robots can't be recognized because every circle of primary color has two child nodes and the constraint satisfaction approach fails. On the Second image sequence (lower line) of Fig. 14 one of the robots is slightly dislocated compared to the first image sequence, resulting in a circle of primary color with only one child node, which results in the recognition of all the robots.

6. Conclusion and Future Work

This chapter described the use of artificial intelligence and computer vision techniques to create a fast and robust real time vision system for a robot soccer team. To increase the system performance, this work proposes a new approach to interpret the space created by the Hough Transform, as well as a fast object recognition method based on constraint satisfaction techniques. The system was implemented entirely in software using an off-the-shelf frame grabber.

Experiments using real time image acquisition allow concluding that the implemented system is robust and tolerant to noises and color variation since it considers just the objects form, and automatically determines the color information, needed only to distinguish the robots among themselves. Robots are well detected in every position of the field, even in the corners or inside the goal area, where light intensity is lower than in the center of the field. The measured execution performance and the tests of object recognition demonstrate that it is possible to use the described system in real time, since it fulfills the demands on performance, precision and robustness existing in a domain as the Robot Soccer.

Future works include the implementation of the control of the camera parameters, such as aperture, zoom, focus, gain and others, in real time. To be able to construct this new part of the system, a camera that allows the control of these parameters through a serial port was bought and is being tested. Finally, distortion lens was not mentioned in this work and, although being small, will be addressed in a future implementation. Another work that is in initial phase of development is the automatic color calibration, in which is intended to use the Hue histogram of colors and clustering algorithms, like K-Means (Forsyth & Ponce, 2002) and is based on recent researches, like (Sridharan & Stone, 2005).

7. References

Bianchi, R. A. C. & Reali-Costa, A. H. (2000). O Sistema de Visão Computacional do Time FUTEPOLI de Futebol de Robôs. *Anais do CBA 2000 – Congresso Brasileiro de Automática*, pp. 2156-2161, Florianópolis, 2000, Sociedade Brasileira de Automática, Brasil.

Bresenham, J. E. (1965). Algorithm for Computer Controlo f a Digital Plotter. *IBM Systems Journal*, Vol. 4, No. 1, 1965, 25-30.

Canny, J. A computational approach to edge detection. (1986). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 6, Nov. 1986, 679-698.

Forsyth, D. A. & Ponce, J. (2002). *Computer Vision: A Modern Approach*. Prentice Hall, New Jersey.

Gönner, C.; Rous, M. & Kraiss, K. (2005). Real-time adaptative colour segmentation for the robocup middle size league. *Lecture Notes in Computer Science*, Vol. 3276, 2005, 402-409.

Grittani, G.; Gallinelli, G. & Ramírez, J. (2000). FutBot: A Vision System for Robotic Soccer. *Lecture Notes in Artificial Intelligence*, Vol. 1952, Nov. 2000, 350-358.

Hough, P. V. C. (1959). Machine analysis of bubble chamber pictures. *International Conference on High Energy Accelerators and Instrumentation*, 1959.

Intel (2007). Intel Computer Vision Library. Online Available at: <http://www.intel.com/technology/computing/opencv/>. Accessed on: 07/30/2007.

Jonker, P.; Caarls, J. & Bokhove, W. (2000). Fast and accurate robot vision for vision based motion. *Lecture Notes in Computer Science*, Vol. 2019, 2000, 149-158.

Martins, M. F.; Tonidandel, F. & Bianchi, R. A. C. (2006a). A Fast Model-Based Vision System for a Robot Soccer Team. *Lecture Notes in Artificial Intelligence*, Vol. 4293, 2006, 704-714.

Martins, M. F.; Tonidandel, F. & Bianchi, R. A. C. (2006b). Reconhecimento de Objetos em Tempo Real para Futebol de Robôs. *Anais do XXVI Congresso da Sociedade Brasileira de Computação*, pp. 173-182, Campo Grande, 2006, Sociedade Brasileira de Computação, Brasil.

Penharbel, E. A.; Destro, R.; Tonidandel, F. & Bianchi, R. A. C. (2004). Filtro de Imagem Baseado em Matriz RGB de Cores-Padrão para Futebol de Robôs. *Anais do XXIV Congresso da Sociedade Brasileira de Computação*, Salvador, 2004, Sociedade Brasileira de Computação, Brasil.

Piccardi, M. (2004). Background Subtraction Techniques: A Review. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics SMC 2004*, The Hague, 2004, Netherlands.

Russell, S. & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.

Sridharan, M. & Stone, P. (2005). Towards Eliminating Manual Color Calibration at RoboCup. In: *RoboCup 2005*, Itsuki Noda, Adam Jacoff, Ansgar Bredenfeld & Yasutake Takahashi, Springer Verlag, Germany.

Weiss, N. & Hildebrand, L. (2004). An exemplary robots soccer vision system. *Proceedings of the CLAWAR/EURON Workshop on Robots in Entertainment, Leisure and Hobby*, Vienna, 2004, Austria.

6

Probabilistic and Statistical Layered Approach for High-Level Decision Making in Soccer Simulation Robotics

Carlos Bustamante and Leonardo Garrido
Tecnológico de Monterrey, Campus Monterrey
México

1. Introduction

In literature, some authors propose to see artificial intelligence as the design of an intelligent agent (Russell & Norvig, 2003). An agent is an entity that perceives its environment, thinks and acts accordingly. An intelligent agent in this sense is one that makes rational decisions. Therefore, some authors like to call them rational agents. The part of artificial intelligence that focuses in the study of rational agents and the way they cooperate, coordinate and negotiate as abstract social entities is called multiagent systems.
A lot of techniques have been proposed in the effort of making rational agents. Every one presents its own advantages and disadvantages and their efficiency varies among different domains. Hence, it seems interesting to try to combine techniques and measure their efficiency when they work together. Such approaches are known as hybrid systems.
Evaluating and testing multiagent systems in real life is very complicated. Many domains are complex, dynamic and uncertain. Diverse testbeds have been created to allow researchers to easily test and compare ideas for extrapolating them to real situations later. One of the most known testbeds for multiagent systems nowadays is the RoboCup competition.
RoboCup initiative is an international project that promotes artificial intelligence, robotics and related areas, through a competition and conferences system with robotic soccer as the base problem. The ultimate goal is "by year 2050, to develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer". The competitions are divided into many leagues, one of them being the 3D simulation league. In RoboCup 3D, the environment is complex, dynamic and noisy.
This chapter focuses on the development of a decision making framework of a RoboCup 3D simulation agent based on a recently explored fuzzy-Bayesian hybrid classifier. Fuzzy theory and Bayesian methods have been used by separate for years and they have presented good results in various domains. A fuzzy-Bayesian approach faces uncertainty in decisions with probabilistic reasoning and learning, and treats variables involved in the process as fuzzy variables, which are expressed linguistically and are computed mathematically. This decision making approach tries to combine the best of statistical data processing with human-like view of attributes related to a problem.

On the other hand, a simulation allows reproducing a real environment approximating physical conditions by means of complex mathematical models. This provides the advantage of being able of changing parameters for representing different environments and to correct mistakes in which many times, in real life, there is no backing out. RoboCup 3D simulation has these characteristics and is a relatively recent RoboCup league with a long way to go.
The efficiency of a fuzzy-Bayesian decision making system for a soccer agent, however, is constrained to the degree of quality of the world model data. This is why the development of the agent is done in a 3-layer fashion. The lowest layer consists of obtaining accurate motion models. The middle layer uses such models and a particle filter to allow a precise self-localization of an agent. With the filtered position of an agent, positions of other objects in the world are easily computed. Finally, the highest layer uses the middle-layer data for making decisions with the fuzzy-Bayesian framework.

2. Soccer

Since 1997, the RoboCup Soccer Simulation 2D has been one of the main contributors to the RoboCup initiative in terms of multiagent learning, coordination, communication and opponent modelling. However, the simulation omits several aspects that affect robots in real situations. The motion of dynamic objects is restricted to two dimensions. Also, the physics are simplistic and don't allow complex behaviours.
In order to fulfill the RoboCup ultimate goal by 2050, the simulation system must treat agents as physical entities with realistic features. After all, the RoboCup goal implies the development of humanoid-like robots and therefore the simulation league must converge with the humanoid league in some point. As more realistic agent models were needed, the 3D simulation league was created. The agents are spheres[1] , but 3D interactions make the environment more complex and more interesting for researchers in artificial intelligence and multiagent theory. The simulator serves as the main platform in the RoboCup Simulated Soccer League, part of the RoboCup competitions, whose main event is celebrated every year.

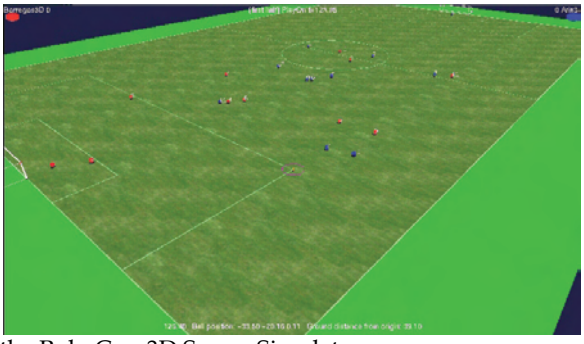


Fig. 1. Monitor for the RoboCup 3D Soccer Simulator

[1] This year the simulator was changed radically and the agents are now humanoids, which makes the RoboCup 3D domain even more complex and interesting.

The RoboCup 3D simulator is a software that provides two elements: a core system called the soccer server and a monitoring tool. The simulator is aimed for three main tasks: allows players to sense and act in their environment, has rules for several situations in a soccer match and applies the laws of physics to recreate a real-like scenario.
The environment of the current soccer simulation[2] is a big box which contains a virtual soccer field. It respects FIFA specification for international real soccer matches. Simulation steps are 0.01 seconds long and agents receive sensations every 20 simulation steps. The global coordinate system of the field is as follows: the x-axis extends over the horizontal line that extends from the left to the right. The y-axis is over the field and is perpendicular to the x-axis. The z-axis points up. The global angle marks 0 degrees in the direction of the x-axis and grows in counter-clockwise direction. A graphical representation of the soccer field coordinate system and the global angle direction is shown in figure 2. The elements in the field are divided in two classes:

- *Static elements* are elements in the environment whose position is fixed.
    - **Landmarks**: reference points for localization.
        - Four posts, two for each of the goals
        - Four flags, which are placed in each corner of the field
    - **Goal**: A rectangular box with width 7.32m, height 2.0m and depth 2.0m. There are two goals, one for each team. They are located over the end line of the field, with two vertical posts touching the line and a crossbar joining those posts. A team scores one point if the ball passes gets inside the goal box.
    - **Field**: A rectangle of width U[64.0m, 75.9m] and length U[100.0m, 110.9m]. It is subdivided in regions such as the left and right half areas, the penalty boxes, the goal boxes and the center circle. The field is contained inside a box of width equal to the field's width plus a border size of 10.0m, length equal to the field's length plus the border size of 10.0m and height 40.0m.
- *Dynamic elements* are elements in the environment which can change their positions over time.
    - **Players**: The main actors in the environment. Players have a spherical shape, with radius 0.22m and mass 75kg. The color depends on the team: one team has blue players and the other has red players.
    - **Ball**: A sphere with radius 0.111m and mass U[0.41kg, 0.45kg], which is much smaller than the mass of the players. The weight of the ball varies from game to game, but it is constant during a game once it has been set.

FIFA rules are implemented in the simulator in different play modes like kick-off, goal kicks, corner kicks, throw-ins and free kicks. If agents try to violate rules entering an area that is prohibited by the current situation, they are teleported to a valid position, which depends of the current situation. In corner kicks, throw-ins and free kicks, the invalid area is inside a circle of 9.15 meters of diameter around the ball. In goal kicks, the invalid area is the

[2] The current simulator version is 0.5.6 (July 2007). It is now more oriented to the new humanoid 3D simulation.

enemy penalty area. In kick-off, the invalid area is the semi-circle of 9.15 meters of diameter in the center of the field plus the opponent's half field.

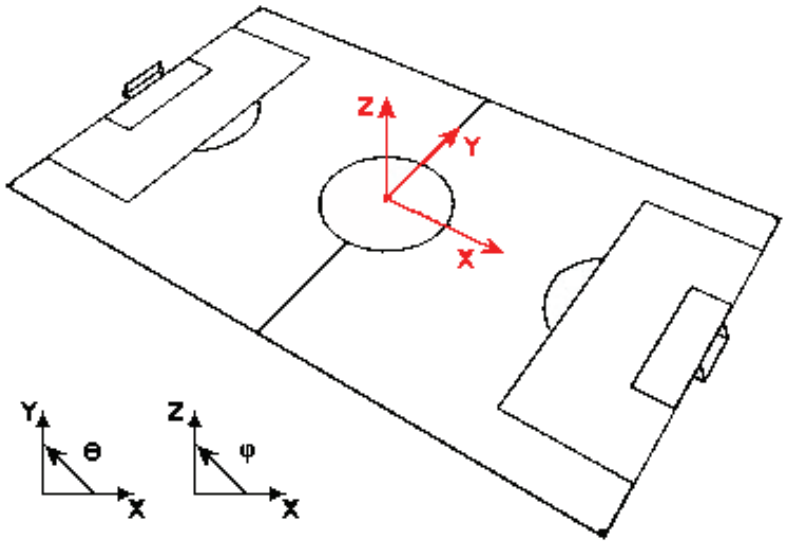


Fig. 2. Global coordinate system in the RoboCup 3D Soccer Simulator

In the RoboCup 3D Soccer Simulation agents are homogeneous, in the sense that their share the same properties and have the same set of perceptors and effectors (except for the goalkeeper that has an extra effector for catching the ball). An agent is an entity that perceives its environment through sensors and acts upon that environment through effectors (Russell & Norvig, 2003). In the 3D simulator, an agent is a client software with such characteristics and connects to the simulation engine through a communication server.
Data packages generated and received by the server are strings in form of s-expressions. Agents that receive these messages must parse these strings in order to analyze the information contained in the package.
The process of interactions between agents and the simulation engine consists of two steps: initialization and life cycle. Agents have a set of effectors and perceptors to act upon their environment. Communication is allowed by using certain effectors and perceptors.
Among the effectors we can find *create*, *init*, *beam*, *drive*, *kick*, *say* and *pantilt*. The goalkeeper has an extra *catch* effector. The agent moves in the environment using its drive effector and interacts with the ball using the kick effector. The set of perceptors includes *vision*, *game state*, *agent state* and *hear*. The say effector and the hear perceptor make communication possible.
To make a more realistic simulation, some effectors and perceptors are affected by white noise. This means that errors are normally distributed with expected values $\mu = 0$ and different standard deviations. The exception is the pantilt perceptor (part of the agent state perceptor) in which values are rounded to the next integer value. Table 1 shows all the error sources and their characteristics.

Uncertainty in perceptors and effectors			
Drive effector	$E_{D_x} \sim N(\mu_{D_x}, \sigma_{D_x})$	$\mu_{D_x} = 0$	$\sigma_{D_x} = 0.005 \cdot D_x$
	$E_{D_y} \sim N(\mu_{D_y}, \sigma_{D_y})$	$\mu_{D_y} = 0$	$\sigma_{D_y} = 0.005 \cdot D_y$
Kick effector	$E_K \sim N(\mu_K, \sigma_K)$	$\mu_K = 0$	$\sigma_K = 0.4$
	$E_{\theta_K} \sim N(\mu_{\theta_K}, \sigma_{\theta_K})$	$\mu_{\theta_K} = 0$	$\sigma_{\theta_K} = 0.02$
	$E_{\phi_K} \sim N(\mu_{\phi_K}, \sigma_{\phi_K})$	$\mu_{\phi_K} = 0$	$\sigma_{\phi_K} \begin{cases} 0.9, & \text{if } \phi_K = 0 \text{ or } \phi_K = 50 \\ 4.5, & \text{otherwise} \end{cases}$
Vision perceptor	$E_r \sim N(\mu_r, \sigma_r)$	$\mu_r = 0$	$\sigma_r = 0.000965 \cdot d$
	$E_\theta \sim N(\mu_\theta, \sigma_\theta)$	$\mu_\theta = 0$	$\sigma_\theta = 0.1225$
	$E_\phi \sim N(\mu_\phi, \sigma_\phi)$	$\mu_\phi = 0$	$\sigma_\phi = 0.1480$
Agent State perceptor	$\alpha_{pan} = \lceil \alpha_{pan} \rceil$		$\alpha_{tilt} = \lceil \alpha_{tilt} \rceil$

Table 1. Sources of uncertainty in the RoboCup 3D simulation environment.

3. Motion Models

Obtaining the motions models in RoboCup 3D is the first and most essential task to accomplish before working on high level design. Knowing the motion models means knowing what is going on behind the scenes in the simulation. Not only we can predict the results of dynamic object actions, but we also can explain some behavior and thus construct better high level solutions.

The models were derived from a formal mathematical analysis. For obtaining accurate and feasible parameters for the motion models, the power of statistical analysis and curve fitting tools was exploded. The parameters were obtained under ideal conditions, i.e. noise produced by the simulator in effectors and perceptors was eliminated. The methodologies and results of this chapter were published in (Bustamante et al., 2007).

3.1 Definition

A motion model of a dynamic object is described formally as a function from certain duple (p_t, v_t) in time t to a duple (p_{t+i}, v_{t+i}) in time $t + i$, where p is the position vector, v is the velocity vector and $i > 0$, that is:

$$M : (p_t, v_t) \rightarrow (p_{t+i}, v_{t+i}) \tag{1}$$

3.2 Agent Motion Model

The movement of the agent is affected by the drive force (applied to its drive effector) and by the air friction. The drive force vector $\vec{F_d}$ is defined as

$$\vec{F_d} = \langle F_{d_x}, F_{d_y} \rangle \tag{2}$$

The force vector does not have a z-axis component because spherical agents can't jump. The air drag force is defined as (Marion & Thornton, 2003)

$$F_{drag} = -\xi v \tag{3}$$

The constant ξ represents the coefficient that imposes an air drag force to a body and v is the body's velocity. The equations that model the forces over each axis are

$$\begin{aligned} \sum F_x &= F_d cos(\theta) - \xi_A v_x = m_A a_x \\ \sum F_y &= F_d sin(\theta) - \xi_A v_y = m_A a_y \end{aligned} \quad (4)$$

Here m_A is the mass of the agent, a is the acceleration, F_d is the drive force (in newtons), ξ_A is the air drag coefficient for the agent and θ is the global horizontal angle in the x-y plane. Also let $F_{d_x} = F_d cos(\theta)$ and $F_{d_y} = F_d sin(\theta)$.
Let $D \in [0, 100]$ be the drive force percentage, i.e. the drive force command sent to the Drive Effector. The relation between D and F_d is given by

$$F_d = \frac{D}{100} F_{d_{max}} \quad (5)$$

As the equations that model the forces for both axis are similar, it is enough to analyze x-axis and generalize for y-axis later. The differential equation that models the movement on the x-axis is expressed as a differential equation from (4)

$$m_A \frac{dv}{dt} + \xi_A v = F_{d_x} \quad (6)$$

3.2.1 Agent Speed Model

Solving the differential equation (6) gives

$$v(t) = \frac{F_{d_x}}{\xi_A} + (v_i - \frac{F_{d_x}}{\xi_A}) e^{-\frac{\xi_A}{m_A} t} \quad (7)$$

Finally for simplicity some constant terms are defined like the terminal speed of the agent

$$v_{A_T} = \frac{F_{d_x}}{\xi_A} \quad (8)$$

which is the maximal speed that the agent can reach when the drag force equals the drive force. Also, let's define a time constant

$$\tau_A = \frac{m_A}{\xi_A} \quad (9)$$

Finally, the *agent's speed model* is expressed as

$$v(t) = v_{A_T} + (v_i - v_{A_T}) e^{-t/\tau_A} \quad (10)$$

3.2.2 Agent Position Model

Once having the speed model, the *agent's position model* is determined relatively easy integrating equation (10) from 0 to t

$$p(t) = p_i + v_{A_T}t + \tau_A(v_i - v_{A_T})(1 - e^{-t/\tau_A}) \tag{11}$$

where p_i is the integration constant and represents the initial position.

3.3 Ball Motion Model

Unlike the agent, the movement of the ball can be separated in two different phases:

1. In the first phase, a kick force is applied to the ball for 10 simulation steps. Thus the ball is affected by the kick force and by the air drag force in X and Y, and is also affected by gravity in Z.
2. In the second phase, the ball decelerates and is affected just by the air drag force in X and Y, and additionally by gravity in Z.

3.3.1 First Phase of Ball Motion

In the first phase, the ball behaves like an agent with constant force (kick force). The force vector is defined as

$$\vec{F_k} = \langle F_{k_x}, F_{k_y}, F_{k_z} \rangle \tag{12}$$

Let θ_k be the global horizontal angle in the x-y plane between the agent and the ball and ϕ_k the elevation angle sent to the kick effector. The equations that model the forces over each axis are

$$\begin{aligned} \sum F_x &= F_k cos(\phi_k) cos(\theta_k) - \xi_B v_x = m_B a_x \\ \sum F_y &= F_k cos(\phi_k) sin(\theta_k) - \xi_B v_y = m_B a_y \\ \sum F_z &= F_k sin(\phi_k) - m_B g - \xi_B v_z = m_B a_z \end{aligned} \tag{13}$$

Here m_B is the mass of the ball, a is the acceleration, F_k is the kick force (in newtons) and ξ_B is the air drag coefficient for the ball. Also let $F_{k_x} = F_k cos(\phi_k) cos(\theta_k)$, $F_{k_y} = F_k cos(\phi_k) sin(\theta_k)$ and $F_{k_z} = F_k sin(\phi_k)$.

Let $K \in (0, 100]$ be the kick force percentage, i.e. the power command sent to the Kick Effector. The relation between K and F_k is given by

$$F_k = \frac{K}{100} F_{k_{max}} \tag{14}$$

As with the agent, we can generalize one equation for both X and Y axis, but we have to define a different equation for z-axis. The differential equation that models the movement on X and Y is expressed as

$$m_B \frac{dv}{dt} + \xi_B v = F_k \tag{15}$$

The differential equation that models the movement on Z is expressed as

$$m_B \frac{dv_z}{dt} + \xi_B v_z = F_{k_z} - m_B g \tag{16}$$

3.3.1.1 Ball Model for the First Phase in X-Y

Notice that equation (15) is the same of that of the agent (6). Then, we only summarize the finals equations:

$$v(t) = v_{B_T} + (v_i - v_{B_T}) e^{-t/\tau_B} \tag{17}$$

$$p(t) = p_i + v_{B_T} t + \tau_B (v_i - v_{B_T})(1 - e^{-t/\tau_B}) \tag{18}$$

where v_{B_T} is the terminal speed of the ball (i.e. the maximal speed that the ball could reach if the kick force was applied for a long time).

3.3.1.2 Ball Model for the First Phase in Z

The solution to the differential equation (16) for Z is expressed as

$$v_z(t) = \frac{F_{k_z} - m_B g}{\xi_B} + A e^{-\frac{\xi_B}{m_B} t} \tag{19}$$

This is very similar to equation (7). We have to define the terminal speed of the ball in Z as

$$v_{Bz_T} = \frac{F_{k_z} - m_B g}{\xi_B} \tag{20}$$

Finally we have

$$v_z(t) = v_{Bz_T}(1 - e^{-t/\tau_B}) \tag{21}$$

$$p_z(t) = p_{z_i} + v_{Bz_T} t - \tau_B v_{Bz_T}(1 - e^{-t/\tau_B}) \tag{22}$$

3.3.2 Second Phase of Ball Motion

In the second phase, the ball decelerates until it stops moving in X and Y, and it bounces until it stops moving in Z. The equations that model the forces over each axis are

$$\begin{aligned} \sum F_x &= -\xi_B v_x = m_B a_x \\ \sum F_y &= -\xi_B v_y = m_B a_y \\ \sum F_z &= -m_B g - \xi_B v_z = m_B a_z \end{aligned} \tag{23}$$

The differential equation that models the movement on X and Y is expressed as

$$m_B \frac{dv}{dt} + \xi_B v = 0 \tag{24}$$

The differential equation that models the movement on Z is expressed as

$$m_B \frac{dv_z}{dt} + \xi_B v_z = -m_B g \tag{25}$$

3.3.2.1 Ball Model for the Second Phase in X-Y

Solving differential equation (24) gives

$$v(t) = v_{i_{phase2}} e^{-\frac{\xi_B}{m_B} t} \tag{26}$$

The amplitude $V_{i_{phase2}}$ represents the initial speed of the second phase, which must be equal to the final speed of the first phase evaluated in 10 simulation steps of 0.01 seconds each). Formally,

$$v_{i_{phase2}} = v_{B_T} + (v_{i_{phase1}} - v_{B_T}) e^{-(10)(0.01)/\tau_B} \tag{27}$$

Using equation (26) the speed is given by

$$v(t) = v_{i_{phase2}} e^{-t/\tau_B} \tag{28}$$

and the position by

$$p(t) = p_i + \tau_B v_{i_{phase2}} (1 - e^{-t/\tau_B}) \tag{29}$$

3.3.2.2 Ball Model for the Second Phase in Z

Solving differential equation (25) gives

$$v_z(t) = A e^{-\frac{\xi_B}{m_B} t} - \frac{m_B}{\xi_B} g \tag{30}$$

The constant A can be calculated using the initial condition $v_z(0) = v_{z_i}$ as

$$A = v_{z_i} + \frac{m_B}{\xi_B} g \tag{31}$$

Substituting this value in equation (30), the speed model is given by

$$v_z(t) = (v_{z_i} + \frac{m_B}{\xi_B} g) e^{-\frac{\xi_B}{m_B} t} - \frac{m_B}{\xi_B} g \tag{32}$$

which in terms of constants is

$$v_z(t) = (v_{z_i} + g\tau_B) e^{-t/\tau_B} - g\tau_B \tag{33}$$

and the position model is given by

$$p_z(t) = p_{z_i} + \tau_B (v_{z_i} + g\tau_B)(1 - e^{-t/\tau_B}) - g\tau_B t \tag{34}$$

When the ball is at rest, p_{z_i} is equal to the radius of the ball.

3.5 Finding the Values of the Coefficients

The next step is to evaluate the coefficients needed by the model. We already know m_A and m_B, so we must look specifically for $F_{d_{max}}$, $F_{k_{max}}$, ξ_A and ξ_B in order to have a complete description of the equations.

Two scenarios were defined for obtaining representative data from the simulation. In the first scenario, an agent is placed in the center of the soccer field and runs over the x-axis with maximum acceleration towards the opponent's goal. In the second scenario, the ball is placed at the center of the field and is kicked by an agent with maximum acceleration towards the opponent's goal.

Tracking the data of the scenarios via the monitor's port, a set of pairs (t, x) were obtained where t is time and x is position, which describes the movement of the agent and the ball over the x-axis. The values of the coefficients were computed with Matlab© and the Curve Fitting Tool, using the set of pairs (t, x) and the motion models, giving the following results

$$F_{d_{max}} \approx 308.0904N \tag{35}$$

$$F_{k_{max}} \approx 57.304638N \tag{36}$$

$$\xi_A \approx 190.65 \frac{kg}{s} \tag{37}$$

$$\xi_B \approx 0.214785 \frac{kg}{s} \tag{38}$$

We can get the medium viscosity with the aid of the Stokes' equation (Marion & Thornton, 2003)

$$\xi = 6\pi r\eta$$

where r is the radius, and η the viscosity coefficient. This quantity must be calculated with ξ_B which is a drag force caused only by the fluid, opposite to ξ_A that represents a drag force caused by the fluid and the system motion. Using the radius of the ball $r = 0.111$ we have

$$\eta \approx 0.1026 \frac{kg}{m \cdot s} \tag{39}$$

We can infer that the simulated medium is not air as the value of η is bigger than the air viscosity which is approximately $1.74x10^{-5}\frac{kg}{m \cdot s}$.

3.6 Practical Applications

The physics model described so far can be used to implement higher level behaviors like soccer skills. In the next sections we describe two of such skills.

3.6.1 Goto

Using the *Goto skill* the agent is capable of moving to any <x, y> coordinate on the soccer field. The movement of the agent consists of three steps: 1) acceleration, 2) constant speed and 3) deceleration. In fact, the most important is step 3 because in the first two steps the agent must apply the maximal force, but in the last step the agent must decide at which moment stops. For breaking, the agent applies a drive force vector of magnitude zero and makes use of the drag force to stop. It calculates the distance D_{stop} that it needs to stop moving when drive force becomes zero. This distance is compared with the distance that the agent needs to cross to reach its destination D_{dest}. If $D_{dest} > D_{stop}$, the agent keeps applying a drive force, otherwise the agent stops applying the drive force. D_{stop} can be calculated as

$$D_{stop} = v_i\tau_A(1 - e^{-t/\tau_A}) = v_i\tau_A(1 - e^{-\infty/\tau_A}) = v_i\tau_A \tag{40}$$

3.6.2 Dribbling

The *Dribbling skill* provides the agent with the ability to move from one place to another without losing the possession of the ball. For accomplishing this task, the agent needs to run in the direction of the ball's velocity vector and kick the ball with the exact force that allows the agent to kick again in a near future without loosing possession and without colliding with the ball.

This is a difficult skill that few teams have implemented efficiently. One of the few teams that have implemented the dribble skill efficiently is SEU3D (Xu et al., 2006), but no precise explanation is given in their team description paper about their method. Our idea is that the agent can decide at which distance d it desires to kick the ball after kicking it for the first time. Then the agent can use this distance to find the needed kick force. In fact, d is the

displacement of the agent between kicks, but is also the displacement of the ball. Using equation (11) with the assumption that the agent has reached its terminal speed and equation (29) we have

$$d = v_{A_T} t = \tau_B v_{i_{phase2}} (1 - e^{-t/\tau_B})$$
$$t = d / v_{A_T}$$
$$d = \tau_B v_{i_{phase2}} (1 - e^{-d/(v_{A_T}\tau_B)})$$
$$v_{i_{phase2}} = \frac{d}{\tau_B (1 - e^{-d/(v_{A_T}\tau_B)})}$$

We can find the value of $v_{i_{phase2}}$ with equation (27). But this equation needs a simplification. In fact, in that equation $v_{i_{phase1}}$ is much smaller that v_{B_T} because the agent can only apply a very small speed to the ball due to the restriction that the kick force is applied only for a little period of time. Hence we have

$$\frac{d}{\tau_B (1 - e^{-d/(v_{A_T}\tau_B)})} = v_{B_T} (1 - e^{-0.1/\tau_B})$$

but also v_{B_T} equals F_k / ξ_B. Then, we finally get the kick force that the agent needs apply to the ball for an efficient dribble skill as

$$F_k = \frac{\xi_B d}{\tau_B (1 - e^{-d/(v_{A_T}\tau_B)})(1 - e^{-0.1/\tau_B})} \tag{41}$$

3.7 Experiments

For evaluating the models of the agent a scenario was defined were an agent is placed in the center of the soccer field and runs with maximum acceleration towards the opponent's goal. For the ball, a scenario was defined where the ball is placed at the center of the field and is kicked by the agent with maximum acceleration towards the opponent's goal. The error of the models against the noiseless real data is computed to make an objective evaluation of such models. Results of comparison are shown in table 2. Also, we present here two graphs that show the efficiency of our physics model when it is applied to a) GoTo Skill and b) Dribbling skill.Mean and standard deviation were computed of the absolute errors between the values thrown by the models and the expected real values, thus

$$|e_{position}| = |Position_{Model} - Position_{Real}| \tag{42}$$

	$\lvert e_{position} \rvert$	
	Mean	Standard Deviation
Agent Motion Model	0.001913	0.000731
Ball Motion Model (X-Y)	0.005004	0.010815
Ball Motion Model (Z)	0.418519	0.160649

Table 2. Mean and standard deviation of the absolute error (in meters) between data thrown by the motion models and real information thrown by the simulator in debug mode

3.7.1 Goto

In this experiment an agent uses the *Goto skill* to move 10 meters away from its initial position. Figure 3 shows the real and estimated values of the distance between the agent position and the destination. We can notice that 1) our physics model is so accurate that both curves almost superpose and 2) the efficiency of the GoTo skill is so good that the agent reach its destination without oscillating in the final position.

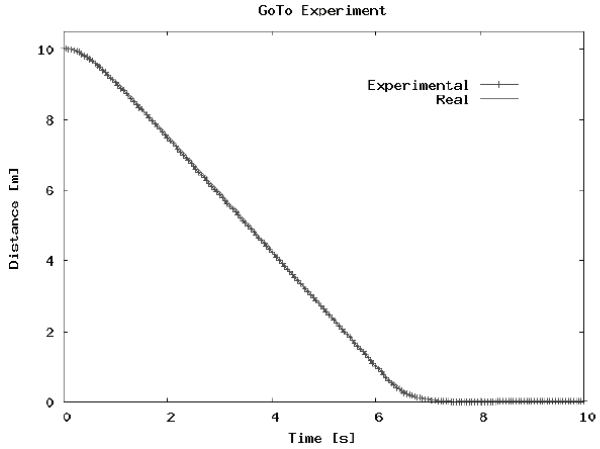


Fig. 3. GoTo Experiment. The real and the estimated positions almost overlap which indicates a good accuracy of the physics models

3.7.2 Dribbling

Figure 4 shows the speed of the agent and the ball versus time. The agent runs towards the ball in the direction of the ball's velocity vector. We can notice that 1) The first kick is weaker than the others because the agent has not reached its maximal speed and 2) The agent never decelerates.

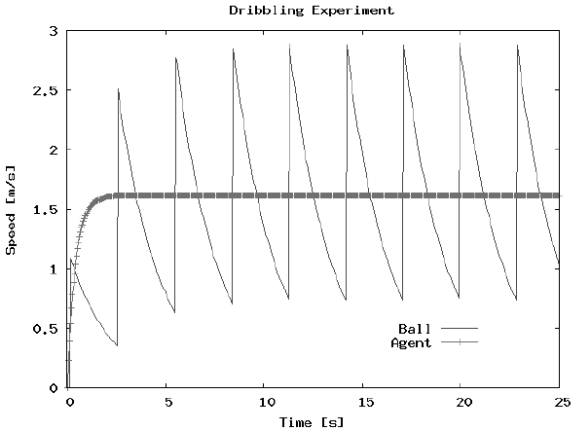


Fig. 4. Dribbling Experiment. The agent does not decelerate which indicates an efficient dribble as the agent never collides with the ball

4. Probabilistic Localization

Localization refers to the problem of determining the pose of an agent from sensor data (Fox, 1998). The pose of an agent represents the location and orientation of a robot relative to a global coordinate frame (Thrun et al., 2005).

The localization problem has been claimed as "the most fundamental problem to providing a mobile robot with autonomous capabilities" (Cox, 1991). It is a fundamental problem because if an agent ignores where it is, it is not feasible to decide what action to execute. Unfortunately, in most situations, an agent cannot sense the pose directly, i.e. it is not equipped with a noise-free sensor for measuring its position. So the agent has to compute its pose based on relative and absolute measurements of reference points. The set of all absolute reference points is the so-called map of the environment. The map contains the reference points in global coordinates.

In RoboCup 3D, the agent's pose is a tuple $[x, y, \theta, \phi]^T$, where $[x, y]^T$ is the 2D position of the robot in the global coordinate frame and $[\theta, \phi]^T$ is the orientation of the agent's pan-tilt angle, respectively. The aforementioned reference points are the corner flags and the goal posts. In robotics, reference points receive the name of landmarks to indicate that they are used for robot navigation. In the soccer simulation, a RoboCup 3D soccer agent receives the range (distance) and bearing (angles) to each visible landmark, along with a signature that identifies it. Hence, there is no uncertainty about the identity of each flag, but the range and bearing measurements are affected by Gaussian noise. In most real situations, a robot does not directly sense the characteristics of the landmarks. Instead, it has to extract or infer important features from data.

When an agent receives relative information of the reference points by means of its vision perceptor, it has to guess its location as accurately as possible. This is even harder because the intrinsic uncertainty in perceptors and effectors. To increase accuracy, an agent must filter the noisy data to get a reliable and precise pose estimate.

4.1 Classification of Localization Problems

Thrun divides the localization problem into three dimensions (Thrun et al., 2005), depending on the nature of the environment and the previous knowledge:

- *Local/Global Localization*: Characterized by the initial knowledge of the agent. It has three categories:
  - *Position tracking*: Assuming that the initial agent's pose is known, this method uses the motion model to track the position of the agent considering a small-effect noise (usually approximated by a unimodal distribution like a Gaussian). As the uncertainty is around the agent's true pose, the problem is called *local*.
  - *Global localization*: In this case, the initial pose of the agent is unknown, i.e. when the agent is placed in the environment it lacks information about where it is. It is a harder problem than position tracking.
  - *Kidnapped robot problem*: It has the same characteristics than global localization, but with more difficulties. It assumes that the agent can be teleported to other location during its operation. It is hard because the agent believes that it knows where it is while in reality it does not. In global localization, the agent knows for sure that it does not know where it is. Usually, wrong beliefs about the state of the world are worst than ignorance about the world itself.
- *Static versus Dynamic Environments*: A static environment is that in which the agent is the only dynamic object. A dynamic environment is that in which many objects change their poses over time. Clearly, a dynamic environment presents much more difficulties than a static environment.
- *Passive versus Active Approaches*: Passive localization refers to the case when a module external to the agent observes the agent's operation over time. An active localization approach is that in which the agent has a control module that minimizes its localization error.

In RoboCup 3D, the localization problem consists of position tracking with possible kidnapping, in a dynamic environment under an active approach. It is considered as position tracking because the agent usually knows its initial position (and the initial position of all its teammates) due to previously defined formations and roles. The kidnapped problem emerges when an agent violates some rule of the soccer simulation, like trying to access a restricted area in a free kick situation, in which the agent is teleported to an allowed sector of the field. The environment is dynamic because there are many moving objects in the environment in addition to the agent that have their own dynamics (the ball, teammates and opponents). Finally, the approach used is active because the agent has its own localization module.

Several approaches have been proposed in literature for the localization problem, trying to reduce the effect of noise and increase the accuracy of the computed position as more information is obtained over time. The two classical approaches in literature are the Kalman Filter and the Monte Carlo localization. The former uses continuous Bayes' filters and the latter uses particle filter principles.

4.2 Markov Localization

Markov localization (Fox, 1998) is a special case of probabilistic state estimation applied to mobile robot localization. It represents the straightforward application of Bayes' filters to localization (Thrun et al., 2005). It addresses the problem of pose estimation from sensor data given an initial hypothesis of a static environment, and uses Bayes' rule and convolution to update the belief whenever the robot senses or moves. As the environment is static, Markov assumption holds: the agent's location is the only state which affects sensor readings.

Instead of maintaining a single hypothesis of the agent's pose, Markov localization maintains a probability distribution over the space of all such hypothesis (Fox et al., 1999). Probabilities are used as weights of these different hypotheses in a formal mathematical way.

A Markov localization method requires both an observation model and a motion model (Röfer et al., 2005). The observation model defines the probability for sensing certain measurements at certain locations. The motion model expresses the probability for certain actions to move the agent to certain relative poses.

Markov localization is a direct application of state estimation within the framework of "Partially Observable Markov Decision Processes" (POMDP). POMDP use a state estimator for estimating the state of the world based on sensor data and on the actions taken by the agent. Markov localization is a special case of such a state estimator: the agent is a mobile robot and the state of the world is the position of the robot within its environment (Fox, 1998).

Algorithm 1 shows the Markov localization method. First of all, a prediction is done using action u_t (line 1). Then the resulting belief is updated using percept z_t (line 2). Finally, the belief is normalized (line 3).

Require: Belief $bel(l_{t-1})$, action u_t and percept z_t with $t \geq 1$
Ensure: Belief $bel(l_t)$

// Prediction (Action Update)
1: $\overline{bel}(l_t) \Leftarrow \int_{l'} p(l_t|L_{t-1} = l', u_t) bel(l_{t-1}) dl_{t-1}$

// Correction (Measurement Update)
2: $bel(l_t) \Leftarrow p(z_t|l_t)\overline{bel}(l_t)$

// Normalize
3: $bel(l_t) \Leftarrow \frac{bel(l_t)}{p(z_t|L_t)}$

Algorithm 1. Markov localization

4.3 Monte Carlo Localization

Monte Carlo localization (MCL) is a type of Markov localization in which the probability distribution over the space of all pose hypothesis of the agent is modeled with a set of particles (Thrun et al., 2005). Monte Carlo Localization is based on particle filters (a.k.a. Sequential Monte Carlo methods), which are approximate Bayes' filters that use random samples for posterior estimation (Thrun et al., 2000) . Each particle represents the hypothesis of an agent having a certain pose. Such particles consist of a robot pose and a certain importance weight. Like in Sampling Importance Resampling (SIR) filters (Skare et al., 2003), the importance weights are approximations to the relative posterior probabilities (or

densities) of the particles. Also, Monte Carlo importance sampling resembles genetic algorithms (Higuchi, 1997). MCL has become very popular among localization algorithms in the last years, mainly because it is easy to implement, it can process raw sensor measurements, it is non-parametric and it can represent non-linear, non-Gaussian, multi-modal probability distributions (Ronghua & Bingrong, 2004).
Formally, the MCL algorithm approximates the belief state $bel(l_t)$ by a set of N weighted samples (which represent a discrete probability density function) in the following way

$$bel(l_t) \approx \langle l_t^{[i]}, w_t^{[i]} \rangle_{i=1:N} \tag{43}$$

The variable $l_t^{[i]}$ is a sample of the random variable L in time t. The variable $w_t^{[i]}$ represents importance weights. Ideally, each particle should be proportional to the posterior belief $bel(l_t)$ such that

$$l_t^{[i]} \; p(l_t | z_{1:t}, u_{1:t}), \; 1 \geq i \leq N \tag{44}$$

These particles, together with the current control u_t, are given as input to the motion model of the agent. Then each particle is weighted using the measurement model. After this, we have an updated set of particles. Finally, the most crucial step in MCL is executed: resampling, in which N new particles are selected with replacement from the updated set, where the probability of selecting each sample is proportional to its weight. After the resampling step, the particles approximate the true posterior belief. The resulting particle set has many duplicates due to selection with replacement, which causes particles with higher weights to appear more in the final set than particles with lower weights. Algorithm 2 shows the Monte Carlo localization method.

Require: Last particle set S_{t-1}, action u_t and percept z_t with $t \geq 1$
Ensure: New particle set S_t

$\bar{S}_t \Leftarrow \emptyset$
$S_t \Leftarrow \emptyset$
$\beta \Leftarrow 0$
// Create temporal set for approximating $\overline{bel}(l_t)$
for $i = 1$ to N **do**
  $l_t^{[i]} \Leftarrow \text{SAMPLEMOTIONMODEL}(u_t, l_{t-1}^{[i]})$
  $w_t^{[i]} \Leftarrow \text{MEASUREMENTMODEL}(z_t, l_t^{[i]})$
  $\beta \Leftarrow \beta + w_t^{[i]}$
  $\bar{S}_t \Leftarrow \bar{S}_t + \langle l_t^{[i]}, w_t^{[i]} \rangle$
end for
// Normalize the importance weights
for $i = 1$ to N **do**
  $w_t^{[i]} \Leftarrow \frac{w_t^{[i]}}{\beta}$
end for
// Resampling (Correction Stage)
for $i = 1$ to N **do**
  select $j \in \bar{S}_t$ with probability $\propto w_t^{[j]}$
  $S_t \Leftarrow S_t + \langle l_t^{[j]}, \frac{1}{N} \rangle$
end for

Algorithm 2. Monte Carlo localization

4.4 Solving the Kidnapped Robot Problem

The classical MCL algorithm presented in the above sections cannot recover from robot kidnapping or global localization failures. As time goes on, particles converge to a single pose and the algorithm is not able to recover if such a pose is invalid. The problem is specially important when the particle set size is small ($N \approx 50$). This problem can be solved by *injection of random particles*. Assume that the agent may be kidnapped at any time t with small probability. Then add a fraction of random samples in the motion model for attacking the problem and adding robustness at the same time.

The number of particles injected at each iteration changes over time. We can use the measurement probability for this purpose. Thrun (Thrun et al., 2005) proposed a modification to the MCL algorithm called *Augmented Monte Carlo Localization* (AMCL), which is shown in Algorithm 3.

Require: Last particle set S_{t-1}, action u_t and percept z_t with $t \geq 1$
Ensure: New particle set S_t

$\hat{S}_t \Leftarrow \emptyset$
$S_t \Leftarrow \emptyset$
$\beta \Leftarrow 0$
// Create temporal set for approximating $\overline{bel}(l_t)$
for $i = 1$ to N **do**
  $l_t^{[i]} \Leftarrow \text{SAMPLEMOTIONMODEL}(u_t, l_{t-1}^{[i]})$
  $w_t^{[i]} \Leftarrow \text{MEASUREMENTMODEL}(z_t, l_t^{[i]})$
  $\beta \Leftarrow \beta + w_t^{[i]}$
  $\hat{S}_t \Leftarrow \langle l_t^{[i]}, w_t^{[i]} \rangle$
  $w_{avg} \Leftarrow w_{avg} + N^{-1} w_t^{[i]}$
end for
// Update the short-term and long-term averages
$w_{slow} \Leftarrow w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
$w_{fast} \Leftarrow w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
// Normalize the importance weights
for $i = 1$ to N **do**
  $w_t^{[i]} \Leftarrow \frac{w_t^{[i]}}{\beta}$
end for
// Resampling (Correction Stage)
for $i = 1$ to N **do**
  $U \Leftarrow U((0, 1])$
  if $U < max\{0, 1 - \frac{w_{fast}}{w_{slow}}\}$ **then**
    $S_t \Leftarrow S_t +$ SAMPLELANDMARKMODEL()
  else
    select $j \in \hat{S}_t$ with probability $\propto w_t^{[j]}$
    $S_t \Leftarrow S_t + \langle l_t^{[j]}, N^{-1} \rangle$
  end if
end for

Algorithm 3. Augmented Monte Carlo localization

Algorithm 3 injects random particles to counterattack the problem of global localization.
The particles could be drawn according to a uniform distribution, but a better idea is to generate particles from the measurement distribution which is feasible due to the fact that the sensor model in our domain is based on landmarks. A new strategy is suggested: to fusion the information of every sensed landmark using a Kalman Filter, thus computing a more accurate pose from the measurements. With this strategy we aim to generate better particles for the injection of particles phase of AMCL. We call this approach KFSF-AMCL (Kalman Filter Sensor Fusion for AMCL). A Kalman Filter is a recursive filter which estimates the state of a system from incomplete and noisy measurements. In position tracking, the Kalman Filter has similar steps to the particle filter: it updates the state using a motion model and corrects it using the measurement model. When used for sensor fusion, only the measurement update is needed which is stated in the following equations

$$\begin{aligned} \boldsymbol{K} &= \boldsymbol{P}\boldsymbol{H}^{\boldsymbol{T}}(\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^{\boldsymbol{T}} + \boldsymbol{R})^{-1} \\ \hat{\boldsymbol{x}} &= \hat{\boldsymbol{x}} + \boldsymbol{K}(\boldsymbol{z} - \boldsymbol{H}\hat{\boldsymbol{x}}) \\ \boldsymbol{P} &= (\boldsymbol{I} - \boldsymbol{K}\boldsymbol{H})\boldsymbol{P} \end{aligned} \tag{45}$$

Here, $\hat{\boldsymbol{x}} \in \Re^n$ is the current state or pose estimate, $\boldsymbol{z} \in \Re^m$ is the vector of measurements, $\boldsymbol{K} \in \Re^{n \times m}$ is Kalman gain which minimizes the a posteriori error covariance, $\boldsymbol{P} \in \Re^{n \times n}$ is the a posteriori estimate error covariance, $\boldsymbol{R} \in \Re^{m \times m}$ is the measurement error covariance and $\boldsymbol{H} \in \Re^{m \times n}$ relates the process state to the measurement.

4.5 Experiments

A experiment was carried out to probe the performance of AMCL algorithm in the localization of a RoboCup 3D agent. Systematic resampling is used in all experiments because implementation of particle filters in robotics use this kind of mechanism very often (Thrun et al., 2005). Furthermore, the size of the particles set was fixed to 100, the vision is restricted (the official ranges are 180 degrees for the horizontal plane and 90 degrees for latitudal angle) and the AMCL parameters for injection of random particles were fixed to $\alpha_{slow} = 0.1$ and $\alpha_{fast} = 0.99$.
The experiment is aimed to prove the accuracy of the AMCL algorithm with different resampling strategies. A graphical explanation of the experiment is shown in figure 5.

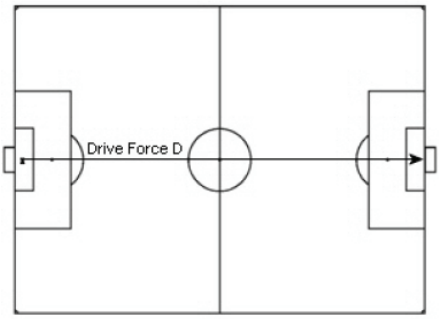


Fig. 5. Scenario for localization experiment 1

Table 3 shows the results obtained with simple MCL. The error is the x-axis is relatively big, given that the minimal kick distance is 0.07 meters between the agent and the ball. The error appears because the noise in effectors and perceptors is accumulated over time and the algorithm is unable to recover from errors due to low variance in particles.

	Error in x-axis
Minimum	0.000174046
Maximum	3.43221
Average	1.18437

Table 3. Accuracy of simple MCL

In table 4 we can see the comparison among different configurations of AMCL. The worst strategy is obviously the random landmark heuristic with an average error of 0.142 in the axis were the agent is moving (x) and 0.063 in the other axis (y). The maximum error is 0.277 which is relatively high considering that the minimum kick distance to the ball is 0.07 meters. Following the random strategy we have the closest landmark heuristic with an average error of 0.079 meters and the average of landmarks heuristic with 0.045 meters. AMCL algorithm with Kalman Filter Sensor Fusion gives the best results with an average error of 0.033 meters, a maximum error of 0.084 meters and a standard deviation of 0.022 meters.

	Absolute Error		
Implementation	maximum	mean	standard deviation
Random	0.277	0.142	0.064
Closest	0.155	0.079	0.050
Average	0.090	0.045	0.022
KFSF	0.084	0.033	0.022

Table 4. Accuracy of AMCL with four different implementations of SampleLandmarkModel

5. Probabilistic Decision Making

RoboCup simulation is an excellent test-bed for machine learning algorithms. It presents a multiagent cooperative and adversarial scenario in a partially observable, episodic, continuous and non-deterministic noisy environment.
Given such uncertainty, classical logic-based approaches fail to achieve a high performance. Thus, a probabilistic method is ideal for dealing with this kind of environment.
The simplest probabilistic approach is the Naive Bayesian classification (Langley et al., 1992) which has proven to be successful in many applications (Lewis, 1998) in spite of the not always fulfilled conditional independence assumption of the attributes given the class. If we wish to use this classifier in the RoboCup simulation domain, we confront two main issues.
First, the classical Naive Bayes classifier assumes that the attributes are discrete, but in RoboCup simulation the attributes are in the range of real numbers and thus are continuous. Second, the classifier must lead to a fast decision process because the soccer simulator demands almost real-time decisions with low thinking times for the sense-think-act cycle of the agents.

In literature, continuous attributes are handled using conditional Gaussian distributions for each attribute's likelihood given the class. Other approach is to discretize by crisp partitioning the domain of the attributes, but this can lead to loss of information.
Instead of discretizing, the issues are overcome using a fuzzy extension namely Fuzzy Naive Bayesian classifier in the following way: the continuous attributes are fuzzified and combined with probabilities of the naive Bayes model in a straight easy way. The formulas used in the fuzzy extension resemble the original naive Bayes equations, so the classification process is still fast and reliable plus providing an incremental learning mechanism.
The Fuzzy Naive Bayesian classifier is implemented in a RoboCup simulation 3D team for decision making. It was tested specifically to evaluate the best receiver of a pass in a given situation. In the next sections, an explanation is given about the Fuzzy Naive Bayes model. Furthermore, it is compared versus a Gaussian Naive Bayes classifier, another approach of handling continuous attributes. Initial results obtained on this chapter for the Fuzzy Naive Bayesian classifier applied to decision making in RoboCup 3D were published in (Bustamante et al., 2006). Later performance comparison to Gaussian Naive Bayes classifier in the same pass skill scenario was published in (Bustamante et al., 2006b).

5.1 Naïve Bayes and the Fuzzy Extension

The Naive Bayes classifier is a simple Bayesian network with one root node that represents the class and n leaf nodes that represent the attributes. Let C be a class label with k possible values, and $X_1 ... X_n$ be a set of attributes or features of the environment with a finite domain $D(X_i)$ where $i = 1..n$. The classifier is given by the combination of the Bayesian probabilistic model with a maximum a posteriori (MAP) rule, also called discriminant function (Rish, 2001). The Naive Bayes classifier is defined as follows

$$NBayes(a) = argmax_{c \in C} P(c) \prod_{i=1}^{n} P(x_i|c) \tag{46}$$

where $a = \{X_1 = x_1, ..., X_n = x_n\}$ is a complete assignation of attributes, i.e. a new example to be classified, x_i is a short for $X_i = x_i$ and c is a short for $C = c$. The equation assumes conditional independence between attributes.
To deal with continuous variables, the domain of attributes can be crisp partitioned, but that could cause a loss of information (Friedman & Goldszmidt, 1996). We use a better method proposed in (Störr, 2002), namely a Fuzzy Bayesian classifier, a hybrid approach in which attributes are fuzzified before classification. The Fuzzy Naive Bayesian classifier is defined as

$$FNBayes(a) = argmax_{c \in C} P(c) \sum_{x_{1j} \in X_1} \frac{P(x_{1j}|c)}{P(x_{1j})} \mu_{x_{1j}} .. \sum_{x_{nj} \in X_n} \frac{P(x_{nj}|c)}{P(x_{nj})} \mu_{x_{nj}} \tag{47}$$

where $j = 1..D(X_i)$ and $\mu_{x_{ij}} \in [0, 1]$ denotes a membership function or degree of truth of attribute value $x_{ij} \in X_i$ in a new example a. All degrees of truth must be normalized such that $\sum_{x_{ij} \in X_i} \mu_{x_{ij}} = 1$ for all attributes $i = 1..n$.

The probabilities required by the fuzzy model can be calculated similarly to classical Naive Bayes as

$$P(C=c)=\frac{(\sum_{e\in L}\mu_c^e)+1}{|L|+|D(C)|} \tag{48}$$

$$P(X_i=x_i)=\frac{(\sum_{e\in L}\mu_{x_i}^e)+1}{|L|+|D(X_i)|} \tag{49}$$

$$P(X_i=x_i|C=c)=\frac{(\sum_{e\in L}\mu_{x_i}^e\mu_c^e)+1}{(\sum_{e\in L}\mu_c^e)+|D(X_i)|} \tag{50}$$

where Laplace-correction (Zadrozny & Elkan, 2001) applied to smooth calculations avoiding extreme values obtained with small training sets. Here L is the set of all training examples e, where $e=\{X_1=x_1,...,X_n=x_n,C=c\}$, $|L|$ refers to the number of examples $e\in L$, $\mu_c^e\in[0,1]$ denotes the degree of truth of $c\in C$ in a example $e\in L$, and $\mu_{x_i}^e\in[0,1]$ is the membership of attribute $x_i\in X_i$ in such example. All degrees of truth must be normalized such that $\sum_{c\in C}\mu_c^e=1$ and $\sum_{x_i\in X_i}\mu_{x_i}^e=1$.

5.2 Gaussian Naïve Bayes

One typical way to handle continuous attributes in the Naive Bayes classification is to use Gaussian distributions (Mitchell, 1997) to represent the likelihoods of the features conditioned on the classes. Thus each attribute is defined by a Gaussian probability density function (PDF) as

$$X_i\sim N(\mu,\sigma^2) \tag{51}$$

The Gaussian PDF has the shape of a bell and is defined by the following equation

$$N(\mu,\sigma^2)(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{52}$$

where μ is the mean and σ^2 is the variance. In Naive Bayes, the parameters needed are in the order of $O(nk)$, where n is the number of attributes and k is the number of classes. Specifically we need to define a normal distribution $P(Xi|C)\sim N(\mu,\sigma^2)$ for each continuous attribute. The parameters of such normal distributions can be obtained with

$$\mu_{X_i|C=c}=\frac{1}{N_c}\sum_{i=1}^{N_c}x_i \tag{53}$$

$$\sigma^2_{X_i|C=c}=\frac{1}{N_c}\sum_{i=1}^{N_c}{x_i}^2-\mu^2 \tag{54}$$

where N_c is the number of examples where $C = c$ and N is the number of total examples used for training. Calculating $P(C = c)$ for all classes is easy using relative frequencies such that

$$P(C = c) = \frac{N_c}{N} \tag{55}$$

5.3 Empirical Scenarios

Selecting a good scenario for training the classifiers is not trivial. In simulated soccer, there is a large set of possible scenarios for a given skill. The pass evaluation skill was chosen as the test-bed for the training of both classifiers. One of the reasons why it was selected is that passing is a fundamental characteristic of an agent that aims to play a soccer game. Specifically, deciding what teammate is the best receiver in a given situation could lead to better chances to score later in the game.

The scenario used to obtain the training set is explained below. A passer agent is placed in the center of the field with the ball at a distance of $d_{AB} \in [kickrange, 2]$, where $kickrange$ is the minimum kicking radial distance between the agent and the ball stated in the soccer server. A teammate agent is placed near the ball at a distance $d_{TB} \in [2, 20]$. An opponent agent is placed similarly, with a distance $d_{OB} \in [2, 20]$ from the ball. The angle between the teammate and the opponent from the ball's view point must be $\alpha \in [0, \frac{\pi}{6}]$.

The passer agent aligns with the ball to pass it to its teammate and both the teammate and the opponent try to intercept the pass. Once the teammate touched the ball, the episode is labeled as *SUCCESS*. If the opponent touches the ball first, the episode is labeled as *MISS*.

A graphical representation of this scenario is shown in figure 6.

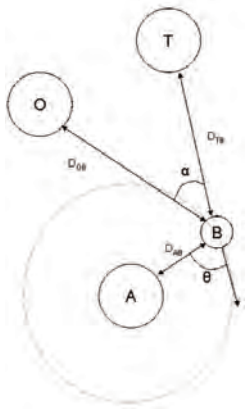


Fig. 6. Training scenario for supervised learning of parameters of each classifier. Three agents are involved: a passer agent (A), a receiver teammate (T) and an opponent (O). The ball is marked as (B)

In the case of the Fuzzy Naive Bayes classifier, aside of obtaining the probabilities of the bayesian model, we have to establish the fuzzy sets for each variable. Fuzzy sets represent linguistic values and are mathematically expressed with membership degree functions. We

defined the fuzzy sets for each variable heuristically. The sets chosen for distance to the ball d_{AB}, distance to teammate d_{AT} and distance to opponent d_{AO} variables are {*short, medium, long*}, and for θ and α variables are {*closed, medium, wide*}. A graphical representation of each fuzzy variable is shown in figure 7.

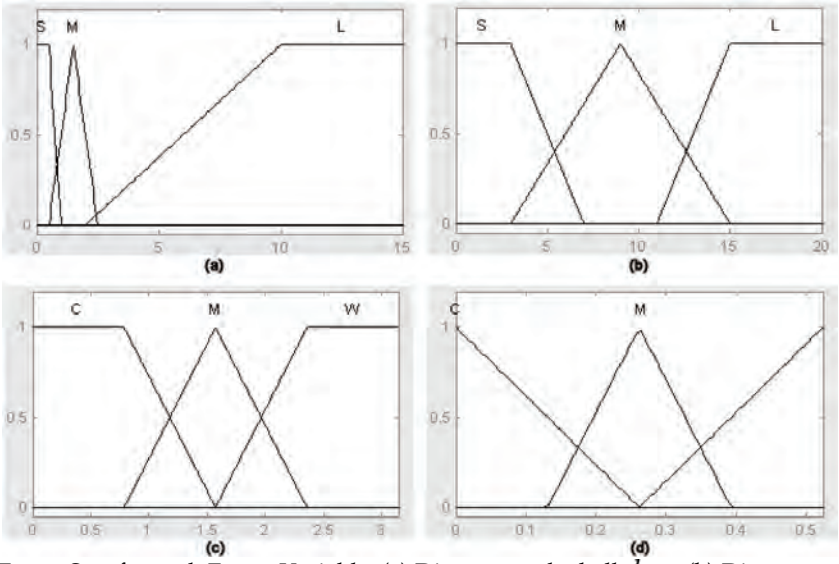


Fig. 7. Fuzzy Sets for each Fuzzy Variable. (a) Distance to the ball d_{AB}, (b) Distance to teammate d_{AT} and distance to opponent d_{AO}, (c) Alignment Angle θ and (d) Angle between teammate and opponent α

5.4 Experiments

For evaluating the efficiency in the domain of interest, we created a simulated-soccer test-scenario shown in figure 8. The ball is placed at $(x = -20, y = 0)$ and the agent is placed at $(x \in [-22, -18], y \in [kickrange, 2])$. After that, three teammate agents and four opponents are placed randomly at $(x \in [-30, -10], y \in [10, 30])$.

The passer uses a classifier to choose the best receiver teammate, i.e. the teammate with better chances to intercept the pass successfully. The passer uses the classifier evaluating all 1 vs. 1 competitions between each teammate and each opponent (because the classifier was trained this way). Then it selects the teammate with the maximum probability of success given its worst probability in all its 1 vs. 1 competitions, formally

$$Receiver = argmax_{t \in T}\ argmin_{o \in O}\ P(SUCCESS_{to}) \tag{56}$$

being T the set of all teammates, O the set of opponents and $P(SUCCESS_{to})$ is the probability of success of the competition between teammate $t \in T$ and opponent $o \in O$.

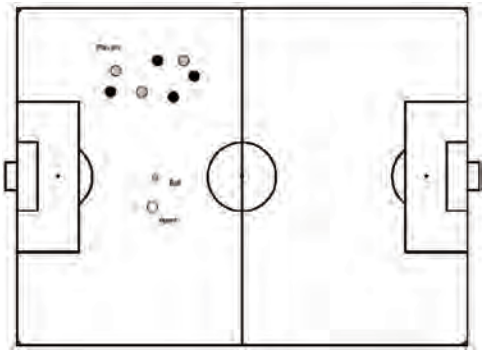


Fig. 8. Test scenario for the pass evaluation skill. Four opponent agents (black circles) and three teammates (gray circles) are placed randomly in a certain area. The passer (white circle) and the ball (little circle) are placed a few meters away

Table 5 summarizes the success rates of Fuzzy Naive Bayes, the Gaussian Naive Bayes and additionally, a random strategy after 500 episodes.

Class	Fuzzy Naive Bayes	Gaussian Naive Bayes	Random Strategy
SUCCESS	80.8	79.6	56.6
MISS	19.2	20.4	43.4

Table 5. Percentage of successful passes after 500 episodes on the test scenario

As we can see in table 5, both the Fuzzy Naive Bayes classifier and the Gaussian Bayes classifier outperform the random strategy. But the difference between the Fuzzy Bayes and the Gaussian Bayes approaches is indiscernible. However, recall that fuzzy variables and fuzzy sets for each variable were chosen heuristically. This leaves an open path for researching the use of better variables and more accurate sets to increase the performance of the hybrid classifier.

8. References

Bustamante, C.; Garrido, L. & Soto, R. (2006a). Fuzzy Naive Bayesian Classification in RoboSoccer 3D: A hybrid approach to decision making, *Proceedings of the RoboCup International Symposium*, , Bremen, Germany, June 2006, Springer Verlag

Bustamante, C.; Garrido, L. & Soto, R. (2006b). Comparing Fuzzy Naive Bayes and Gaussian Naive Bayes for Decision Making in Robocup 3D, *Proceedings of the 5th. Mexican International Conference on Artificial Intelligence (MICAI 06)*, pp. 237-247, ISBN 3540490264, Apizaco, Tlaxcala, Mexico, November 2006, Springer

Bustamante, C.; Flores, C. & Garrido, L. (2007). A Physics Model for the RoboCup 3D Soccer Simulation, *Proceedings of Agent-Directed Simulation Symposium (ADS 07)*, Norfolk VA, USA, March 2007

Cox, I. J. (1991). Blanche -- an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation,* Vol. 7, No. 2, 1991, pp. 193-204

Fox, D. (1998). *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation, Ph.D. Thesis,* University of Bonn, Germany

Fox, D.; Burgard, W. & Thrun, S. (1999). Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research,* Vol. 11, 1991, pp. 391-427

Friedman, N. & Goldszmidt, M. (1996). Discretization of continuous attributes while learning Bayesian networks, *Proceedings of 13th International Conference on Machine Learning*, pp. 157-165, Morgan Kaufmann, San Francisco, CA

Langley, P.; Iba, W. & Thompson, K. (1992). An Analysis of Bayesian Classifiers, *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 223-228, AAAI Press and MIT Press, USA

Lewis, D. (1998). Naive Bayes at forty: The independence assumption in information retrieval, *Proceedings of the 10th European Conference on Machine Learning*, pp. 4-15, ISBN 3540644172, Chemnitz, DE, 1998, Springer Verlag

Marion, J. & Thornton, S. (2003). *Classical dynamics of particles and systems,* Brooks Cole, ISBN 0534408966, San Diego

Mitchell, T. (1997). *Machine Learning,* McGraw Hill, ISBN 0070428077, New York

Rish, I. (2001). An empirical study of the naive bayes classifier, *Proceedings of IJCAI-01 workshop on Empirical Methods in AI*, pp. 41-46

Röfer, T.; Laue, T. & Thomas, D. (2005). Particle-Filter-Based Self-localization Using Landmarks and Directed Lines, *Proceedings of the RoboCup International Symposium*, pp. 608-615, ISBN 3540354379, Osaka, Japan, July 2005, Springer

Ronghua, L. & Bingrong, H. (2004). Coevolution Based Adaptive Monte Carlo Localization (CEAMCL). *International Journal of Advanced Robotic Systems,* Vol. 1, No. 3, September 2004, pp. 183-190

Russell, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach,* Prentice Hall, ISBN 0137903952, Englewood Cliffs, NJ, USA

Skare, Ø.; Bølviken, E. & Holden, L. (2003). Improved Sampling-Importance Resampling and Reduced Bias Importance Sampling. *Scandinavian Journal of Statistics,* Vol. 30, No. 4, December 2003, pp. 719-737

Störr, H. P.; Xu, Y. & Choi, J. (2002). A compact fuzzy extension of the Naive Bayesian classification algorithm, *Proceedings of InTech/VJFuzzy*, pp. 172-177, Hanoi, Vietnam, 2002, Science and Technics Publishing House, Hanoi, Vietnam

Thrun, S.; Fox, D.; Burgard, W. & Dellaert, F. (2000). Robust Monte Carlo Localization for Mobile Robot. *Artificial Intelligence,* Vol. 128, No. 1, 2000, pp. 99-141

Thrun, S.; Burgard, W. & Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents),* The MIT Press, ISBN 0262201623, Cambridge, Massachusetts

Xu, Y.; Jiang, C. & Tan, Y. (2006). SEU-3D Soccer Simulation Team Description, *Proceedings of the RoboCup International Symposium*, Bremen, Germany, June 2006, Springer Verlag

Zadrozny, B. & Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers, *Proceedings of 18th International Conference on Machine Learning*, pp. 609-616, ISBN 1558607781, Williams College, Massachusetts, June 2001, Morgan Kaufmann, San Francisco, CA

7

Simulated Environment in Robot Soccer

Gregor Klančar and Rihard Karba
University of Ljubljana
Slovenia

1. Introduction

In the last two decades the concept of multi-agent mobile systems has been observed in many computer simulations, laboratory examples and in some practical applications. Among such systems robot soccer has shown to be a very popular research game and has served as a perfect example of multi-agent systems in the last few years (Ferber, 1999; Moss & Davidsson, 2002; Stone & Veloso, 2000).
In this work the mathematical background of the developed robot soccer simulator is presented. The main purpose of the simulator design procedure is to obtain a realistic simulator which would be used as a tool in the process of strategy and control algorithms design for real world robot soccer as well as for other mobile-robotics related topics. To assure transferability to the real system the obtained strategy algorithms have to be designed on a realistic simulator. The main motivation for robot soccer simulator development was to design and study multi-agent control and strategy algorithms in FIRA Middle or Large League MiroSot category (5 against 5 or 11 against 11 robots). However, on FIRA's (Federation of International Robot Soccer Association) official website (www.fira.net) there exists a simulator for SimuroSot league, which could only be used in Middle League MiroSot (5 against 5 robots). A similar simulator was built by (Liang & Liu, 2002) where robot motion is simulated by dynamic model, collisions remaining oversimplified. There also exist a number of other simulator applications but not many papers are available. An important part of every realistic robot soccer simulator is collision modelling and simulation. Good mathematical background in rigid body collisions modelling and simulation could be found in (Baraf, 1997). Another useful contribution in the field of robotic simulator is (Larsen, 2001) where collisions are treated by spring-dumper approach rather than by impulse force only. The use of spring-dumper linkage in collisions makes velocities changes continuous, which is less problematic for simulation than discontinuous change of velocities (Fremond, 1995) obtained by impulse usage. However, spring and dumper coefficients are not easy to identify. Moreover, when observed from macroscopic time scale (as it is in simulation) collisions are indeed discontinuous events.
Simulated robots should have a realistic shape, which should not be represented simply with a square (the real shape of the robot is not a square) otherwise the simulation of ball guidance and other collisions becomes unrealistic. Furthermore, some of the available robot soccer simulators do not treat collisions well, especially the collisions among robots (robot corners), collisions between robot and boundary and situations where the ball is in-between

two robots or robot and boundary. Algorithms on such simulators are also not transferable enough to the real system. A majority of them is used for competitions in simulation league and these simulators do not need to be realistic.

With a rapid progress of computer graphics used in computer games, animated movies and other purposes a number of physics engines have appeared which can realistically simulate rigid body dynamics considering variables such as mass, inertia, velocity, friction, etc. Some of available physics engines are ODE – Open Dynamics Engine, Ageia physX, AERO, Karma in Unreal Engine and many others. Their usege enables computer simulations, animations and games such as racing games to appear more realistic. Depending on their usage there exist two types of physics engines, namely real-time and high precision. When dealing with interactive computing (e. g. video games), the physics engines are simplified in order to perform in real-time. On the other hand high precision physics engines require more processing power to be able to calculate very precise physics and are usually used by scientists and computer animated movies. Some of physics engines are free and open source. As such they can also be used to simulate physics in different research oriented experiments. These packages are usually comprehensive and therefore quite difficult to manage, use and modify. When constructing the mobile robot its mathematical background was completely developed by our team, which enabled us to get a better insight into the problem domain and gave us the possibility to efficiently solve some simulator specifics as mentioned in the sequel.

The presented simulator is mainly used as a tool in control and strategy design of multi-agent system in real game and therefore needs to be realistic. Strategy design could be developed also on a real plant but there are some important reasons which benefit the usage of realistic simulator as stated in the paper. Some vital parts of the simulator are explained and modelled in more detail, beginning with the kinematics and dynamic motion modelling considering kinematics constraints and, further on dealing with different collisions modelling. The stress is given to the motion modelling where the assumptions of pure rolling conditions are made and dynamic properties are included. The results of this part are motion models of the ball and the robot with differential drive. Some new ideas of collision formulation and realization (taking into account the real robot shape) are used as well. Collisions are simply solved by mathematically correct discontinuous change of velocities (states of the velocity integrators), which is more convenient for realization than simulating collisions by applying impulse force (Baraf, 1997; Larsen, 2001). However, collisions are only described by approximate models, which are sufficient enough for realistic behaviour of the obtained simulator. Precise collisions modelling is usually very demanding because of many factors, which should be considered during collision. When simulating a realistic game a precise collision modelling is less important than motion modelling. This is because the game strategy is designed to play a good game where different collisions are undesired and we want to avoid them. Nevertheless collisions still happen and have to be handled. The problems of collision detection and the method of finding the exact time of the collision are exposed too. For the latter the existing algorithms in Matlab Simulink are used.

The system presented in this paper is available for other researchers. It can be used for mobile-robot related experiments, such as multi-agent strategy design, agent behaviour analysis, robot motion planning, cooperation, collision avoidance, motion planning, control and the like. The presented simulation is available at our website (Klančar, 2007).

The work is organized as follows. First, a brief system overview is revealed, followed by the mathematical model derivation of basic agents (robots and ball). Then some new ideas of collisions modelling considering complex robot shape are presented in more detail. Finally some experimental results and conclusions are given.

2. System Overview

The robot soccer set-up (see Fig. 1) consists of ten Middle League MiroSot category robots (generating two teams) of size 7.5*cm* cubed, orange golf ball, rectangular playground of size 2.2×1.8*m*, colour camera and personal computer. Colour camera is mounted above playground (each team has its own) and is used as a global motion sensor. The objects are identified from their colour information; orange ball and colour dresses of robots. The agent-based control part of the programme calculates commands for each agent (robot) and sends them to the robot by a radio connection. The robots are then driven by two powerful DC motors; one for each wheel.

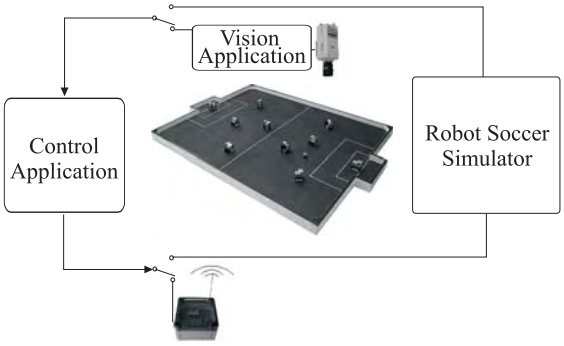


Fig. 1. Robot soccer system overview

The role of the simulator developed in the paper is to replace the real playground, camera, robots and ball, which is expensive and needs a large place to be set up. Therefore the simulator must include mathematical models of motion as well as collisions which happen on the playground.

3. Mathematical Modelling

To simulate robot soccer game mathematic motion equations should be derived first. The playground activities consist of two kinds of moving objects: robot and ball. Therefore their motion modelling (Egeland, 2002) is presented in the sequel.

3.1 Robot Model

The robot has a two-wheel differential drive located at the geometric centre, which allows zero turn radius and omni-directional steering because of nonholonomic constraint (Kolmanovsky & McClamroch, 1995). It is an active object in the robot soccer game. Its

appearance is given in Fig. 2 and its motion is described in the sequel by kinematics and dynamic motion equations.

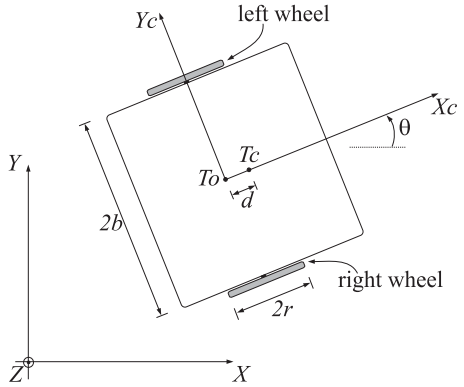


Fig. 2. Symbol description

Where $T_o=(x_o, y_o)$ is robot geometric centre, $T_c=(x_c, y_c)$ is its mass centre, m_c is body mass, m_k is wheel mass and J_c, J_k, J_m are moments of inertia for robot body around axis Z, for wheel around its axle and wheel around axis Z, respectively. Supposing pure rolling conditions of the wheels, the following kinematics constraints can be written:

$$\begin{aligned} \dot{y}_c \cos\theta - \dot{x}_c \sin\theta - \dot{\theta} d &= 0 \\ \dot{x}_c \cos\theta + \dot{y}_c \sin\theta + b\dot{\theta} &= r\dot{\phi}_r \\ \dot{x}_c \cos\theta + \dot{y}_c \sin\theta - b\dot{\theta} &= r\dot{\phi}_l \end{aligned} \tag{1}$$

Where θ is robot orientation, ϕ_r and ϕ_l are angles describing wheels rotation and d is distance between mass centre and geometric centre. According to the first constraint in Eq. (1), the robot cannot slide in the sideways, while the second and the third constraints describe pure rolling of the wheels. The null space of kinematics constraints (1) defines robot kinematics motion equation, given as:

$$\begin{bmatrix} \dot{x}_c \\ \dot{y}_c \\ \dot{\theta} \\ \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} = \begin{bmatrix} \frac{r}{2b}\left(b\cos(\theta) - d\sin(\theta)\right) & \frac{r}{2b}\left(b\cos(\theta) + d\sin(\theta)\right) \\ \frac{r}{2b}\left(b\sin(\theta) + d\cos(\theta)\right) & \frac{r}{2b}\left(b\sin(\theta) - d\cos(\theta)\right) \\ \frac{r}{2b} & -\frac{r}{2b} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\phi}_r \\ \dot{\phi}_l \end{bmatrix} \tag{2}$$

Dynamics motion equation can further be derived using Lagrange formulation (Welles, 1967)

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_k}\right) - \frac{\partial L}{\partial q_k} + \frac{\partial P}{\partial \dot{q}_k} = f_k - \sum_{j=1}^{m} \lambda_j a_{jk} \tag{3}$$

the last part of Eq. (3), λ_j are Lagrange multiplicators associated with j-th (j=1…3) constraint equation and a_{jk} is k-th (k=1…5) coefficient of j-th constraint equation. Lagrangian is defined as:

$$L = \frac{m_c}{2}\left(\dot{x}_c{}^2 + \dot{y}_c{}^2\right) + \frac{m_k}{2}\left(\dot{x}_{k_r}{}^2 + \dot{y}_{k_r}{}^2\right) + \frac{m_k}{2}\left(\dot{x}_{k_l}{}^2 + \dot{y}_{k_l}{}^2\right) + \\ + \frac{J_c}{2}\dot{\theta}^2 + 2\frac{J_m}{2}\dot{\theta}^2 + \frac{J_k}{2}\dot{\phi}_r{}^2 + \frac{J_k}{2}\dot{\phi}_l{}^2 \tag{4}$$

Defining $m=m_c+2m_k$, $J=J_c+2J_m+2m_k(d^2+b^2)$ and expressing (4) by robot mass centre variables the following is obtained:

$$L = \frac{m}{2}\left(\dot{x}_c{}^2 + \dot{y}_c{}^2\right) + \frac{J}{2}\dot{\theta}^2 + \frac{J_k}{2}\dot{\phi}_r{}^2 + \frac{J_k}{2}\dot{\phi}_l{}^2 + 2m_k d\dot{\theta}\left(\dot{x}_c \sin\theta - \dot{y}_c \cos\theta\right) \tag{5}$$

According to (3) the dynamic model is written as:

$$\begin{aligned} m\ddot{x}_c + 2m_k d\left(\ddot{\theta}\sin\theta + \dot{\theta}^2\cos\theta\right) - \lambda_1 \sin\theta + \left(\lambda_2 + \lambda_3\right)\cos\theta = 0 \\ m\ddot{y}_c - 2m_k d\left(\ddot{\theta}\cos\theta - \dot{\theta}^2\sin\theta\right) + \lambda_1 \cos\theta + \left(\lambda_2 + \lambda_3\right)\sin\theta = 0 \\ J\ddot{\theta} + 2m_k d\left(\ddot{x}_c \sin\theta - \ddot{y}_c \cos\theta\right) - \lambda_1 d + \left(\lambda_2 - \lambda_3\right)b = 0 \\ J_k\ddot{\phi}_r + \mu\dot{\phi}_r - \lambda_2 r = \tau_r \\ J_k\ddot{\phi}_l + \mu\dot{\phi}_l - \lambda_3 r = \tau_l \end{aligned} \tag{6}$$

where λ_1, λ_2, λ_3 are Lagrange multiplicators which can effectively be eliminated by the procedure given in (Oriolo et al., 2002; Sarkar, 1994). Brief summary is given in the sequel. Lagrangian formulation (3) can be expressed in matrix form, such as:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{F}(\dot{\mathbf{q}}) = \mathbf{E}(\mathbf{q})\mathbf{u} - \mathbf{A}^T(\mathbf{q})\boldsymbol{\lambda} \tag{7}$$

where $\mathbf{M}(\mathbf{q})$ is inertia matrix, $\mathbf{V}(\mathbf{q},\dot{\mathbf{q}})$ is vector of position and velocity dependent forces, $\mathbf{F}(\dot{\mathbf{q}})$ is vector of friction or dumping forces, $\mathbf{E}(\mathbf{q})$ is input transformation matrix, $\mathbf{u}$ is input vector of actuator forces and torques and $\mathbf{A}(\mathbf{q})$ is the matrix of kinematics constraints. System kinematics from Eq. (2) expressed in matrix form reads:

$$\dot{\mathbf{q}} = \mathbf{S}(\mathbf{q})\mathbf{v}(t) \tag{8}$$

and matrix form of kinematics constraints from Eq. (1) is

$$\mathbf{A}(\mathbf{q})\dot{\mathbf{q}} = 0 \tag{9}$$

Calculating first derivative of (8) gives

$$\ddot{\mathbf{q}} = \dot{\mathbf{S}}\mathbf{v} + \mathbf{S}\dot{\mathbf{v}} \tag{10}$$

Lagrange multiplicators can finally be eliminated by substituting (8) and (10) in Eq. (7) and pre-multiplying by $\mathbf{S}^T$. The part with Lagrangian multiplicators vanish because $\mathbf{S}^T\mathbf{A}^T=\mathbf{0}$.
The dynamics of electric part (the motors) can usually be neglected, as electrical time constants are usually significantly smaller than mechanical time constants.

3.2 Ball Model

The ball is a passive object whose motion across the playground can be described by five generalized coordinates as shown in Fig. 3.

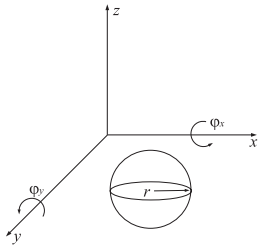


Fig. 3. The ball rolling on the plane

Dynamics motion equation can be derived using Lagrange formulation

$$\frac{d}{dt}\left[\frac{\partial L}{\partial \dot{q}_k}\right]-\frac{\partial L}{\partial q_k}+\frac{\partial P}{\partial \dot{q}_k}=f(t) \tag{11}$$

where L stands for difference between kinetic and potential energy, P stands for power function (dissipation function), q_k stands for generalized coordinate and $f(t)$ is external force respectively and is nonzero when the ball collides. Lagrangian is defined as

$$L=\frac{1}{2}m(\dot{x}^2+\dot{y}^2)+\frac{1}{2}J(\dot{\varphi}_x{}^2+\dot{\varphi}_y{}^2+\dot{\varphi}_z{}^2) \tag{12}$$

where m is the ball mass and J is moment of inertia. Supposing pure rolling conditions the following kinematics constraints follow

$$\begin{aligned}\dot{x}+r\dot{\varphi}_y&=0\\ \dot{y}-r\dot{\varphi}_x&=0\end{aligned} \tag{13}$$

where r is ball radius. Both conditions in Eq. (13) give perfect rolling of the ball, i. e. motion with no slipping. Constraints in Eq. (13) are holonomic (integrable) and can be used to eliminate two generalized coordinates. Further on, by neglecting rotation around z axis $\omega_z=0$ and using constraints (13), equation (12) is rewritten as

$$L=\frac{m+\dfrac{J}{r^2}}{2}\left(\dot{x}^2+\dot{y}^2\right) \tag{14}$$

The power function is

$$P=\frac{1}{2}f_D\dot{x}^2+\frac{1}{2}f_D\dot{y}^2 \tag{15}$$

where f_D is dumping coefficient. Considering (11) the final motion equation of the ball are as follows

$$\begin{aligned}\ddot{x}&=\frac{F(t)-\dot{x}\cdot f_D}{m+J/r^2}\\ \ddot{y}&=\frac{F(t)-\dot{y}\cdot f_D}{m+J/r^2}\end{aligned} \tag{16}$$

4. Collisions Modelling

During the motion of the robots and the ball on the playground several collisions between them are possible. They are given as submodels and describe the collision between moving objects: the robot-ball collision model, the robot-boundary collision model, the ball-boundary collision model and the collision between robots model. When simulating a realistic game, a precise collision modelling is less important than motion modelling. This is because the game strategy is designed to play a good game where different collisions are undesired and we want to avoid them. Nevertheless collisions still happen and have to be handled. However, in the sequel the collision models only approximately describe real situations. Most of the presented models are therefore relatively simple for realization in a simulator.

4.1 Robot-Boundary Collision

When modelling collision of the robot to the boundary, the test whether all robot corners are inside the playground must be performed first. If they are, this means that there is no such collision. The procedure is represented by diagram in Fig. 4.

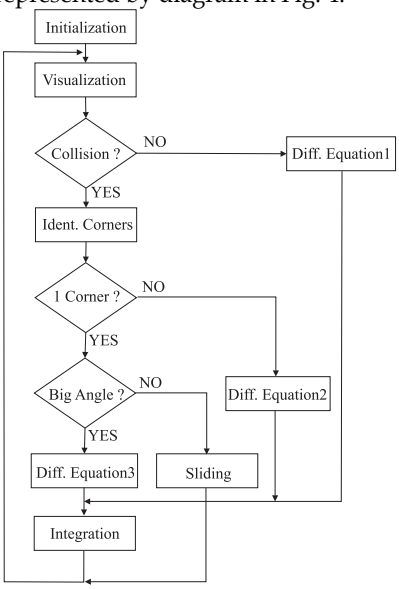


Fig. 4. Robot–boundary collision simulation diagram

The notation Diff. Equation 1 in Fig. 4 stands for Eq. (3). When the robot hits the boundary with two corners, it stops and so robot kinematics equation (in Fig. 4 marked as Diff. Equation 2) becomes:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

More demanding case appears when the robot hits the boundary with one corner only. If the angle between the robot and the boundary is greater than the proposed threshold value, the robot starts to rotate around the corner (see Fig. 5).

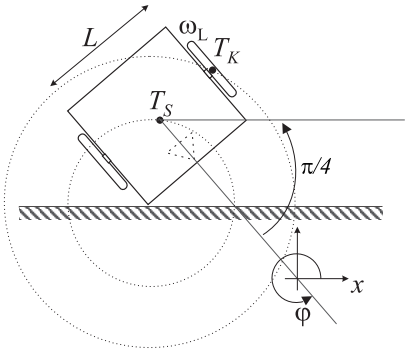


Fig. 5. One-corner collision with the boundary

The velocity in point T_K with tangential direction to the outer circle in Fig. 5 is obtained by a transformation of the left wheel rim velocity ($\omega_L \cdot r$). Angular velocity ω_{TK} in point T_K is thus:

$$\omega_{T_K} = \frac{\omega_L \cdot r \cdot \cos(\alpha)}{\sqrt{L^2 + L^2/4}} \tag{18}$$

where angle α is

$$\alpha = \operatorname{arctg}\left(\frac{L/2}{L}\right) \tag{19}$$

and linear velocity of the robot centre (v_{Ts}) is:

$$v_{T_S} = \omega_{T_K}\sqrt{\frac{L^2}{2}} = \omega_L r\sqrt{\frac{2}{5}}\cos(\alpha) \tag{20}$$

Robot kinematics equation (in Fig. 4 marked as Diff. Equation 3) then becomes:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\varphi}\end{bmatrix} = \begin{bmatrix}\sqrt{\frac{2}{5}}\cdot r\cdot\cos(\alpha)\cdot\cos\left(\varphi+\frac{\pi}{4}\right) & 0\\ \sqrt{\frac{2}{5}}\cdot r\cdot\cos(\alpha)\cdot\sin\left(\varphi+\frac{\pi}{4}\right) & 0\\ -\sqrt{\frac{4}{5}}\cdot\frac{r}{L}\cdot\cos(\alpha) & 0\end{bmatrix}\cdot\begin{bmatrix}\omega_L\\ \omega_R\end{bmatrix} \tag{21}$$

If the angle between the robot and the boundary is less than the mentioned threshold, the robot slides along the boundary (see Fig. 4).

4.2 Ball-Boundary Collision

In the ball-boundary collision elastic collision is supposed. The velocity component parallel to the boundary remains the same, while the perpendicular velocity component changes sign and is multiplied by a factor less than one, representing energy loss. To assure proper rebound without penetration, zero crossing algorithm implemented in Matlab Simulink environment is used to treat the problem of integration over discontinuities correctly and efficiently. This algorithm simply changes the integration step by bisection, according to some input variable (distance between ball and boundary multiplied by sign which is negative if the ball is outside the playground), until the exact time of discontinuity appears.

4.3 Robot-Ball Collision

Mutual impact of the robot and the ball can be described with collision model of two spheres (Fig. 6). Mathematically the model is based on kinetic energy and momentum balance equations as follows

$$\begin{aligned} & m_1 v_{x_1}{}^2 + m_2 v_{x_2}{}^2 + m_1 v_{y_1}{}^2 + m_2 v_{y_2}{}^2 = \\ & = m_1 w_{x_1}{}^2 + m_2 w_{x_2}{}^2 + m_1 w_{y_1}{}^2 + m_2 w_{y_2}{}^2 \\ & m_1 v_{x_1} + m_2 v_{x_2} = m_1 w_{x_1} + m_2 w_{x_2} \\ & m_1 v_{y_1} + m_2 v_{y_2} = m_1 w_{y_1} + m_2 w_{y_2} \end{aligned} \tag{22}$$

where indexes 1 and 2 stand for the first and second sphere, v represents the velocities before and w the velocities after the collision, while m_1 is robot and m_2 ball mass respectively.

The playground coordinate system is rotated so that axis x connects mass centres of the spheres (see Fig. 6).

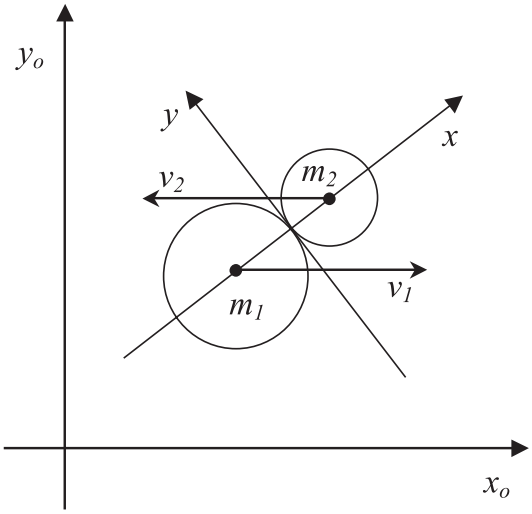


Fig. 6. Collision of two spheres

Because of the coordinate system rotation the impact force is different from zero only in normal direction of the collision, i. e. direction x. Thus the velocities in direction y remain the same. Final non-trivial velocities after the collision are then given by:

$$\begin{aligned} w_{x_1} &= \frac{-m_2 v_{x_1} + m_1 v_{x_1} + 2m_2 v_{x_2}}{m_1 + m_2} \\ w_{x_2} &= \frac{2m_1 v_{x_1} + m_2 v_{x_2} - m_1 v_{x_2}}{m_1 + m_2} \\ w_{y_1} &= v_{y_1} \\ w_{y_2} &= v_{y_2} \end{aligned} \tag{23}$$

where index 1 stands for the robot and index 2 stands for the ball. If m_2 is very small in comparison with m_1, a simplification of Eq. (23) is justified. Some manipulations give:

$$\begin{aligned} w_{x_1} &= v_{x_1} \\ w_{x_2} &= v_{x_1} + k\left(v_{x_1} - v_{x_2}\right) \\ w_{y_1} &= v_{y_1} \\ w_{y_2} &= v_{y_2} \end{aligned} \tag{24}$$

Furthermore, energy loss is realized by multiplying the part of Eq. (24) inside the brackets by factor k less than one.

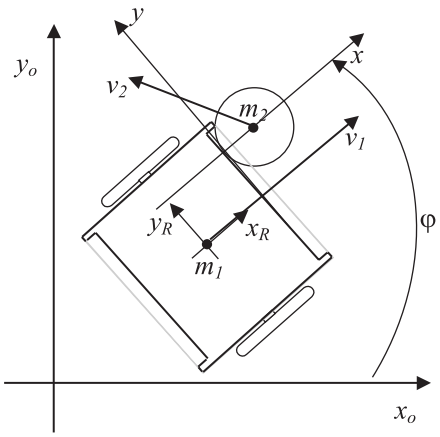


Fig. 7. Robot-ball collision

Calculated velocities after the collision are then used as new initial states of the integrators in the simulator. This is equivalent to applying and simulating impulse force caused by collision but is less suitable for realization (Egeland, 2002; The Math Works, 1998).
However to assure a realistic collision of the robot and the ball, a concrete robot shape has to be modelled. The actual robot shape is shown in collision situation in Fig. 7 and the idea of how to include the real robot shape into the model is given in Fig. 8.

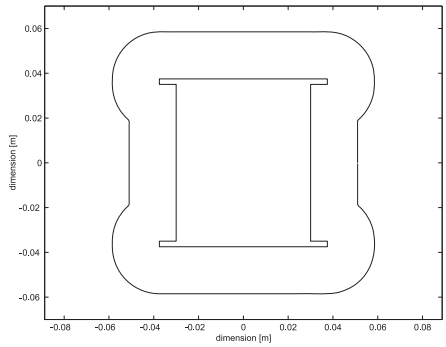


Fig. 8. Shape of the robot (inner) and its rim

The outer shape is the rim of the robot obtained if the ball is rolled around the robot and its positions are recorded. With the proposed reshaping the collision of the robot with the ball can be treated as a collision between two points (ball centre and point on robot rim). Because linear and angular velocities of the robot are given for geometrical centre, the following transformations have to be done in order to obtain the velocities in the point of the rim where the collision with the ball occurs:

$$\begin{aligned} v_{x_1} &= v - \omega r(\varphi)\sin\varphi \\ v_{y_1} &= \omega r(\varphi)\cos\varphi \end{aligned} \tag{25}$$

Function $r(\varphi)$ is the distance from the robot centre to the collision point on the rim and φ is the angle from the local robot axis x to the line connecting the robot centre and the collision point. To solve Eq. (23) the playground coordinates are rotated first so that axis x is in tangential direction of the rim (in the point of collision). After that the collision results are transformed to the global coordinates.

The shape of the robot is described with two look-up tables (distance $r(\varphi)$ and $tangent(\varphi)$ of the rim), which are addressed with angle φ. To detect if the ball hits the robot, a check of the distance between their centres must be performed. If the distance is less than the one obtained from look-up table $r(\varphi)$, the ball hits the robot. The accurate time of the collision is again obtained by zero crossing algorithm. So proper collision without penetration (within machine precision) and accurate integration over velocities are assured.

4.4 Collisions Between Robots

The collision of two or even more robots is undoubtedly problematic from the modelling point of view. However, the complexity of the model must be strongly dependent on the demands of the realistic simulator, where the compromise between reality approximation and simulation precision must be found according to the simulation usage aims. During simulator design a few more or less approximate solutions were tested until finally the best one was implemented. When designing the control strategy of the robot soccer game, it seems that collisions between robots are not so important because one focuses mainly on shots on goal, on passes, organizing defence and similar actions, while collisions between robots are more or less undesired. However, collisions between robots are quite frequent in

the game and in the case of defence also very important. Therefore they must be treated correspondingly in a realistic simulator.

Collision Detection

A collision detection algorithm (Klančar et al. ,2003) consists of two steps. In the first step only the information about a possible collision is obtained. The second step is then performed only if the possibility obtained from the first step exists. In the second step a separating plane between objects is found. The reason for performing collision detection in two steps is only due to lower computational burden. Thus, the second step is performed only in situations where collision is almost inevitable.

The first step is performed by analyzing bounding boxes of all robots. The latter have their sides parallel to the global coordinate axes, thus representing the rectangle in which robot in its current position is included (see Fig. 9). The possibility of two objects colliding exists only if the bounding boxes overlap. The overlapping between two bounding boxes is determined by checking if their sides overlap in both axis directions (x and y) at the same time.

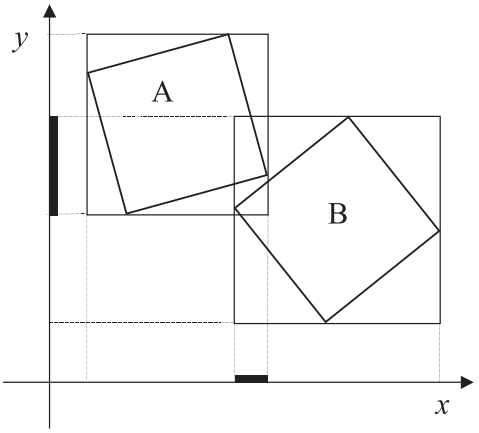


Fig. 9. Overlapping of bounding boxes in both directions

As mentioned before the second step is performed only if the overlapping of bounded boxes from the first step exists. The separating plane is calculated so that one object (convex polyhedrons) is on one side of the plane and the other on another side of the separating plane. The separating plane always exists if two objects do not invade.

Collision Realization

In a two-dimensional space the separating plane is a straight line. It is convenient that the separating plane has a normal in the same direction as is the normal direction of collision. A separating plane should thus contain the side of one of the two objects which are involved in collision (see Fig. 10).

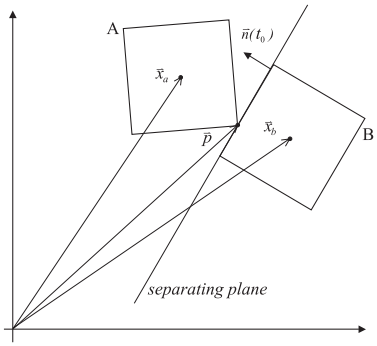


Fig. 10. Collision of two robots

When a collision of two robots appears, the following holds:

$$\Delta\vec{G} = \int \vec{F}dt \tag{26}$$

where $\vec{G}$ stands for conservation of momentum and $\vec{F}dt$ is force impulse acting at the time of collision. Because of the force impulse a sudden change in velocities of the two robots occurs. Force impulse acts only in normal direction of the collision. Thus only the velocity components in the normal direction of the collision change while perpendicular components remain the same. To calculate the new velocities of the robots after collision the force impulse $\vec{J} = \vec{F}\Delta t$ has to be calculated. The detailed procedure to estimate the velocities of two rigid bodies after collision is described in (Baraf, 1997; Klančar et al., 2003). The idea is to calculate the relative velocities in the collision point $\vec{p}$ (see Fig. 10) before and after the collision in normal direction. It is always true that the absolute value of the relative velocity in normal direction after the collision remains the same compared to the absolute value of the relative velocity in normal direction before collision in point $\vec{p}$. From that property the amplitude of force impulse can be calculated.

Force impulse in normal direction $\vec{n}$ of the collision (also normal of the separating plane at the time of the collision, see Fig. 10) of the two frictionless bodies is given by

$$\vec{J} = j\vec{n}(t_0) \tag{27}$$

where t_0 is time of the collision and j is amplitude of the force impulse. For the normal direction of the collision the following relation can be written

$$v_{rel}^{+} = -\varepsilon v_{rel}^{-} \tag{28}$$

meaning that absolute value of relative velocity in normal direction after collision v_{rel}^{+} remains the same or is lowered for energy loss factor ε in comparison with to absolute value of relative velocity in normal direction before collision v_{rel}^{-}. From the property (28) the amplitude of force impulse j in Equation (27) can be estimated according to procedure described in (Baraf, 1997). Let $\dot{\vec{p}}_a^{-}(t_0)$ be the velocity of contact point of robot A before impulse $\vec{J}$ is applied and $\dot{\vec{p}}_a^{+}(t_0)$ velocity of contact point of robot A after applying impulse. Similarly notations $\dot{\vec{p}}_b^{-}(t_0)$, $\dot{\vec{p}}_b^{+}(t_0)$ are used for the second robot B taking part in the collision. Relative velocity in normal direction before applying impulse is thus

$$v_{rel}^{-} = \vec{n}(t_0) \cdot (\dot{\vec{p}}_a^{-}(t_0) - \dot{\vec{p}}_b^{-}(t_0)) \tag{29}$$

and after applying impulse

$$v_{rel}^{+} = \vec{n}(t_0) \cdot (\dot{\vec{p}}_a^{+}(t_0) - \dot{\vec{p}}_b^{+}(t_0)) \tag{30}$$

Defining

$$\vec{r}_a = \vec{p} - \vec{x}_a(t_0) \tag{31}$$

where $\vec{r}_a$ is the displacement vector between mass centre $\vec{x}_a$ of the robot A and collision point $\vec{p}$. Further let $\vec{v}_a^{-}(t_0)$ and $\vec{\omega}_a^{-}(t_0)$ be the liner and angular velocity of robot A before and $\vec{v}_a^{+}(t_0)$ and $\vec{\omega}_a^{+}(t_0)$ after applying force impulse. The following velocities can be written for mass centre of robot A and for the point of collision

$$\vec{v}_a^{+}(t_0) = \vec{v}_a^{-}(t_0) + \frac{j\vec{n}(t_0)}{M_a} \tag{32}$$

$$\vec{\omega}_a^{+}(t_0) = \vec{\omega}_a^{-}(t_0) + I_a^{-1}(\vec{r}_a \times j\vec{n}(t_0)) \tag{33}$$

$$\dot{\vec{p}}_a^{+}(t_0) = \vec{v}_a^{+}(t_0) + \vec{\omega}_a^{+}(t_0) \times \vec{r}_a \tag{34}$$

Here M_a stands for mass of robot A and I is the corresponding moment of inertia. The same notation is used for robot B. Inserting Equations (32) and (33) to Equation (34), the following relation is obtained

$$\dot{\vec{p}}_a^{+}(t_0) = \dot{\vec{p}}_a^{-}(t_0) + j \cdot (\frac{\vec{n}(t_0)}{M_a} + I_a^{-1}(\vec{r}_a \times \vec{n}(t_0))) \times \vec{r}_a \tag{35}$$

The velocity in the contact point of robot B considering opposite direction of impulse force is thus

$$\dot{\vec{p}}_b^{+}(t_0) = \dot{\vec{p}}_b^{-}(t_0) - j \cdot (\frac{\vec{n}(t_0)}{M_b} + I_b^{-1}(\vec{r}_b \times \vec{n}(t_0))) \times \vec{r}_b \tag{36}$$

Inserting Equations (35) and (36) into Equation (30) and then combining obtained equation with Equation (28) the amplitude of impulse is finally calculated as

$$j = \frac{-(1+\varepsilon)v_{rel}^{-}}{\frac{1}{M_a} + \frac{1}{M_b} + \vec{n}(t_0) \cdot (I_a^{-1}(\vec{r}_a \times \vec{n}(t_0))) \times \vec{r}_a + \vec{n}(t_0) \cdot (I_b^{-1}(\vec{r}_b \times \vec{n}(t_0))) \times \vec{r}_b} \tag{37}$$

Having estimated the impulse, linear velocity $\vec{v}^{+}$ and angular velocity $\vec{\omega}^{+}$ for robot mass centre can be calculated by using relations (32), (33). It is namely equivalent to impulse force because of collision simulation but more suitable and accurate for realization. To obtain accurate t_0 zero crossing algorithm implemented in Matlab Simulink could be used in order to assure accurate integration of discontinuous velocities signals. This algorithm simply changes integration step by bisection, according to some input variable (distance between robots multiplied by a sign which is negative if robots penetrate), until exact time of discontinuity is achieved. However, the problem of high frequency oscillations around a discontinuity (chattering) appears when two or more robots stay in contact (robots pushing each other). Therefore step size of simulation becomes very small which results in halting of the simulation. Thus a better solution is to check for correspondingly small distance between one robot corner and the separating plane belonging to another robot. If separating plane does not exist, the time before penetration of the simulated robots must be taken into

account. The obtained velocities after the collision are then used to determine new initial states of the integrators in the simulator, which is equivalent to simulating impulse force because of the collision. The former is more suitable and accurate for realization, though.

5. Experimental Validation

In the sequel a few examples of simulator operation will be compared to the operation of a real set-up. In these comparisons similar conditions (initial pose, velocities and situations) are ensured. These visual comparisons give some impression about the capability of the simulator to realistically describe the real set-up.
The situation where the ball collides with the wall and the robot is presented in Fig. 11. Here the robot stands still while the ball starts to move with initial velocity v= 0.8 m/s. In the left graph of Fig. 11 the experiment result from the real set-up is presented while the right one shows a similar simulated experiment. In both figures the object shapes are drawn with 165 ms resolution (simulation sampling time is 33 ms).

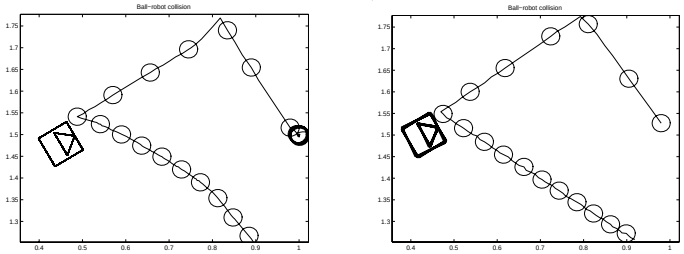


Fig. 11. Comparison of collision between the ball and the wall and between the ball and the robot on a real set-up (left) and on the simulator (right)

In both cases a similar ball motion is recorded. More interesting is the second ball collision where the ball hits the robot and rebounds from the robot specific shape presented in Figs. 7 and 8. The difference between both of thecompared figures is the course of the ball which is supposed to be a straight line on the simulator but in a real set-up it has a slight deviation from the straight line motion. This might happen because of the ball spinning effect after the collision and some other (stochastic) effects such us uneven terrain, dirt on the ground which were not considered in the simulator.
In Fig. 12 the simulated and real robot hits the boundary at the 45° angle relative to the boundary. In both cases the robot starts with constant velocity (v=0.5 m/s).

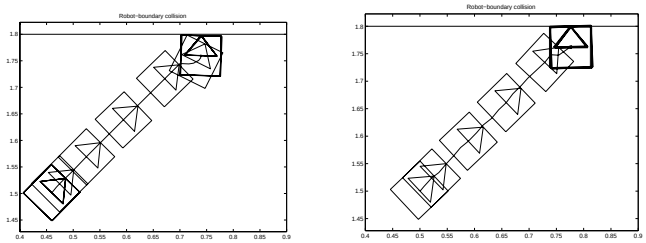


Fig. 12. Comparison of collision between the robot and the wall and between the ball and the robot on a real set-up (left) and on the simulator (right)

It can be observed that both examples in Fig. 12 (real and simulated) are almost identical. In Fig. 13 comparisons between robots from a real set-up (first column) and simulated robots (second column) for three different collision situations (rows in Figure 13) are given. The experiments were performed with the same initial conditions (starting positions, orientations and velocities).

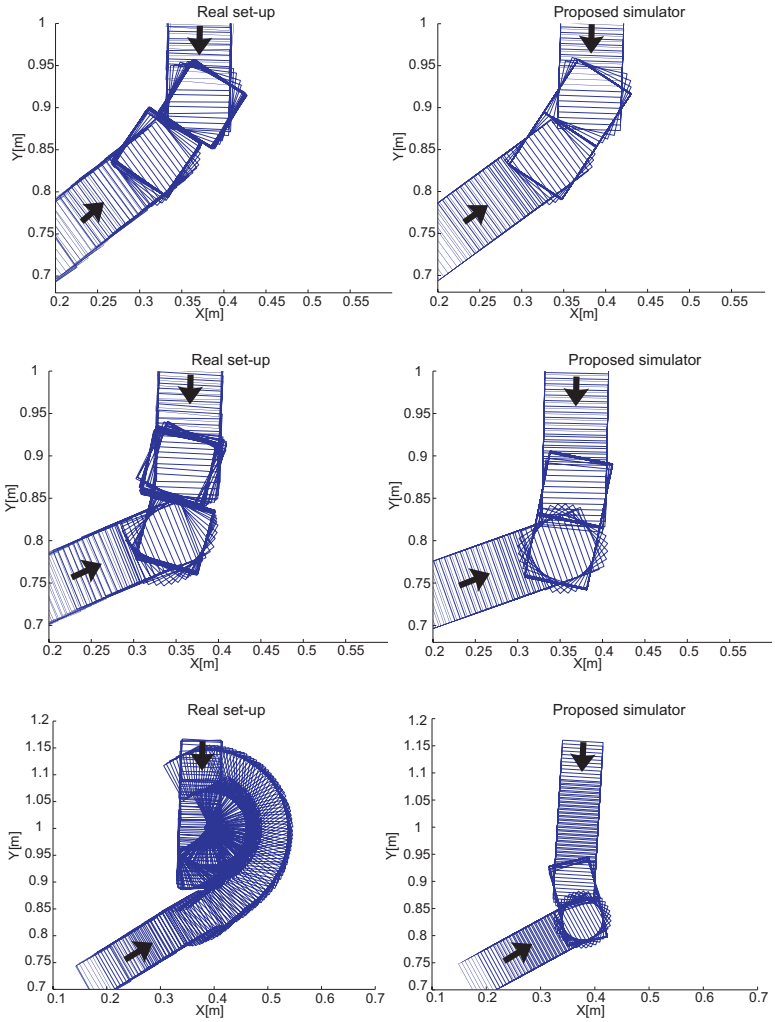


Fig. 13. Comparison of collisions between real robots and simulated robots

From the proposed representation also the estimation of robots course and their speeds in certain time (sample time is 33 *ms*) can be observed. The first and second row of Fig. 13 show the situation where compared subjects are relatively equal. The real situation where

robots wheels slide on the real set-up is shown in the third row in Fig. 13. Here of course simulator gives wrong results. It is evident that the proposed robots collision model captures the behaviour of the real robots to the reasonable extent, which means that simulated situations cover a vast majority of collisions in real game sufficiently well.
Presented collision models give sufficient representation of real situation. However, a lot of factors in real set-up are of significantly stochastic character what means that their modelling is not justifiable from the usable simulator point of view (fast enough on available personal computers, simple enough, etc.). The mentioned factors are: nonuniform friction, dirt or dust on the playground or wheels, shape of the robot, robot strength which depends on battery status, wheel sliding, friction is different for the direction along or perpendicular to the direction of wheels, etc. If comparison is performed over longer time interval shown results are useless due to above reasons. Main goal of the work however is to present reasonably accurate motion and collision models and thus contributes to obtain more realistic simulator, which would be used as a tool in the process of strategy and control algorithms design. Therefore, the validation of the simulator as a whole should be done through transferability of obtained strategy algorithms to the real system. It can be confirmed that the behaviour of the simulator is similar enough to the real setup which means that the designed algorithms on a simulator (strategy and low level control) can be without modifications directly used also in real games. The simulator was tested in a number of European and World competitions in FIRA MiroSot league (real robots) category. There the game strategies used in real competitions were mostly developed by using the presented simulator.

6. Conclusion

The introduced simulator is mostly used as a tool in the process of strategy and control design for real robot soccer game. Therefore, its verification is done through transferability of the obtained strategy algorithms to the real system. The verification shows that the behaviour of the simulator is similar enough to the real setup, which means that the designed algorithms (strategy and low level control) can directly be used without modifications in real games as well.
The designed simulator has significant improvements in comparison with the available simulator in MiroSot leagues (simulator for SimuroSot) and other available simulators; the advantages being dynamics motion modelling and a realistic shape of the robots, which contributes to a more realistic simulation of robot ball interactions, collisions with robots, robots and boundary interactions and the situations where the ball is captured between two objects (it cannot invade any object). The presented simulator proved to be a good approximation of the real system. The motion models as well as collision models give realistic descriptions, which enable the simulator designed algorithms to be used on the real system.

7. Acknowledgment

The authors would like to acknowledge the Slovenian Research Agency under CRP MIR M2-0116 project for partly funding this work.

7. References

Baraf, D. (1997). An Introduction to Physically Based Modeling: Rigid Body Simulation II – Nonpenetration Constraints, In: *SIGGRAPH '97 Course note*s, Carnegie Mellon University.

Egeland, O. & Gravdahl , J. T. (2002). *Modeling and Simulation for Automatic Control*, Marine Cybernetics, Trondheim, Norway.

Ferber, J. (1999). *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*, Addison-Wesley, Essex, England.

Fremond, M. (1995). Rigid bodies collisions. *Physical Letters*, Vol. A, No. 1, 34-41.

Klančar, G.; Lepetič, M.; Karba, R. & Zupančič, B. (2003). Robot soccer collision modelling and validation in multi-agent simulator. *Mathematical and computer modelling of dynamical systems*, Vol. 9, No. 2, 137-150.

Klančar, G. (2007). Mobile Robot Simulator, available at: *http://msc.fe.uni-lj.si/PublicWWW/Klancar/RobotSimulator.html*.

Kolmanovsky, I. & McClamroch, N. H. (1995). Developments in Nonholonomic Control Problems, *IEEE Control Systems*, Vol. 15, 20-36.

Larsen, E. (2001). A Robot Soccer Simulator: A Case Study for Rigid Body Contact, *Sony Computer Entertainment America R&D*, March 2001.

Liang, T. C. & Liu, J.S. (2002). A Distributed Mobile Robot Simulator and a Ball Passing Strategy, *Technical Report TR-IIS-02-007*, Institute of Information Science, Academia Sinica, Nankang, Taiwan.

Moss, S. & Davidsson, P. (2002). *Multi-Agent-Based Simulation*, Springer-Verlag, New York.

The Math Works, Inc., Simulink (1998). *Dynamic System Simulation for Matlab*, Natick, USA.

Oriolo, G.; Luca, A. & Vandittelli, M. (2002). WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation. *IEEE Transactions on Control Systems Technology*, Vol. 10, No. 6, 835-852.

Sarkar, N.; Yun, X. & Kumar, V. (1994). Control of mechanical systems with rolling constraints: Application to dynamic control of mobile robot. *The International Journal of Robotic Research*, Vol. 13, No. 1, 55-69.

Stone, P. & Veloso, M. (2000). Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, Vol. 8, 345-383.

Welles, D. A. (1967). *Lagrangian Dynamics*, Schaum's Outline Series, McGraw Hill Book Company.

8

A Robust and Scalable Pareto Optimal Ball Passing Algorithm for the Robotic Soccer

Vadim Kyrylov
Rogers State University
USA

Abstract

We are looking for a generic solution for the optimized ball passing problem in the robotic soccer which is applicable to different RoboCup leagues and other digital simulated sports games like basketball or ice hockey. In doing so, we show that previously published ball passing methods do not properly balance the anticipated rewards, costs, and risks. The multi-criteria nature of this optimization problem requires using the Pareto optimality approach. As the problem itself is substantially inconvex, nothing else except the search of all available alternatives in the Pareto set appears to be applicable in this case. Real-time constraints are further complicating the problem. We propose a scalable and robust solution for decision making with multiple optimality criteria; its quality degrades in a graceful way once the real time constrains are kicking in. Our method is treating equally direct and leading passes to the partners and self passing while fast dribbling the ball by the player. The new method also allows easily modeling the whole spectrum of risk aversive to risk talking attitudes; therefore it is generic indeed.

1. Introduction

1.1 Ball Passing Algorithms: State of the Art

In the real-life and robotic soccer the rational player who controls the ball has four choices: shooting at the opponent goal, dribbling (moving without losing control of the ball), holding the ball, and passing it to a teammate or to self.

We are assuming that at given moment of time the choice between these higher-level options has been already made and the player is going to pass the ball by making the further choice between lower-level options representing alternative ways to execute this pass. As this decision making skill is generally regarded to be critical for the success of a soccer team, we investigate how ball passing could be implemented in the best possible way.

Early RoboCup scholars have identified the major peculiarities of ball passing and developed algorithms for the simulated soccer [1, 2]. These methods have proved to be good, as the soccer teams using them have been performing in the RoboCup competitions very successfully. Because we have not found in the academic literature any other published meth-

ods since then, this is presumably the state of the art; thus in what follows, we will be using these two algorithms as the prototypes for further improvement.
In both algorithms the soccer agent chooses values of the direction of the kick and its force. In [1] the anticipated outcome of this action is evaluated using two heuristic indicators: (1) the tactical value of the end point where the ball can be intercepted by the fastest teammate and (2) the fuzzy estimate of the likelihood of this interception. The latter is the function of the time balance between the fastest teammate to the ball and the fastest opponent. Stone & McAllister [1] also proposed using both direct passes to the receiver's position and leading passes that allow sending the ball to the teammate who is blocked by opponents from receiving a direct pass. In this method, the tactical value of the interception point is the only criterion for selecting the best option; the likelihood of success is used as a constraint for eliminating poor alternatives. Although this method has proved to be good enough for ATT-CMUnited-2000 making it to the third place in the RoboCup 2000 competition in the simulation league, it neglects some risks that are indeed present in the soccer game. One of them is the possible proximity of other opponents to the anticipated interception point. We would also disagree with [1] in that the ball should be always intercepted by the receiver in the minimal time. This may result in lost opportunities in executing leading passes when the ball is deliberately sent to the point of the field having the best tactical value. Thus the receiving player must be moving exactly towards this point if it is indeed safe to do so; the ball passing algorithm should be based on this assumption.
The ball passing decision making algorithm implemented in FC Portugal, the 2000 World champion [2], in many respects is similar to [1]. Additionally it is taking into account the opponent player congestion in the vicinity of the ball destination. Several additional factors are considered, such as ball travel distance, opponent goal scoring opportunity if the pass is successful, and the possible outcomes if the ball would not be intercepted as intended. The decision is made by deliberating on 5 options for each receiving teammate: direct pass, leading pass, pass to the expected location of the teammate, pass to a point near teammate having low congestion, and pass along a low congestion line. Each alternative is evaluated using 9 performance indicators. With the purpose of making a choice, these indicators are analyzed using a set of heuristic logical rules, i.e. a sort of a decision tree. It is known that FC Portugal outplayed ATT-CMUnited-2000 in the 2000 RoboCup competitions. We can only speculate whether the improved ball passing algorithm was a decisive factor or not. It is clear that the second algorithm must be good; yet it is much more complex and more difficult to analyze and optimize.
A presumably even more advanced ball passing algorithm incorporating leading passes with player collaborating using inter-player communication was recently reported in a very short paper [3]. This algorithm is all based on a decision tree. In what follows, we demonstrate that a decision tree may overlook some indeed good ball passing options, which is the shortcoming of this algorithm.

1.2 Unresolved Issues and Research Objectives

We would mention three issues that we address in this paper.

1. No benchmark so far. Although known algorithms have proved to be good indeed, from the scholarly standpoint their common weakness is in that they are just collections of sophisticated heuristics; it is still unknown to what extent they could be further improved and what the benchmark solution is.

2. Smoothly balancing rewards and risk. We believe that the ability to easy implement a continuous spectrum of risk-taking vs. risk aversive strategies by the soccer agent is highly desirable feature in any algorithm for modeling player behavior. Indeed, while risky passes should be completely avoided near own goal, they are affordable or even could be desirable on the opponent side. However, in the existing ball passing methods the deliberations with regards to risks and rewards, if any, do not render themselves as a controlled feature.
3. Avoiding possible conflicts with the real-time constraints, as the ball passing algorithms are computation intensive. So far the RoboCup scholars have been tackling with this inventively, yet somewhat unsystematically, by reducing computations to the reasonable minimum just by using various heuristics. Further reduction in the amount of required computations in such algorithms may be normally possible at the expense of abrupt loss in the quality of decisions, as this is done, for example, by removing some branches in the decision tree.

We believe that these issues can be resolved and it is indeed possible using the Decision Science techniques, such as multi-criteria decision analysis (MCDA). In doing so, we are pursuing the following objectives.

Developing a theoretical framework for a truly optimal ball passing algorithm that is utilizing all the potential of the soccer agent and could serve as a benchmark. We want this solution to be generic and thus reusable in any robotic soccer league. This intention is standing in a concert with other RoboCup scholars looking for generic solutions [4].

Fully identifying rewards, risks, and costs involved in passing the ball and demonstrating how they could be balanced in the proposed framework. In particular, we wish to offer an easy way to implementing a continuous spectrum of attitudes by the soccer player. We also want that decisions about ball passes to stem from the deliberation of all reasonable options that are available to the moment with very few parameters controlling the balance between risk taking and risk aversion. These parameters must all have clear meaning.

All-in-one implementation. We would like to have a generic solution that subsumed direct and leading passes and passes to self. With regards to leading passes, too few details have been revealed in both [1] and [2] so far on how the end point is determined, although this is a non-trivial task. We wish to propose a simple method for implementing this critical feature.

Addressing the real-time constraints. We want to propose a truly scalable solution with just one parameter which determines the amount of the required computations. We also want to design a robust ball passing algorithm so that if we are forced to eliminate some computations for the sake of saving computation time, that would be resulting only in a gradual loss of the decision quality.

Section 2 provides the reward/risk/cost analysis of ball passing. In Section 3 we briefly explain the essence of the Pareto optimality and demonstrate how this concept applies to optimized ball passing. Section 4 provides a solution to searching the Pareto set for the optimal decision. Section 5 explains how the real-time constraints in the proposed method could be walked around in a graceful way. Section 6 concludes this work.

2. Rewards, Risks, and Costs in Ball Passing

Prior to developing the optimal decision making algorithm, in this section we identify the criteria that govern the decision by the soccer player to pass the ball; we wish this list to be as complete as possible.

In doing so, we slightly modify the ball passing problem formulation as compared to [1, 2]. In our case, player with the ball considers all possible points (x, y) in the field and must decide to which point he should send the ball now; he also must choose the speed that the ball will have upon the arrival in this point. The end speed affects the probability of the successful interception by the receiving teammate; it also determines the ball travel time and thus the incurred risk. The decision is made by comparing these and some other criteria calculated for different ball passing options.

Once the passing player has made his choice of the point and of the ball end speed, he is able to determine the kicking force and direction, which are the actual decision variables. If the required kicking force exceeds the available limit, the point is just removed from the consideration. Likewise, points are eliminated if the perceived risks are prohibitively high. This feature could be implemented similar to the existing algorithms; conservative elimination rules can be applied to avoid unnecessary deliberations and reduce computations. In what follows, we demonstrate this on examples.

Each remaining potential destination point for pass is assigned a vector criterion having continuous values of its m components. So there is a two-dimensional decision space (*kicking_force, direction*) and an m-dimensional criterion space.

For the analysis, we replace the continuous decision space by a discrete one. Initially, with the purpose to find a benchmark solution, we deliberately consider that the number of points in this space is large enough so that in terms of our criteria the difference between the neighbors was negligibly small.

For the further analysis of the decision criteria semantics, we split them in three categories: (1) rewards, (2) risks, and (3) costs.

Rewards. We see two rewards, or gains, from passing the ball; we wish to maximize both.

First can be measured by the tactical value $f_1(x,y)$ of the point where the ball will be sent to. As the tactical value proposed in [1] and [3] serves this purpose very well, we will be using a similar one. It encourages sending the ball up the field to the opponent side; points closer to the opponent goal have higher tactical value. Because of the function is anti-symmetrical for x-coordinate, it equally discourages from passing the ball close to own goal.

The second reward function reflects the chance to score the opponent goal once the ball arrives to its destination (x, y). This could be measured by the following heuristic criterion:

$$f_2(x, y) = \alpha(x, y)/(n_{opp}(x, y) + 1), \quad (1)$$

where $\alpha(x,y)$ is the angular size of the goal as seen from (x, y) and $n_{opp}(x,y)$ is the anticipated number of opponent players in the cone having (x, y) as its vertex; the base of this cone is the opponent goal that is slightly stretched sideways.

Risks. The risks involved in ball passing can be all regarded as soft constraints because of presence of uncertainty; this is a further improvement of the idea proposed in [1]. Risks were identified bearing in mind that the receiving player is intercepting the ball in minimal time if only he has no other choice; rather we assume that he must be moving to the point where the sender decided to get it delivered to. (This could be implemented using communication and/or so-called 'smart' ball interception algorithm.) Hence we need to address more risk factors than our predecessors. If the risk incurs any time balance, we calculate it

with some conservative margin in the favor of the opponents. This margin increases with the distance to the ball passing end point.

1. Opponent may reach (x,y) before the teammate. The risk function $r_1(x,y)$ is the time balance between the arrivals of the fastest opponent and the fastest teammate to this point. If this balance is negative, $r_1(x,y)$ is zero.
2. Ball can be intercepted by the opponent on its way to (x,y). The risk function $r_2(x,y)$ is the time balance between the intended arrival time of the ball in (x,y) and the earliest time when it can be stolen by the opponent. If this balance is negative, $r_2(x,y)$ is zero.
3. Teammate may be too late in point (x,y) after the ball rolls by. So the risk function $r_3(x,y)$ is the probability of the teammate failing to reach the ball in (x,y). This probability is 1 if the teammate cannot arrive in (x,y) before the ball and decreases fast to 0 with the extra time available for the player. The risk also increases if the ball is moving in (x,y) too fast which is making it difficult to intercept.
4. Once the ball has been received by the teammate, too many opponents may get close by. The risk function $r_4(x,y)$ is the time balance between the arrivals of the first and second fastest opponents in (x,y), plus some positive heuristic constant $r_{4\max}$. If this sum is negative, $r_4(x,y)$ is zero. If both opponents arrive simultaneously, $r_4(x,y)=r_{4\max}$.
5. If the teammate fails to intercept the ball, it may cross the field boundary; the interruption of the game by the referee that would follow thus giving some advantage to the opponent. Similar case is when passing the ball may create the offside situation. The risk function $r_5(x,y)$ is zero everywhere except the points where the ball trajectory, if continued, may intersect the field boundary or the offside line. In these cases the risk is some constant $r_{5\max}$ less the time remained until the ball crosses the line. Thus on the field boundaries we get $r_5(x,y)= r_{5\max}$. Negative values are brought up to zero.
6. The receiving player may have too low stamina to chase and handle the ball (this information is made available to the ball controller via aural sensor). The risk function $r_6(x,y)$ is the time when the receiving teammate reported low stamina minus current time plus positive heuristic constant $r_{6\max}$. If this sum is negative, $r_6(x,y)$ is zero.
7. Ball may not reach the destination point at all, as (x, y) is too far away for given initial ball speed. As the ball movement is distorted by noise, the actual maximal ball traveling distance may differ from the calculated theoretical one, $D_{\max}$. A soft constraint $r_7(x,y)$ is used to reflect this risk. For points whose distance is significantly less than $D_{\max}$, $r_7(x,y)$ is zero; near $D_{\max}$ values of $r_7(x,y)$ are positive.

For convenience, the seven risk functions described above might be scaled so that they all are taking values in, say (0, 10). We wish to simultaneously minimize each of them.

Costs. The obvious cost factor is the time required for obtaining the anticipated rewards, which we want to minimize. Taking this in consideration makes sense because the precision of the situation prediction decreases with the forecast time substantially. In this respect, it is just another risk factor. Yet cost is also may be an objective, as we are obviously interested in achieving the tactical gain faster. Cost could be measured by the nonlinear function $c_1(x,y)$ whose value is zero for short passes and grows with the time needed the ball to reach (x,y). This criterion would be discouraging too long passes if, given all the rest conditions equal, shorter ones exist.

Discussion. We believe that this is the most complete list of factors affecting ball passing, as it contains new elements. Heuristic constants are used for scaling only; thus they are not the 'magic numbers' that are omnipresent in existing ball passing algorithms. Some factors have been taken into account by previous scholars, yet none of them has provided a comprehensive list in academic publications. The identified risks have different severity and likelihood to occur. Of them, risk factor 1, is probably best known to all RoboCup scholars, as it is incurred very often. On the other hand, factor 7, although well understood, has been presumably treated in [1, 2] just as a hard constraint. However, a soft constraint is more applicable in the general case as the ball movement noise may be significant in the RoboCup leagues dealing with physical robots. As in [1, 2] the interception of the ball only in minimal time was assumed, risk factor 2 was not considered. By deliberately distinguishing between risks 1 and 2, we thus propose more general approach. Risk factor 6 has been also left unnoticed in the RoboCup literature; in some situations, however, this may result in the complete failure to pass the ball. Also new is the cost consideration; it looks like nobody before has tried to consider the cost as a factor. Yet the most significant difference of our approach from [1, 2] is in that we are not applying any logical rules to the performance indicators, nor do we suggest merging them in one.

Concept demonstration. With the sole purpose of the concept demonstration used throughout this paper, we have designed an example with three simplifications. (1) Decision space is further reduced to determining the direction of pass only; end speed in the destination point (x, y) is a fixed parameter of the algorithm. (2) Only the tactical value of the end point is used as the reward. (3) Risk and cost factors merged in just one parameter by applying heuristic rules.

This allowed using two-dimensional displays for the visualization. The full-blown algorithm is treating all criteria separately without merging those using heuristics.

Screenshots in Fig.1 illustrate application of different constraints to a grid of 3600 points considered as candidates for passing the ball by player 11 from the right-hand team. The eliminated points are shown in light gray; the darker points are the remaining alternatives. The player must select the best from them based on the vector of performance indicators available for each point.

We wish this decision to be optimal in some sense. This sense is the Pareto optimality.

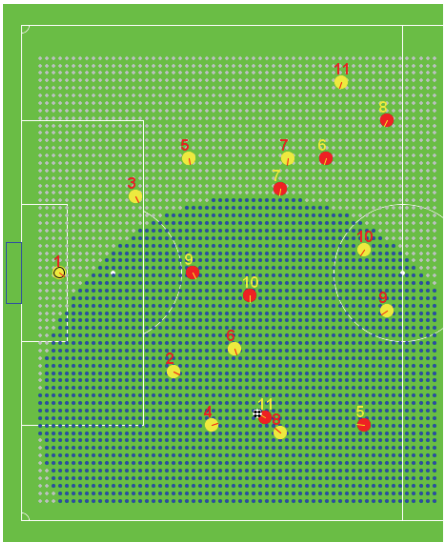
3 Applying the Pareto Optimality Principle to Ball Passing

Pareto optimality, first originated in economics, is now a standard principle for solving vector optimization problems with conflicting criteria [5, 6]. The criteria identified in Section 2 are indeed conflicting, as the higher rewards are normally coming at higher risk and cost. In what follows, for the purpose of illustration, we will replace the reward function with the negation thereof; thus we want all our criteria to be minimized simultaneously.

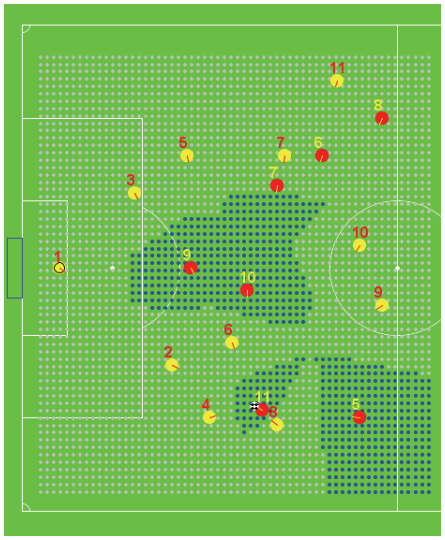
In the general case, though, simultaneous minimization cannot be achieved. The Pareto optimality principle only offers a method for substantially reducing the set of decision alternatives by identifying among them the set of so-called non-dominated alternatives; altogether they are making the Pareto set, or the Pareto frontier [5, 6].

By definition, the criteria vector $\mathbf{v}_i = \{v_{i1},\ldots,v_{im}\}$ is **dominated** by vector $\mathbf{v}_j = \{v_{j1},\ldots,v_{jm}\}$ if the following condition holds:

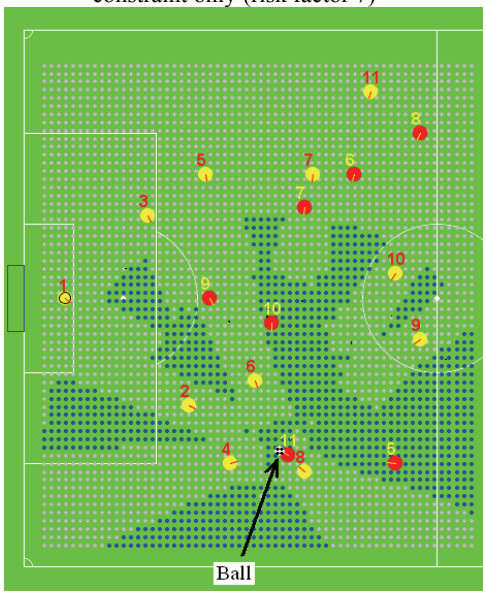
$$\forall k\{v_{ik} > v_{jk}\}. \tag{2}$$



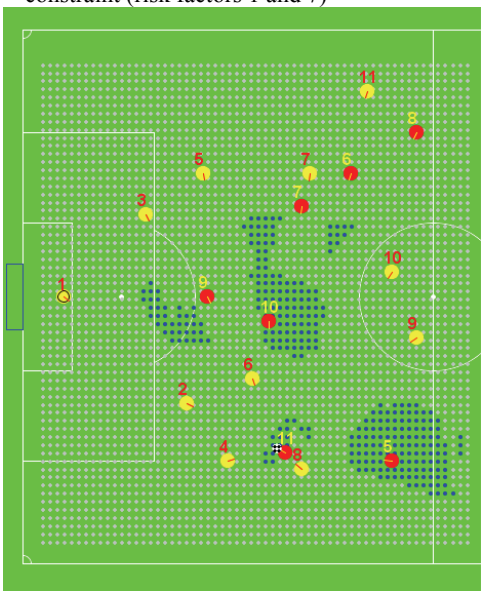
(a) applied the maximal ball rolling distance constraint only (risk factor 7)



(b) added the opponent/teammate time balance constraint (risk factors 1 and 7)



(c) added the early interception of the ball by the opponent (risk factors 2 and 7)



(d) all risk constraints applied; the remaining points are the player decision making options

Fig.1. Screenshots of the software tool developed for analyzing the soccer player tactics. Of the original 3600 points, some have been eliminated by applying the ball distance constraint first. Individual captions explain additional constraints that have been applied then

This means that $\mathbf{v}_j$ is located inside the cone in R^m with the vertex $\mathbf{v}_i$, the sides of this cone being parallel to the coordinate subspaces R^{m-1}. Fig.2 illustrates this situation for m=2.
By definition, the Pareto set is the subset of the alternatives which are all non-dominated. An example is shown in Fig. 3. Note that the Pareto set is not necessarily convex, nor is it in the general case even connected. The computational complexity of determining the Pareto subset in the finite set with N elements is $O(N^2)$.

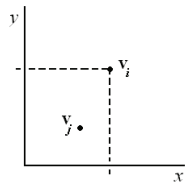


Fig.2. Vector $\mathbf{v}_j$ dominates $\mathbf{v}_i$

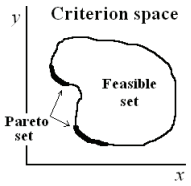


Fig.3. The Pareto set contains all non-dominated points

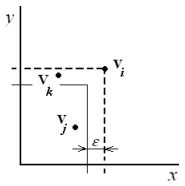


Fig.4. While $\mathbf{v}_j$ both dominates and ε-dominates $\mathbf{v}_i$, $\mathbf{v}_k$ dominates $\mathbf{v}_i$ only plainly

In Section 4, we will be also using a weakened version of the dominance relation, which is called ε-domination (Fig.4). The set of non-ε-dominated points is referred to as ε-Pareto set.
The common-sense meaning of a non-dominated alternative $\mathbf{v}_j$ is that outside the Pareto set there is no another alternative that outperforms $\mathbf{v}_j$ simultaneously by all criteria; at least one criterion value is worse, anyway. From this follows that the optimal decision should be sought within the Pareto set; all the rest alternatives could be just eliminated as they are all inferior.
Noteworthy that, eliminating decision alternatives before identifying the Pareto set, as it has been done before in the ball passing algorithms, may result in that some of the Pareto optimal points would be apparently removed without even evaluating thereof. This is exactly what may happen in decision trees. Unless the tree decision conditions are designed so carefully that any eliminations do not affect the Pareto set, there is no guarantee that the decision making algorithm yields optimal solution to the problem in all cases. However, the trouble is in that such a decision tree is difficult to design, and for each new applied problem this must be done over and over again. On the other hand, the Pareto optimality principle offers a general solution.
So in ball passing we follow this principle. The Pareto set of the alternatives that player 11 in Fig.1(d) should be indeed choosing from is shown in Fig 5. As all the rest alternatives are dominated, we should eliminate them. This example suggests that either the leading pass to player 9 should be executed (five slightly different options), or player 11 should leave the ball for himself (two options). Passes to player 10 are not in the Pareto set. Fig. 6 shows the situation as it occurs in the criterion space. It is clearly seen that the Pareto set is non-convex and disconnected. Disjoint parts of the Pareto set correspond to different teammates who can receive the pass; this is the result of that the criteria are substantially non-monotonic, multi-modal functions.

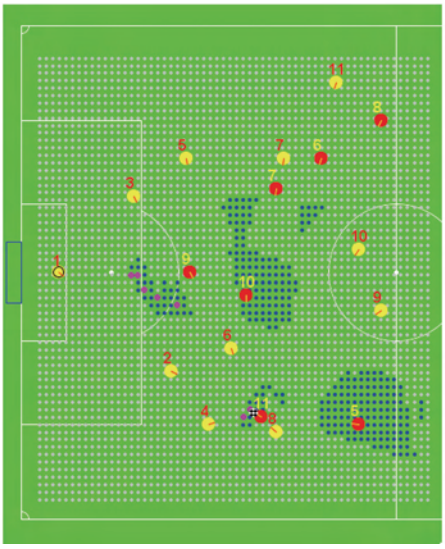


Fig. 5. Situation in Fig.1(d) with the points making the Pareto set highlighted

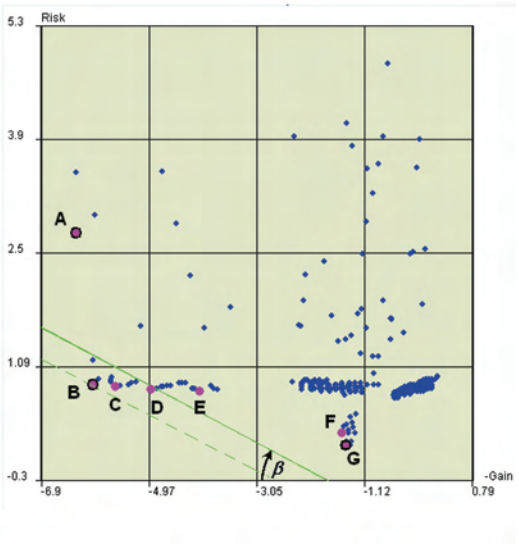


Fig.6. The ball passing alternatives in the criterion space. Points in the Pareto set have larger size

The MCDA theory leaves the final choice of the single alternative from the Pareto set up to the decision maker. In doing so, the latter should rely on his/her aspirations, such as risk taking and risk aversion. In our case, however, the decision maker is the artificial agent; it is the algorithm developer who must formalize the player preferences which could be used for searching the Pareto set for the only 'truly optimal' alternative. This search is exactly about balancing the rewards, risks, and costs; in what follows, we explain the idea of balancing.

A naïve approach suggests merging the criteria in just one and applying commonly known single-criterion optimization techniques. For example, one can use the utility function U of the decision variables (x,y), which is a weighted sum of risk *Risk* and gain *Gain*:

$$U(x,y) = -w \cdot Gain(x,y) + (1-w) \cdot Risk(x,y), \quad (3)$$

where w is the positive weight, $0 \le w \le 1$; it reflects the importance of *Gain* for the decision maker, as compared to *Risk* whose weight is thus 1-w. (Note the minus sign indicating that we are using $-Gain(x,y)$ as criterion).

To find the solution, the utility function (3) must be minimized. Equation $U(x,y) = c$, where c is some constant, in the criterion space represents the slant straight line shown in Fig.6. Search for the optimal solution in this case would be moving this line towards the origin by decrementing c until the line (shown in the dashed style) intersects with just one decision alternative **B**. Presumably, this would be the optimal, balanced solution sought.

Unfortunately, this simple approach does work only when the Pareto set is convex [6]. If non-convexity is in place, some elements of the Pareto set would be never rendered as the solutions to the optimization problem, no matter what values the decision maker assigns to w. However, this is counter intuitive, as each point in the Pareto set is the best option for some combination of the decision maker preferences. In our example we can scan all possible preferences by making the weight w taking all possible values in the range $0 \le w \le 1$. Note

that, as β =tan(w), this parameter determines the angle β of the line $U(x,y) = c$ (in Fig.6, w=0.335). With w changing, the line is inclined in all the range of angles from 0 to 90 degrees. This 'optimization' process would render only three points, **A**, **B**, and **G** of total seven available in the Pareto set (marked with black circles). The rest four would be never returned as solutions, though, in spite of that the weight is continuously running in the whole permissible range and the missed points are lying in between.
Other known methods based on merging criteria in a single one would be also unsuccessful in the general case if the Pareto set is non-convex. This just illustrates the fact that with the multi-modal criteria functions which we are dealing with in the robotic soccer, a different way to finding the balanced solution of the optimization problem should be taken.

4. Searching the Optimal Ball Passing Decision in the Pareto Set

The different way is applying more sophisticated methods for searching the Pareto set that can work with non-convex problems. As there is a plethora of such methods, we will demonstrate just one, developed by the author of this paper. The method is called '*the sequential elimination of the poorest alternative*'. Because it does not rely on any information about the criteria functions, it is applicable to any MCDA problem with a finite Pareto set. This nicety, comes at rather low cost: with the total of K elements in the Pareto set, the computational complexity of this algorithm is $O(K^2)$. (Note that $K<<N$, where N is the number of points in the set of the alternatives before any eliminations.)
The key assumption is that each criterion has its relative weight; in our case this information is reflecting the preferences of the developer of the decision making algorithm. So let **X** be the set of all alternatives, $P \subset \mathbf{X}$ be the Pareto set, $\mathbf{x} \in \mathbf{X}$ be a decision vector, $g_1(\mathbf{x}),\ldots \mathbf{g}_n(\mathbf{x})$ be the criteria functions (all of which we want to simultaneously minimize), and $w_1,\ldots,w_n$ be the non-negative weights whose sum is 1. The algorithm is shown in Fig.7.

```
S := P;
for ( k := 1 to K-1 )
{
  With probability w_j, randomly select j-th criterion;
  Find the element x ∈ S having the maximal value of g_j(x);
  remove x from S;
}
return the last remaining element in S
```

Fig.7. The algorithm for searching the Pareto set for the single optimal solution

The algorithm has K-1 iterations, eliminating one element from the Pareto at a time. In each step, one criterion is randomly selected; let j be its index. Because weights are used as the probability distribution, more important criteria are being chosen more frequently than the less important ones. For j -th criterion, the alternative having the maximal (i.e. the poorest) value of $g_j(\mathbf{x})$ is removed from the working copy of the Pareto set **S**. The process ends when only one element remains in the working copy. This is the approximation of the balanced, optimal solution to the problem. With K increasing, this approximation converges to the precise optimum.

A simplified version of this algorithm with equal weights does not even require any randomization; criteria for the element removal can be selected in turns. If applied to the example in Fig.6, this simplified method will be first using the *-Gain* criterion to remove point **G**. On the next iteration point **A** will be removed using *Risk*. Next **F** will be removed using *-Gain*, and so on. The sole surviving alternative in this case will be **D**, which is the solution.
The too scarce discrete subset of the continuous Pareto set like shown in Fig.6 appears to be too rough approximation. Recall that this is all what has remained from the original 3600 points; so if we want a denser approximation of the Pareto frontier, we should be starting with even larger number. Because further increasing this number is not an option, we are using the ε-dominance relation instead of the strict one (Fig.4). This concession can be justified by that the criteria values are calculated with some errors anyway. As we can guesstimate the standard deviation of these errors, we can choose ε of the same order of magnitude. Thus the discrete ε-Pareto set will be much denser; the application of the random elimination in this case would result in much smaller volatility of the solution.
Fig.8 and 9 give the idea of what happens to the situation in Fig.5 and 6 once ε-dominance is applied; the player indeed gets much more options to chose from. The cost for this is slight deviation from the strict Pareto optimality and a longer, yet not prohibitive, computation time. The benefit is the better robustness of the solution search algorithm.

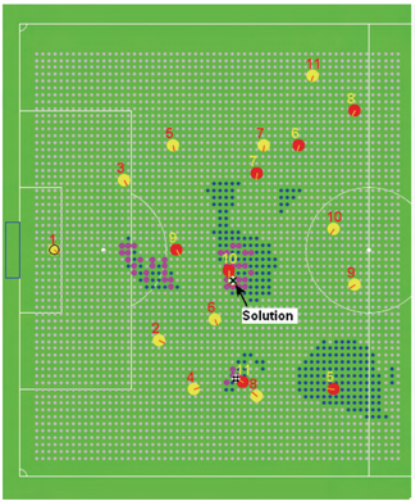


Fig.8. Situation with the ε-Pareto set. Note the increased number of points. The solution is shown as a black cross. Player prefers avoiding risky passes

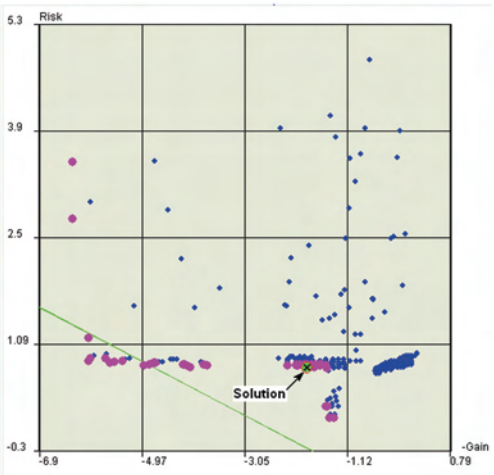


Fig.9. The ball passing alternatives in the criterion space. Points in the ε-Pareto set have larger size. The solution is shown as a black cross. The *-Gain* weight is 0.335

Assigning weights to the criteria in the proposed framework has transparent meaning. Unlike using weights to sum up criteria similar to (3), in our method there is no way for that a high value of one criterion apparently compensates for the insufficient value of the other. This is because weights determine only the priorities in the elimination of poor alternatives. So it is safe to say that the proposed technique allows easily modeling the full spectrum of

risk taking and risk aversive attitudes of the decision maker. This is made possible by assigning weights.
So far we have been using the example with the weight of -*Gain* 0.335, i.e. *Risk* had twice as much higher weight. This results in the risk-aversive decision shown in Fig. 8 and 9. Player 11 prefers to pass the ball to teammate 10 rather than taking the chance of sending the ball to teammate 9 whose position appears to be is much better. By changing the weight in favor of higher gains, it is possible to persuade the player to pass the ball to player 9 (see Fig. 10, 11). The slant straight lines shown in Fig. 9 and 11 indicate the weight w only; they are not used to find the solution, as different procedure applies.

5. Addressing the Real-Time Constraints

As described so far, the optimal ball passing decision making algorithm in terms of computations appears to be even more demanding than the algorithms proposed in [1, 2, and 3]. In the first experiments in 2003 with our simulated soccer team *SFUnleashed* we have indeed found that the quality of decisions made by players while passing the ball non-monotonically depended on the total number of points N. Starting with small number of points, quality was noticeably increasing with N. Then, with greater N, we observed significantly decreased performance. Indeed, with large N the player process could not complete all required computations during one simulation cycle.
As it should be expected for a real-time system like robotic soccer, attempts to utilize all the player potential by using sophisticated optimization may be counterproductive because of the prohibitive computation time. Still we decided to find a way out so that the real-time constraints were not so restrictive. Our solution comprises two ways for the time reduction.
The first way is further reducing the number of alternatives that wittingly are not in the Pareto set; this can be done by applying more heuristics. Besides those outlined in Fig.1, we are using one more, by replacing the equidistant grid with randomly scattered points in the vicinity of each teammate. In doing so, we want to ensure that the teammate can reach the given point before the ball arrives in it. This is achieved by determining the size of the area around the teammate; this area is then randomly populated by the points (Fig.10, 11).
The second way is automatically adjusting the number of generated points N during the run time with respect to the actually available time in the simulation cycle. As we know that the complexity of the whole method is $O(N^2)$, it is always possible to estimate affordable N in advance in the current simulation cycle and thus prevent real-time constraints from kicking in. Reducing N would result in only gradual increase of the random deviations from the optimal solutions, without any abrupt losses in the quality of decisions on the average. This behavior is quite different from that of the algorithms based on decision trees whose real-time scalability is very limited.
Thus the proposed algorithm is robust by design and it is indeed scalable with respect to tightened or relaxed real-time constraints.

6. Conclusion

The full-blown algorithm is just a straightforward generalization of the simplified method illustrated in the above examples. The only difference is that instead of the two criteria function we are using all ten. The algorithms for computing these criteria have been described in Section 2; some of them are similar to that can be found in the RoboCup literature. More

elaborated criteria set combining different rewards, risks, and costs allows coupling our method with the 'smart' receiving player behavior when the ball is intercepted in the point having best tactical value rather then just in the fastest way. One new criterion allows avoiding failed passes that are possible in the previous algorithm when the receiving player is exhausted. The proposed method is general, as it is equally treating direct and leading passes and passes to self.

Still the major new element of our ball passing algorithm is the application of the Pareto optimality principle and the related method for searching the Pareto set. This innovation guarantees that, for given set of the decision maker preferences, with the discrete approximation of the Pareto set dense enough, the solution of the ball passing problem yielded by this method cannot be further improved. Therefore, the proposed method can be used as a benchmark for evaluating different algorithms based on heuristics.

Our method provides a convenient way to implementing all the range of the decision maker attitudes by balancing gains, risks, and costs in a flexible way. Therefore, the new approach allows enriching the soccer team tactics without complicating the control logic.

The proposed algorithm is robust with respect to the real-time constraints. It allows avoiding the violation of these constraints by adjusting during the run time just one parameter, the total number of decision alternatives in the initial set. As this set is generated randomly, its size reduction would only result in a gradual random deviation from the optimal decision.

As the proposed approach to optimization is general, besides ball passing, we consider using it for optimizing other low-level decision making in the robotic soccer, such as player positioning, dribbling, and scoring the goal. We also expect that similar approach is applicable to higher-level decisions involving the choice between the lower-level actions.

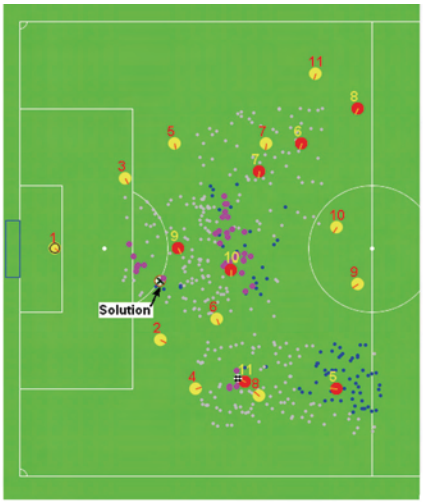


Fig. 10. Situation with 400 points randomly generated about the teammates. The decision is more risky than in Fig. 7., but its gain is higher

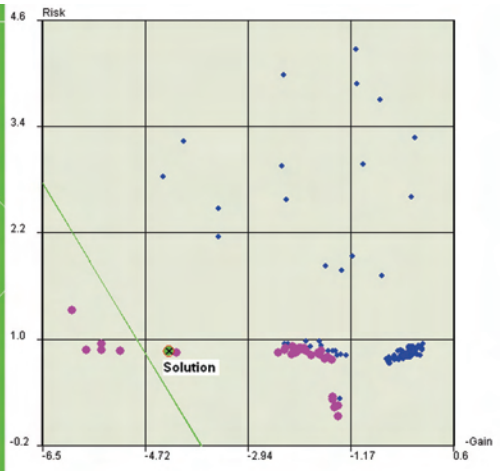


Fig. 11. The ball passing alternatives in the criterion space with ε -Pareto set highlighted. The *-Gain* weight is 0.614. Note the different angle of the line

7. References

1. Stone, P.; McAllester, D.: An Architecture for Action Selection in Robotic Soccer. In: Proceedings AGENTS'01, 5th International Conference on Autonomous Agents, May 28-June 1, 2001, Montreal, Quebec, Canada. (2001) 316-323
2. Reis, L. P.; Lau, N.: FC Portugal Team Description: RoboCup 2000 Simulation League Champion. In: Stone, P.; Balch, T., and Kraetzschmar, G. (eds.): RoboCup 2000, LNAI 2019. Springer: Berlin Heidelberg New York (2001) 29-40
3. Wang, C.; Chen, X.; Zhao, X. & Ju, S.: Design and Implementation of a General Decision-making Model in RoboCup Simulation, pp., International Journal of Advanced Robotic Systems, 1, No. 3 (2004) 207-212
4. Dylla, F.; Ferrein, A.; Lakemeyer, G. ; Murray, J.; Obst, O.; Rofer, T.; Stolzenburg, F.; Visser, U. and Wagner, T.: Towards a League-Independent Qualitative Soccer Theory for RoboCup. In: Nardi, D. et al. (eds.): RoboCup 2004, LNAI 3276. Springer: Berlin Heidelberg New York (2005) 29-40
5. Ehrgott, M.: Multicriteria Optimization. 2nd ed. Springer: Berlin Heidelberg New York (2005)
6. Jahn, J.: Vector Optimization. 2nd ed. Springer: Berlin Heidelberg New York (2004)

9

FC Portugal - High-level Coordination Methodologies in Soccer Robotics

Nuno Lau and Luis Paulo Reis
University of Aveiro
Portugal
University of Porto
Portugal

Abstract

This chapter presents the research performed in the context of FC Portugal project in the areas of agent architectures, coordination methodologies, coaching and agent development tools. FC Portugal's research has been integrated in several teams that have participated with considerable success in distinct RoboCup leagues and competitions. The chapter includes a brief description of the main competitions in which FC Portugal has participated with focus in the simulation leagues and related challenges. It also presents some of the developed techniques and results achieved in controlled experiments. These results, together with the impressive record of results achieved by FC Portugal teams in RoboCup competitions show that the techniques developed can significantly improve any soccer robotics team performance.

1. Introduction

This chapter will be structured around the main contributions of the FC Portugal project in the field of Soccer Robotics.
Initially, the environment of the RoboCup Simulation League Competitions (2D and 3D Simulators, Physical Visualization and Nanogram), and also of the Middle-Size League will be presented, as they form the base where the contributions have been applied.
The main research goal of FC Portugal project is the development of a formal model for the concept of team strategy for a competition with an opponent team having opposite goals, general enough to be instantiated to various dynamic competitive domains. The formal model enables the design of an agent architecture suitable for RoboCup simulation league agents and a world state model capable of storing the information needed for an intelligent agent to play soccer. It has been applied in Simulation (2D, 3D, Coaching and Physical Visualization) and Real Robot RoboCup Soccer Leagues (middle-size, small-size and legged leagues) allowing for a flexible and structured strategy instantiation. It has also been applied to other domains such as the RoboCup Rescue.

The project research focus is also concerned with developing general decision-making and cooperation models for soccer playing. Cooperation mechanisms include the Situation Based Strategic Positioning and Dynamic Positioning and Role Exchange mechanisms.
Situation Based Strategic Positioning (SBSP) mechanism is used to calculate the strategic positionings of all agents in the team according to the game situation. The agent autonomously calculates its base strategic position, adjusting it according to the ball position and velocity, situation and player type strategic information. This mechanism enables the team to move similarly to a real soccer team, covering the ball while remaining distributed along the field allowing for a cooperative positioning among autonomous agents.
The Dynamic Positioning and Role Exchange (DPRE) enables players to exchange their positionings and player types in the current formation if the utility of that exchange is positive for the team. Positioning exchange utilities are calculated using the distances from the player's present positions to their strategic positions and the importance of their positionings in the formation on that situation.
Communication languages and protocols, to convey the most relevant information at the right times to players were also developed. Research was also focused on intelligent control of players' sensors to achieve maximum coordination and world state accuracy.
Coaching is a very important research topic in Soccer Robotics. Coach Unilang – a general language to coach a (robo)soccer team was developed to enable high-level communication between a coach agent (or human coach) and a soccer robotics playing team. Our FC Portugal coach conveys strategic information to players, while keeping their individual decision autonomy, by using this language.
FC Portugal is also very concerned with the development of agent evaluation and monitoring tools like our offline client methodology, that permits the reiteration of the execution of the agent without real-time constraints allowing a precise insight into the agent's reasoning; WstateMetrics, that evaluates the accuracy of world states, and is used to assess the development of the intelligent sensors and communication protocols capabilities; and Visual debugger used to graphically, and in a very intuitive way analyze the reasoning of all the agents in the team in a integrated manner.
While most of these techniques have been developed in the context of simulated soccer robotics, some are already applied in real robot teams and other domains.
The chapter presents the evaluation of the developed techniques performed in controlled experiments and also in real competitions where teams that resulted from the presented research have participated.
The rest of the chapter is structured as follows. Section 2 presents the RoboCup initiative focusing in the rules of the competitions where FC Portugal research has been applied and tested. Section 3 is focused on some of the aspects of the architecture of FC Portugal agents used in most of our teams. In section 4 the developed coordination methodologies are presented while section 5 presents and gives directions into our coaching research. Section 6 is devoted to the principles and implementation of our multi-agent development tools and section 7 presents the results of some controlled experiments that were performed to test the coordination methodologies. Finally, section 8 draws the conclusions.

2. RoboCup International Initiative

RoboCup is an international initiative that aims to motivate the research on multi agent systems and intelligent robotics (Kitano, H., et al. 1997). Every year RoboCup organises scientific meetings and world robotic competitions in the fields of robotic soccer and robotic search and rescuing. Some competitions use simulated environments while others use real robots.
In the real robots competitions there are several leagues including the middle-size, small size, four-legged (based on AIBOs from Sony) and humanoid soccer leagues. These competitions, besides having different rules concerning robots and field dimensions differ in autonomy of the robots and robot construction details. While in the middle-size and legged league, robots are autonomous and sensors are mounted on the robots, in the small-size league a single agent may decide and send commands to every robot of a team. The agent of the small size league typically receives information from a camera positioned above the field to detect the robots and ball positions. The middle-size league will be explained in more detail, as some of the coordination methodologies developed by FC Portugal are currently being transferred to this league. In simulated environments there are several competitions that are detailed in the following sections.

2.1 Middle-Size League

In the middle size league two teams of at most 6 real autonomous robots play soccer in a 18x12m field. Robots height is limited at 80cm, the radius is limited at 50cm and the weight is limited at 40kg. Robots are completely autonomous, although they are allowed to communicate each other, and all sensors must be mounted on the robots. The environment is color marked i.e. the field is green, the lines are white, goals are painted in blue and yellow and the ball is orange. Robots must be black except for the markers of each team that must be cyan and magenta.
The mechanical and electrical/electronics solutions found to build the robots play a very important role in the final efficiency to play soccer. Also the vision subsystem is critical for the final performance of the robots. Games are very active and interesting and the top teams exhibit some very interesting coordinated behaviour.

2.2. Simulation 2D League

RoboCup Simulation League is one of the 3 leagues that started the RoboCup official competitions in 1997. In fact a demonstration of the soccer simulator used in this competition had already been done during the pre-RoboCup 2006. The view of the RoboCup Organizers is to concentrate the research in this league at the top-level modules of the soccer robotics problems: the high-level decision and the coordination of teams of, possibly heterogeneous, robots. Over the years the 2D simulator has evolved, including new features and tuning some others, but the core architecture of the simulator is the same as the one used in 1997.
In the 2D simulation league a simulator, called soccerserver (Chen et al., 2007), creates a 2D virtual soccer field and the virtual players, modelled as circles (Fig. 1). The simulator implements the movement, stamina, kicking and refereeing models of the virtual world. The models in the simulator are taken from real robots (like the differential drive steering) and from human-like characteristics (like the stamina model).

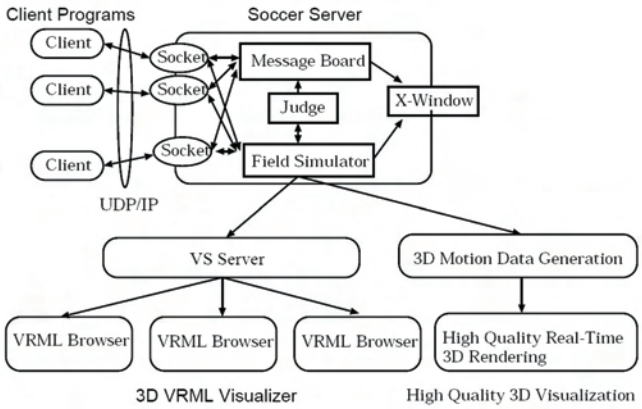


Fig. 1. 2D simulator architecture from (Kitano, H., et al. 1997)

Teams must build the software agents that control each of the 11 virtual robots and also a coach agent. The control of the agents is performed by sending commands to the simulator. The main commands are *dash(dpower)*, *turn(tpower)*, *kick(kpower,kangle)*, *tackle(tangle)* and *catch(cangle)* (used only by the goalie). The simulator implements several virtual sensors for each robot and sends the measures of these sensors to the robots periodically. The most important sensor is the vision sensor, but there's also a sense body sensor (that informs the player of its stamina and own speed) and a hearing sensor. Sensory data is in general subject to noise or to some other type of pre-processing that precludes the agents from knowing the exact value of the measures.

The simulation advances in steps of 100ms, meaning that every 100ms the positions and velocities of every player and of the ball are updated by the simulator. Some of the sensory data (like some modes of the vision sensor) is sent to the agents with a different period than that of the simulation update.

Each agent controls only one virtual robot and must coordinate its efforts to make its best contribution for the teams' goals. It is important to note that the knowledge of the various agents about what is happening at a certain moment is not identical, due to noise and restrictions on several sensors (like the angle of vision or the cut-off hearing distance). Also the environment is very dynamic with the opposite team controlling their robots to oppose the teams' goals.

The coach agent is a special kind of agent that receives, from the simulator, the positions of all players in the field and of the ball without noise. However, the coach as severe limitations on its communication with field agents, that make it impossible to control the field robots using the coach information. The coach may have a very significant impact on team performance by giving advice to the field players and using it to perform high-level tasks like tactic analysis and selection or opponent modelling.

Visualization of the games is assured by an independent application that communicates to the simulator to receive the players and ball positions. Fig. 2 and Fig. 3 show two possible visualizations of the games in the 2D simulation league. All the 3D features of the viewer in Fig. 3 are not modelled in the simulator but inferred by the viewer for better attraction.

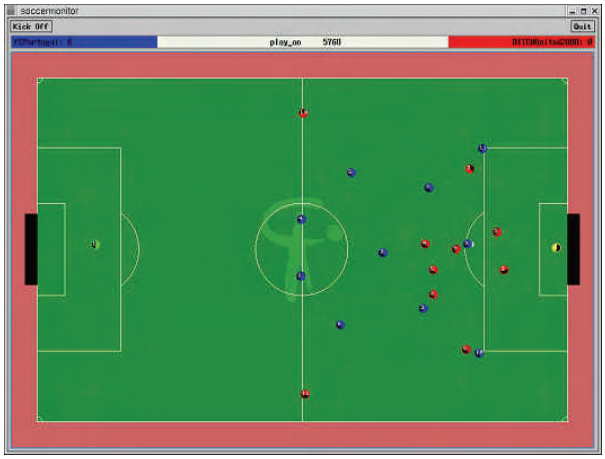


Fig. 2. 2D Simulation League Traditional Viewer

Fig. 3. 2D Simulation League Viewer with 3D displaying capabilities (Sedaghat M. & al. 2003)

2.3. Simulation 3D League

The first version of the 3D simulation league simulator was made available to the RoboCup community during January 2004. The proposal of the 3D simulator had the following objectives:

- Replace the 2D environment of previous simulator with a 3D environment;
- New, more realistic, physics model;
- Simulation results should not be dependent on available computational power or on the quality of network resources.

The differences between the new 3D simulator (Obst, O. & Rollman, M., 2005) and the 2D simulator (Chen, M. et al., 2007) used in previous RoboCup competitions, and in our previous research, are very significant.

Similarly to the 2D simulator, the simulation environment of the RoboCup 3D Simulation League is based on a client-server model. The simulator is the server and agents and visualization tools are the clients. The simulator creates the virtual environment (soccer field, markers, goals, etc.) where agents live, sends sensory information to the agents, receives their actions and applies the virtual physics model in order to resolve positions, collisions and interactions with the ball. Each team plays with 11 agents that must cooperate to score as much goals as possible while not allowing the other team to score.

The development of the 3D simulator used available open-source tools extensively. It uses the SPADES (Riley, P. 2003, Riley, P., 2003a) framework for the management of agent-world communication and synchronization, ODE (Smith, R., 2006) for the physical model, expat (Expat XML Parser, 2004) for XML processing, Ruby (Ruby 2004) for scripting language support and boost (Boost, 2004) for several utilities.

The 3D simulation server is implemented above a platform called SPADES (System for Parallel Agent Discrete Agent Simulation) (Riley, P., 2003). SPADES is a middleware system for agent-based distributed simulation. It aims to provide a generic platform to run in multi-computer systems. It implements the basic structure to allow the interaction between agents and a simulated world so that the users do not have to worry about communication and synchronization mechanisms such as sockets, addresses, etc.

SPADES' main features are:

- Agent based execution – support to implement sensations, thinking and actions.
- Distributed processing – support to run the agents applications on many computers.
- Results unaffected by network delays or load variations among the machines – SPADES ensure that the events are processed in the appropriate order.
- Agents can be programmed independently from the programming language – the agents can be programmed in any language once it provides methods to write/read to/from Pipes.
- Actions do not need to be synchronized in the domain – the actions of the agents can take effect at varying times during the simulation.

SPADES components are organized in a client-server architecture. The Simulation Engine and the Communication Server are provided by SPADES; while the Agents and the World Model are built by the user and run upon the formers.

The Simulation Engine is a generic piece of software that provides abstractions to create specific world models upon it. Agents may run in the same computer of the Simulation Engine on in remote computers linked to the network, in this case a Communication Server must be running in the remote computer. The World Model module must be running in the same computer of the Simulation Engine. This module specifies the characteristics of the environment where the agent will live.

SPADES implements what it calls the sense-think-act cycle in which each agent receives sensations and replies with actions. That means that an agent is only able to react after receiving a sensation message. The agent is also capable of requesting its own sensations, but the principle remains - a sensation must always precede an action. In order to allow actions between "normal" sensations, SPADES provides an action called *request time notify* that returns an empty sensation and after receiving it the agent is able to respond with actions. For example, if an agent received a sensation at cycle 100 and wants to produce an action at cycle 110, and if the next sensation will only arrive at cycle 120, the agent can ask to

receive a time notify message at cycle 110 and just reply with the desired action after receiving it.
Fig. 4 depicts the sense-think-act cycle and the time where each of its components runs. From A to B a sensation is sent to the agent. After receiving the sensation (from B to C) the agent decides which actions will be executed; then from (C to D) the actions are sent do the server.

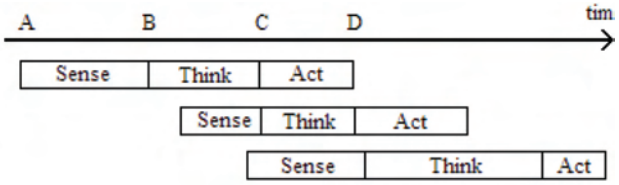


Fig. 4. SPADES Sense-Think-Act Cycle

In many agents, the sense, think and act components may be overlapped in time (like in Fig. 4). There is just one restriction: The thinking cycles for one agent cannot be overlapped. This constraint makes sense, since just a single processing unit is used per agent, and thus, just one sensation at time can be processed.
As stated before the simulator runs upon the SPADES, and uses ODE to calculate the physical interactions between the objects of the world. The graphical interface is implemented using OpenGL.
The 3D simulation server (Obst, O & Rollmann, M. 2005) allows twenty two agents (eleven from each team) to interact with the server in order to play a simulated robotic soccer game. Each agent receives sensations about the relative position of the other players and field goals and other information concerned with the game state and conditions. The information about the positioning of the objects in the world is given by the vision sensor that initially allowed the agent to receive visual information in 360 degrees but changed in 2006 to a 180 degrees angle of vision. The agents have the shape of a sphere (Fig. 5). Replying each sensation an agent sends actions like *drive* or *kick*. Driving implies applying a force on the body with a given direction and kicking implies applying a force on the ball radially to the agent. Each sensation is received on every 20 cycles of the server and each cycle takes 10 ms.

Fig. 5. 3D Simulation League Match – Sphere Robot Model

Each sensation takes 10 cycles to reach the agent (send delay) and actions sent by the agent take 10 cycles to reach the simulator. Hence a sent action starts to take effect at the time the next sensation is sent by the server as it is shown in Fig. 6.

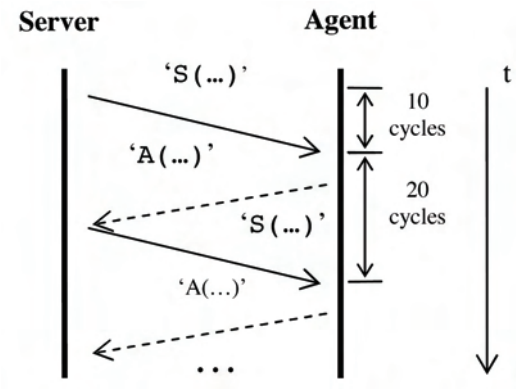


Fig. 6: Server-Agent Protocol in 3D simulator

In 2007 the robot model used in the 3D simulator changed from the sphere to a humanoid model. Also the SPADES support was abandoned and a new simpler type of timer was developed. The humanoid simulator only allowed games of two against two, but it is expected that in the future the number of robots for each team will increase. Figure 7 shows an image of the RoboCup 2007 3D Simulation Final.

Fig. 7. 3D Simulation League Match – Humanoid Robot Model

2.4. Microsoft Soccer Robotics Challenge

As part of its efforts to promote their recent Robotics Studio (Microsoft, 2007), Microsoft developed a simple soccer simulator and sponsored a new demonstration league at RoboCup 2007. Microsoft Robotics Studio uses the AGEIA PhysX physics-based 3D engine (AGEIA, 2007) to create simulated but realistic environments for robotics research while

making it easy to move the developed controllers to real robots. The AGEIA PhysX enables developing rich immersive physical gaming environments with features such as explosions; and characters with complex, jointed geometries.
Initially, it was supposed that teams would work with an early version of the simulator supporting only wheeled robots so that developers could have a chance to become familiar with the simulator and could work on team strategy and high-level coordination. However, Microsoft fastly developed a reasonably stable simulator capable of using legged robots. This was the simulator used in RoboCup 2007 competition. However, focus of the competition had to be moved from coordination to vision, low-level robot control and navigation, since the simulator was not stable enough to accommodate more than two players by team.
The new robot soccer simulation developed, included a 3D simulated soccer field and refereeing services, as well as support enabling different simulated robots to be used as soccer players. A simulated four-legged robot player, called RobuDog (Fig. 8), from Robosoft, was developed to be the base of the competition (Robosoft, 2007). Robosoft also previewed its hardware-based RobuDog robot, stating that the code developed for use in the simulated competition can be used directly on this new physical robot.

Fig. 8. The Robudog Real Robot (Robosoft, 2007)

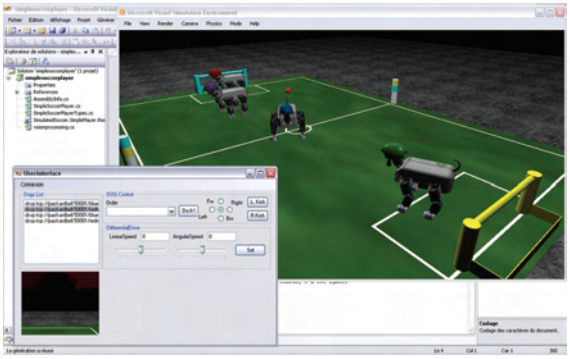


Fig. 9. The Robudog Simulated Robot and Micrsoft Robotics Soccer Challenge Field (Robosoft, 2007)

The new league was quite successful in RoboCup 2007 – Atlanta (Fig. 9). In the future it will become more clear that Microsoft is not limiting itself to just simulation. They intend to

work closely with robot manufacturers to develop real soccer playing robots with software development based on the Microsoft Robotics Studio, making an impact in robotics over the next years. Microsoft gained support from a number of robot manufacturers including Kuka, Lego Systems, iRobot, SRI International, Yujin Robotics, Coroware, Parallax, Robosoft, etc. Thus, support for this new league seems well secured and new, more challenging rules are being developed for next year.

2.5. Physical Visualization League

The Physical Visualization (PV) Soccer League is a new RoboCup league where small real robots, called Eco-Bes, play soccer on top of a virtual field with a virtual ball, thus using the concept of augmented reality (Mackay, W. & Gold, R., 1993). Augmented Reality is an environment that includes both virtual reality and real-world elements (Milgram, P. & Kishino, F., 1994). Most augmented reality research uses a processed video which is augmented with virtual elements. The Augmented Reality at Physical Visualization League can improve the simulation, adding virtual elements that surround the real player. An example applied in that soccer league is a virtual leg with the ability to kick and to dribble the virtual ball and vision to perceive the world state (Azuma, 1997).

The Physical Visualization League offers a very interesting challenge for teams since several research challenges are included in this setup (Vision Based Self Localization, Data Fusion, Real-Time Control; Decision and Cooperation). The simplicity of this setup compared with the small-size league, makes it very interesting for educational and demonstration purposes (RoboCup, 2007a).

The Eco-Be is a very small vehicle remotely controlled by infrared commands, currently handmade. As presented in Fig. 10, it is composed by two step motors (1), a li-ion polymer battery (2), a control board (using an 8bit PIC18 family processor) (3), an Infrared Sensor (4) to receive its movement commands and an aluminium body (5). The robot can not send any kind of messages and its position is determined by an external camera (Yanagimachi, S. & Guerra, R., 2007). Each robot has a configurable ID, which can be freely changed. The robot can use each motor individually and each motor has three different speeds available. While using the fastest speed, all resources are drained from the controller, disabling the reception of new commands during this fast movement.

The kit furnished by Citizen contains a set of 20 robots with one charger, two AC/DC adapters and one infrared transmitter.

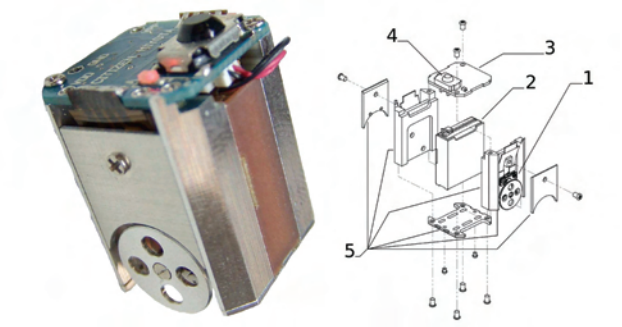


Fig. 10. The Eco-Be Robot (Yanagimachi, S. & Guerra, R. 2007)

The Eco-Be has a simple protocol of communication to control its movement. The robot's IR sensor recognizes a command through the modulation of the flash light as a square wave at 40 kHz with an on-to-off ratio of 50%. The period of Signal/Space allows the translation into valid logical states.
The LIRC software (Linux Infrared Remote Control) encodes the correct sequence of flash lights to compose a valid string to the robot (LIRC, 2007). LIRC is an open source software capable to decode and send infrared signals from and to the common serial port. It receives commands via socket and sends them to the infrared device driver. A specific communication protocol is followed in order to send commands to the robot. The transmitter uses a public circuit which receives a signal from the serial port and polarizes the infrared LED accordingly. The robot's Infrared Protocol accepts a string of 12 bits as a valid word. The first 5 bits identify the destination robot. The next three bits command the left motor and the following three command the right motor. The last bit is used for parity.
The Physical Visualization Sub-League started in 2007 focused on augmented reality, on its practical simple environment, and on demonstrating that the possibilities of the Eco-Bes are not limited to robot soccer. The league format had three competitions: demonstrations, technical developments and soccer tournament (RoboCup, 2007a). The main rules and league format details are shown as follow.
The final rules applied at RoboCup 2007 defined that each team plays with just two players in the soccer games. The number of players was fixed at only two players by the league committee due to technical limitations of the platform in this initial stage of development. Each soccer match takes ten minutes and each team has an optional break time of two minutes.
The PV League main part, the soccer tournament, is organized in two stages: the group stage and the elimination rounds. In the group stage, like in real soccer, each team is awarded 3 points winning, 1 for a drawing and 0 for a losing. In this group stage each team plays once with the other teams in the same group and the top two teams move forward to elimination rounds.
The elimination rounds have extra time of five minutes in case of a draw in the regular time. If no team scores a goal, teams alternately start a sequence of five penalty kicks. In this penalty kicks the attacker has to score a goal, the ball starts at middle field and the attacker a little behind. There is no limit to the number of times each player can touch the ball however the penalty kick is over if the ball exits trough the back line or after thirty seconds without a goal. If after five penalty kicks for each team the score is even, a new series of five penalties without goalies starts. Finally if the game is still tied a coin is flipped to decide the winner.
Rules of the tournament disallow communication between the teammates in such a way that it is solely up to the server to provide players' with sensory data.
The soccer tournament was organized with two groups of five teams playing the round-robin model. The best four go to the semi-finals, and, then, the winners pass to the finals.

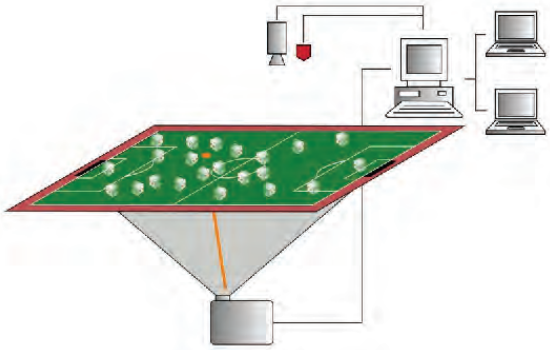


Fig. 11. PV League Schematic Setup (RoboCup, 2007a)

Prior to any game the camera is calibrated to recognize the field and the robots' markers. Each robot is identified by an individual marker that is recognized by the vision system. Each robot then receives an ID that corresponds to its marker and thus may be controlled by an autonomous agent running on a separate PC. Fig. 11 presents the field Schematic Setup.
A server application is responsible for controlling the socket connections with the clients, the monitor, the camera and the communication with the infrared USB transmitter. It has a file that defines the field size and the positions of each goal and each flag pole. For each instant, the server just has to identify the robots using their color markers. For the other elements, the server uses the positions in the configuration file. The server, then, compiles all information and, for each robot, it sends the relative positions (polar coordinates) of all elements in the field, including the ball and the other robots.
An associated monitor application uses the same information to draw the field and project it at the display where the robots play. The server sends the robot's absolute position to the monitor that projects them in the screen with a considerable precision. The monitor also shows the virtual ball.
Fig. 12 depicts a match with all the necessary elements. The field's background, lines, goals and poles are the passive elements drawn by the monitor. The marks below the robots and the ball is repeatedly updated, this way when the robots are over the screen monitor, the setup transmits the sensation the robots are really controlling the virtual ball with their physical movements.

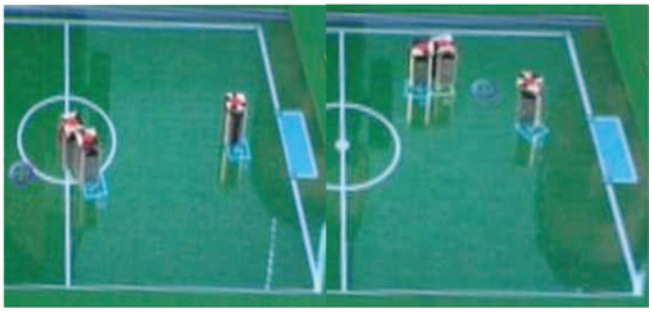


Fig. 12. Snapshot from a PV-League official match

Developing new low level skills for physical robots in virtual environments is a very challenging task as it was clear in RoboCup 2007, first edition of the PV-League. Methods for navigation, ball dribbling, passing, shooting and for goalie positioning were the basis for having a successful team.
In RoboCup 2007 it was also clear that: this new competition was clearly both viable and successful, the hardware platform can have a bright future, where multi-robot research can be directed to new and diverse scientific goals and finally that the league is very attractive to a wider audience.

2.6. Nanogram League

The RoboCup Nanogram competition challenges researchers to construct microscopic robots that compete against each other in soccer-related agility challenges. These robots measure a few tens of micrometers to a few hundred micrometers in their largest dimension and have masses ranging from a few nanograms to a few hundred nanograms.
The playing field consisted of a set of insulated interdigitated electrodes, across which an AC waveform can be applied (Donaldt, B.; Levey, C. ; McGray, C.; Rus, D. & Sinclair M. 2003), (Donald, B; Levey, C.; McGray, C.; Paprotny, I. & Rus, D., 2006). This waveform provides both the electrical power and the control instructions for the micro robots through a capacitive coupling. A digital camera placed on top of the field captures the action through a microscope sending it to the robot's control systems.

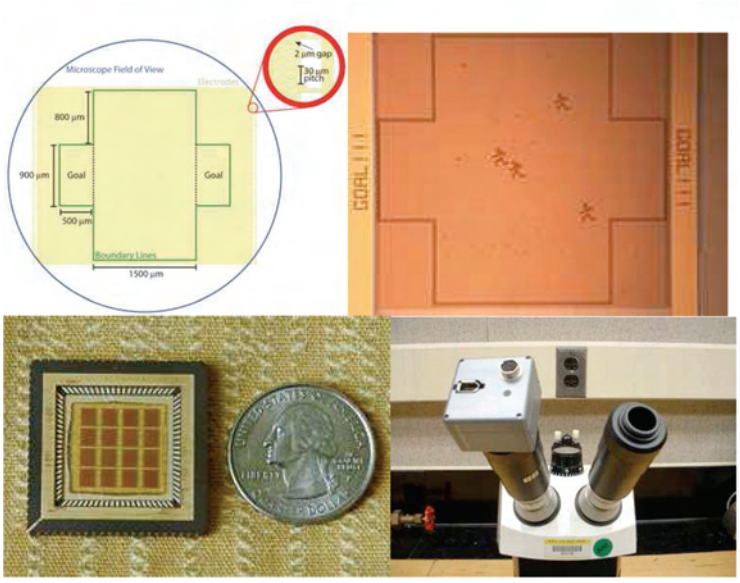


Fig. 13. Nanogram League Field (top), comparison of 16 fields with a coin (bottom-left) and the league Visualization Equipment (bottom-right) (McGray, C.; Arai, F.; Jacoff, A.; Tadokoro, S. & Gaitan, M., 2007)

For the first demonstration competition held in Atlanta in 2007, the contest consisted of three compulsory exercises: The 2 Millimeter Dash; the Slalom Drill; and the Ball-Handling Drill.

Fig. 14. The 2 Millimeter Dash; the Slalom Drill; and the Ball-Handling Drill (McGray, C.; Arai, F.; Jacoff, A.; Tadokoro, S. & Gaitan, M., 2007)

In the 2 Millimeter Dash, each microrobot must sprint across the playing field from one side to the other. The micro-robot begins with its entire structure behind one of the goal lines, and then it must dash until the first point of its structure crosses the opposing goal line. Each team will be allowed three trials being the winner determined by the best of the three.
In the Slalom Drill, the path between goals was blocked by simple objects (representing inanimate "defenders") that the micro-robot should avoid as it goes from one goal to the other. Each team was allowed three trials counting for scoring the best of the three.
The Ball Handling Drill was a more soccer-like challenge. The objective was for the robot to dribble as many balls as possible into the goal within three minutes, also avoiding the inanimate "defenders". The balls consisted of thin-film discs of silicon nitride, with dimples on their base to enable easy sliding along the field of play. Also, each team had three trials for this ball handling drill, with balls and defenders placed in different locations for each trial.
The league was also quite successful in its first appearance in RoboCup 2007. ETH Zurich (Swiss Federal Institute of Technology) won all the competitions, achieving the 2 Millimeter Dash in 316 milliseconds, the Slalom Drill in 583 milliseconds and scoring three goals in the Ball Handling Drill.

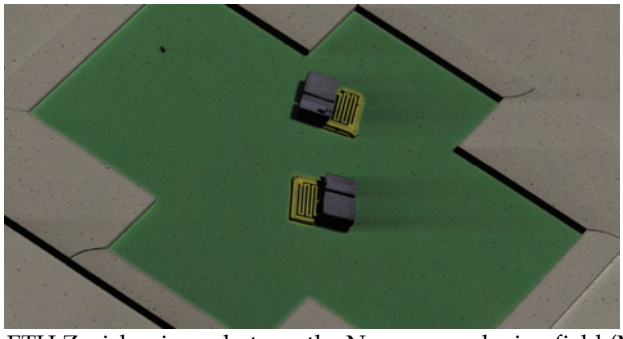


Fig. 15. Two IRIS ETH Zurich microrobots on the Nanogram playing field (McGray, C.; Arai, F.; Jacoff, A.; Tadokoro, S. & Gaitan, M., 2007)

In future years of competition, it is predictable that the competition should become more complex and the environment less structured, to encourage progress in robot cooperation, 3D manipulation, on-board power supplies, integrated logic and sensing, locomotion over rough terrain and robust operation in dirty, wet, and volatile environments (McGray, C.; Arai, F.; Jacoff, A.; Tadokoro, S. & Gaitan, M., 2007)

3. Agent Architecture and Knowledge Structures

To enable a team to perform cooperative multi-agent tasks, like playing simulated soccer, in a partially cooperative, partially adversarial environment a lot of knowledge is needed. Also, agents must have a world state representation as updated and as accurate as possible. Whenever the domain becomes more complex, knowledge importance is even greater. This is the case in multi-objective, partially cooperative and adversarial domains in which agents have limited perception and action capabilities. For this type of domains we argue that to correctly perform cooperative tasks, agents should include knowledge at three levels:

- Individual action execution;
- Individual decision-making;
- Cooperation.

Knowledge for executing actions is concerned with the specific commands needed to perform a given low-level skill. Individual decision-making knowledge is concerned with the way agents choose the action to execute. Knowledge for cooperation is concerned with tactics, situations, dynamic formations, roles, dynamic plans and communication protocols (Reis, L.P. & Lau, N., 2001). Representation structures for this type of multi-level knowledge are one of our research goals. The world state representation must remain updated so that it may be used to effectively decide the individual and cooperative actions to perform. The following methods are used to update our agent's world state information: visual perception analysis; communication; and action prediction.

4. Coordination Methodologies

In RoboCup past editions, teams with the best decision making mechanisms were very successful. In RoboCup 2001, the champions (Tsinghuaeolus (Yao, J.; Chen, J.; Cai, Y., Li, S. 2002) also champions in 2002) and vice-champions (Brainstormers (Riedmiller, M. & Merke, A., 2002)) did not use the coach agent and trusted mainly on their better low-level skills and well-tuned individual decision-making mechanisms. In RoboCup 2004 STEP won the competition mainly due to their fast dribbling ability. So, besides having a configurable strategy and flexible coordination mechanisms (Lau, N. & Reis, L.P. 2002), well-tuned individual decision making mechanisms are still very important in RoboCup simulation league (Teixeira, C.; Lau N. & Reis, L.P. 2004).

How to define roles based on standardized agent behavior characteristics for the RoboCup simulated soccer domain is one of the problems that has been tackled. To improve the flexibility of our team, agents are able to switch their relative positions (for a given formation) and roles (that define agent behavior at several levels), at run-time, on the field. We have proposed, and continually developed, Situation Based Strategic Positioning (SBSP) mechanism (Reis, L. P.; Lau, N. & Oliveira, E. C. 2001) that may be used to dynamically spatially position a team using different flexible formations for different situations. This

mechanism is based on the distinction between active and strategic situations (Reis, L.P. & Lau, N. 2001). If an agent is not involved in an active situation then it tries to occupy its strategic positioning that change according to the situation of the game. Situation is a concept on a high-level analysis of the game (attacking or defending for example). SBSP was one of the main innovations of FC Portugal and is now used directly or as the base for the positioning systems of many simulated soccer teams and being used in some middle-size league teams.

4.1. Strategical Coordination

CMUnited brought the concepts of formation and positioning to RoboSoccer (Stone, P. & Veloso, M.) (1999; Stone, P., 2000) and used dynamic switching of formations as well. FC Portugal extended these concepts and introduced the concepts of tactics and player types. FC Portugal's team strategy is based on a set of tactics to be used in different game situations and a set of player types (Fig. 16).

Tactics include several formations used for different game specific situations (defense, attack, goalie free kick, scoring opportunity, etc). Formations are composed by eleven positionings that assign each player a given player type and a base strategic position on the field.

One of the most significant features is the clear distinction between strategic situations (when the agent believes that it is not going to use an active behavior soon) and active situations (ball recovery and ball possession). In strategic situations, players use a SBSP mechanism (presented in section 4.2). For active situations---ball possession, ball recovery or game stopped---decision mechanisms based on the integration of real soccer knowledge are used.

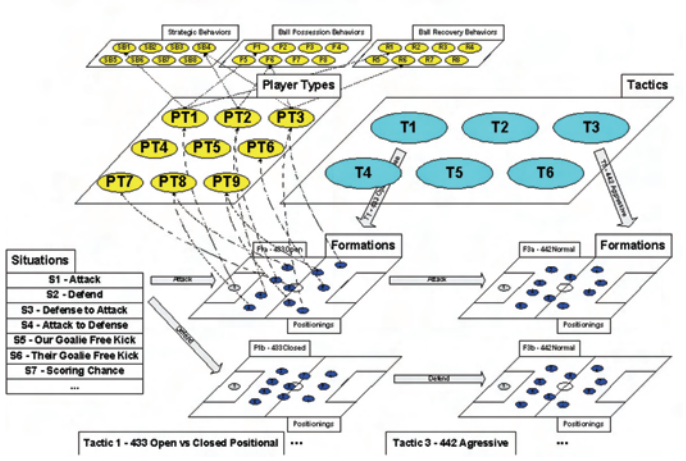


Fig. 16. FC Portugal's Strategical Model

4.2. Situation Based Strategic Positioning

Situation Based Strategic Positioning (SBSP) mechanism (Reis, L.P. & Lau, N. 2001; Reis, L. P.; Lau, N. & Oliveira, E.C. 2001) is used for strategic situations (in which the agent believes

that it is not going to enter in active behavior soon). To calculate its strategic positioning, the agent analyzes which is the game situation. Then the agent calculates its base strategic position in the field in that formation, adjusting it according to the ball position and velocity, situation and player type strategic information. The result is the best strategic position in the field for each player in each situation. Since, at each time, only a few players are in active behavior (conducting the ball or trying to recover the ball) most players are close to their strategic positionings. SBSP enables the team to move similarly to a real soccer team, covering the ball while the team remains distributed along the field.

```
ALGORITHM SituationBasedStrategicPositioning(Tactic,
                Situation, Player)
RETURNS Position
PARAMETERS Tactic, Situation, Player
{
  Formation = SituationFormation(Tactic, Situation)
  FormHeight =
      SituationFormationHeight(Tactic, Situation)
  FormWidth =
      SituationFormationWidth(Tactic, Situation)
  Positioning = PlayerGetPositioning(Player)
  Position =
      HomeSBSPPosition(Formation, FormWidth,
                       FormHeight, Positioning)
  Role = FormationGetRole(Formation, Positioning)
  RoleStrategic = RoleGetStrategicRole(Role)
  BallAdjPos = AdjustedBallPosition(BallPosition)
  Position = BasicSBSPPosition(Position, Formation,
              RoleStrategic, Positioning, BallAdjPos)
  Position = RegionalAdjustSBSPPosition(Position,
   Formation, RoleStrategic, Positioning, BallAdjPos)
  Position =
       LegalAdjustSBSPPosition(Position, BallAdjPos)
  Position =
       DomainAdjustSBSPPosition(Position, BallAdjPos)
  RETURN Position
}
```

Fig. 17. SBSP – Situation Based Strategic Positioning

4.3. Dynamic Positioning and Role Exchange

The Dynamic Positioning and Role Exchange (DPRE), and Dynamic Covering (Reis, L.P. & Lau, N. 2001), was based on previous work from Peter Stone (Stone, P. 2000) which suggested the use of flexible agent roles with protocols for switching among them. The concept was extended and players may exchange their positionings and player types in the current formation if the utility of that exchange is positive for the team. Positioning exchange utilities are calculated using the distances from the player's present positions to

their strategic positions and the importance of their positionings in the formation on that situation.

```
ALGORITHM DynamicPositioningExchange(WorldState,
                              Situation, Positionings)
RETURNS
        Positionings(TeamSize)
PARAMETERS
        WorldState, Positionings[TeamSize], Situation
{
FOR Pl1 = 2 TO TeamSize-1 DO
  FOR Pl2 = Pl1+1 TO TeamSize DO
   IF PositionValid(Pl1) AND PositionValid(Pl2) THEN
   {
    Dist11 = Distance(Position(Pl1),SBSPPosition(Pl1))
    Dist22 = Distance(Position(Pl2),SBSPPosition(Pl2))
    Dist12 = Distance(Position(Pl1),SBSPPosition(Pl2))
    Dist21 = Distance(Position(Pl2),SBSPPosition(Pl1))
    Adeq11 = PosAdequacy(Pl1, Positioning[Pl1])
    Adeq22 = PosAdequacy(Pl2, Positioning[Pl2])
    Adeq12 = PosAdequacy(Pl1, Positioning[Pl2])
    Adeq21 = PosAdequacy(Pl2, Positioning[Pl1])
    Util = ExchangePositions(DPREMode, Situation,
         Dist11, Dist22, Dist12, Dist21,
         Adeq11, Adeq22, Adeq12, Adeq21,
         PosImportance(Positioning[Pl1]),
         PosImportance(Positioning([Pl2])
    IF Util > ThresUtil(Situation) THEN {
         Aux = Positionings[Pl1]
         Positionings[Pl1] = Positionings[Pl2]
         Positionings[Pl2] = Aux
         }
    }
RETURN Positionings
}
```

Fig. 18. DPRE – Dynamic Positioning and Role Exchange

4.5. Intelligent Perception and Communication

The Communication model of the simulation league is restricted by the available bandwidth and uncertainty of message delivery. In 2002 the communication rules have changed: bandwidth has been constrained (messages with maximum length of 10 bytes), but uncertainty on delivery can now be reduced. Also, a new form of visual communication (enabling players to point to regions of the field) has been introduced. We identify four potential research areas in the definition of a communication protocol:

- What to communicate?
- When to communicate?
- Who should be heard at each time?

• How received messages will affect player's behavior?

Previous approaches have used message contents, mainly, to share world state knowledge between players (Stone, P. 2000), to communicate useful events/opportunities and to enhance cooperation (Reis, L.P. & Lau, N. 2001). With the reduction in message size, teams must carefully select which information should be conveyed. We have extended our ADVCOM principle: "Communicate only when you have something important so say" (Reis, L.P. & Lau, N. 2001) with "Communicate only what is important", measuring the importance of each piece of information through utility metrics based on the current situation and on estimated teammates knowledge. This idea was also been extended with a Situation Based Communication framework (Ferreira R.; Reis, L.P. & Lau, N. 2004). In RoboCup simulation league, if two or more players talk in the same simulation cycle, the simulator delivers only one of the messages to the other players. Teams have dealt with this restriction either by assuring that only one player talks in each cycle or by allowing several players to talk (Reis, L.P. & Lau, N. 2001). We believe that the first approach is not sufficiently flexible and reliable and thus have used the second approach (Reis, L.P. & Lau, N. 2001). In our protocol every player estimates the importance of his knowledge to the rest of the team through utility measures and only communicates when its communication utility his higher than the others or is above a threshold.

In the simulation league agent's visual perception is obtained through controllable sensors. Players may control their visual quality, the sensibility angle and the position of their neck relatively to the agent's body. We have developed intelligent perception mechanisms, namely, SLM - Strategic Looking Mechanism (Reis, L.P. & Lau, N. 2001). SLM decides the direction a player should look, in each cycle, maximizing the predicted world state update value from that perception.

5. Game Analysis and Coaching

In RoboCup simulation league, the online coach agent has a global vision of the field (without errors) gathered from the soccer server (Chen et al., 2007). The coach agent is able to analyze the game and send high-level commands to his team in order to improve the team global behavior.

In previous work (Reis, L.P. & Lau, N., 2002) we have explored different ways of implementing the coach agent. Different coaching architectures have been implemented and compared (Fig. 19), including the division of the coach agent into one assistant agent (capable of gathering game statistical information and opponent modeling information) and a principal coach (that uses the information provided by the assistant coach in order to decide the best tactic to be used by the team at each moment in the game). Other coaching architectures were also explored, including the subdivision of the coach functions by the players in order to have a completely distributed coaching behavior.

We have proposed Coach Unilang – a general language to coach a (robo)soccer team (Reis, L.P. & Lau, N., 2002). The development of translators from Coach Unilang to Clang has provided us with a very useful tool to test team behaviors and to participate with success in the Coach Competition.

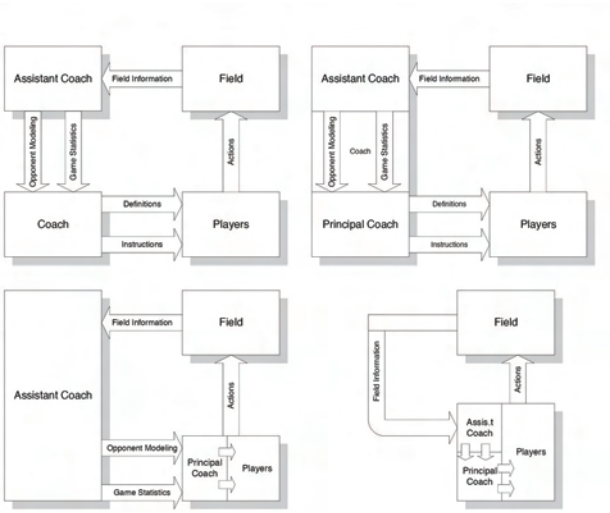


Fig. 19. Coaching Architectures of COACH UNILANG

6. Agent Analysis and Debugging Tools

Soccer Playing Agents have the following characteristics: autonomy, reactivity, pro-activity and social ability. Although agents take their decisions autonomously, the developer should be able to motivate the use of "good" actions and deprecate the use of "bad" actions. Furthermore the environment of the simulation leagues is uncertain, dynamic, real-time, heterogeneous, partially cooperative, and partially adversely. Agents take a large amount of decisions in very short time and understanding their decisions at the same time they are executing is a very complex task. Having these considerations in mind, we feel that offline analysis of agent's decisions, i.e. examining agent reasoning after execution, without real-time pressure, is the best way of making in-depth analysis of agent's reasoning.
Our debugging methodology is based on the following principles:

- Offline debugging;
- Visual debugging;
- Superimposed real environment and agent physical knowledge;
- Feature-focused debugging;
- Information structured in layers of abstraction with different detail levels.

In the context of the development of FC Portugal agents we have identified the following features that should be object of independent selection by the developer during a debugging session:

- Communication;
- Ball Possession;
- Ball Recovery;
- Synchronization;
- Low-level skills;
- Opponent Modeling.

The selection of each of these features, although independent, is not exclusive. The developer can activate one or more of the above listed features at each moment providing him with maximum flexibility in the selection of the information he finds relevant.
During the course of the development of FC Portugal agents, the following development tools have been implemented:

- Visual Debugger used to analyze the reasoning of agents (Reis, L.P. & Lau, N., 2001).
- Team Designer that enables the graphical definition of soccer strategies;
- Offline client methodology;
- WstateMetrics that evaluates the accuracy of world states;

Evaluation by domain experts using graphical tools is another methodology that has been used to fine tune our team.

6.1. Visual Debugger

The main debugging tool of FC Portugal is called Visual Debugger (Fig. 20). Its implementation is based on CMUnited99 layered disclosure tool (Stone, P., Riley, P. & Veloso, M., 2000) and the soccerserver logplayer application. CMUnited layered disclosure tool included the possibility of synchronous visualization of the game (using soccermonitor) and of one of the players reasoning (at several levels of abstraction) saved in action logfiles. We have integrated the two applications (logplayer with layered disclosure and soccermonitor) in a powerful team debugging tool and added the visual debugging capabilities and real and believed world-states superposition.

Fig. 20. The Visual Debugger window

Visual information is much more easily handled than text information. Soccermonitor (Chen et al., 2007) includes the possibility of drawing points, lines and circles over the field, but this functionality is not reported as being used by other teams. This soccermonitor feature was exploited, modified and extended. Agents can present their reasoning in graphical way using a simple API that includes functions like:

```
LogDrawLine(level, xy_i, xy_f, color);
LogDrawCircle(level, xy_center, rad, color);
```

This way the agent knowledge can easily be superimposed (and compared) with real data taken from the simulator logfile. This allows the developer to understand why the agent decided in a particular way and to focus developer attention on the most relevant features that need to be improved.
The developer can navigate over the pre-recorded game back and forth, examining the reasoning of each of the 11 players in the team. The level of detail of the information retrieved by the visual debugger may be controlled by the developer.

6.2. Team Designer

Team Designer application includes a tactics editor, a game statistical analysis tool, an offline coach and an online coach. The tactics editor allows the definition of the whole strategy to use during a given game: tactics, formations, individual player decision, strategic positioning features, etc. may all be changed in a friendly and safe way. Some of the tactic parameters are defined by direct manipulation of their graphic view. This is the case for the definition of players' home positions inside a formation and for the definition of new situations using the integrated offline coach.
The creation of a new strategy may use features from previously saved strategies, through the selection of which items are interesting to merge (Fig. 21).

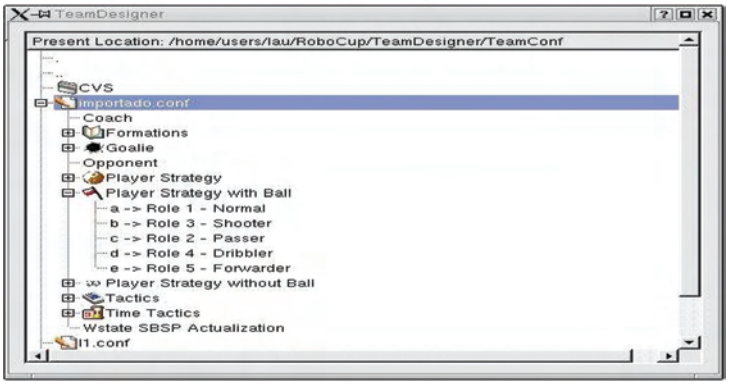


Fig. 21. Defining a tactic using Team Designer

The analysis tools gather game statistical information that is shown to the user and sent to the online coach. The statistics include ball position in several field matrixes, shoots and passes by field areas, ball loss and recovery positions, etc (Reis, L.P. & Lau, N., 2002).

6.3. Offline Client

In some situations the action log files saved by players are not sufficient to understand what really happened in a certain situation. A finer debugging degree can be achieved by employing our offline client tool.
The principle of the offline client is that we can repeat the execution of an agent over exactly the same setting of a previous game without the intervention of the server and without real-time constraints. Then we can use a normal debugger (like gdb) to examine the contents of all variables, set breakpoints, etc. at the execution situations we want to analyze.

A special log file, that records every interaction with the server and the occurrence of timer signals, must be generated to use the offline client. If an agent has probabilistic behavior some more information might be needed. The offline execution of an agent is achieved through a stub routine that reads the special log file. Player's behavior is maintained, as it is not affected by the substitution of this stub routine. The execution of a normal debugger over this offline client permits complete control of the execution flow of the agent reasoning.

7. Results and Discussion

The experiences performed to validate the approach used eight of the historical best known 2D soccer simulation teams. FC Portugal base team, using a single, well-tuned tactic, was tested against these teams using different coordination methodologies.

The first experiment was performed by performing 10 soccer 2D simulated games each of the 8 opponents selected, using five distinct positioning systems. A total of 10*8*5=400 games were performed for this experiment. The positioning systems used were:

- PACTS (Simple Active Positioning) – No strategic positioning. Agents always assume active positioning (ball possession or ball recovery active behavior).
- PACTF (Active Positioning with Static Formation) – Similar to PACTS but all agents posses a default static formation. If no active action is sufficiently good, agents return to their base positions.
- SPAR (Positioning by Attractions and Repulsions) – Positioning based on Stone et al. algorithm (Stone et al., 2000). Players are attracted by the ball, repelled by teammates and attracted by opponents if they have the ball or repelled otherwise.
- SP (Simple Strategic Positioning) – Using only one situation and one dynamic formation.
- SBSP (Situation Based Strategic Positioning) - Using situations for attack, defense, goal kicks, corners and throw-ins.

The analysis was made by automatically calculating, using the tool previously described, the goals scored and conceded in each game, the number of games won, draw or loosed by the team, the number of shoots made and conceded and global ball possession statistics in three field regions (attack, middle field and defense). The results achieved are shown in Table 1.

Analysing the results achieved it is easy to conclude that the best results are achieved using SBSP positioning. Another conclusion is that a good positioning system like SBSP enables the team to achieve better results against the best teams. The difference between the SP and SBSP positioning systems is clearly visible mainly against the best opponents (TSI and KB) while has low or almost no influence against the weakest opponents.

The second experiment was performed in order to evaluate the usefulness of the DPRE – Dynamic Positioning and Role Exchange mechanism. Ten games were performed against each of the eight selected opponents with and without using DPRE. The results achieved in the 10*8*2= 160 games conducted, are summarized in Table 2.

By analysing the results achieved, the main conclusion is that DPRE has greater impact against the best opponents. Although the number of shoots of the opponents does not increase significantly, the number of conceded goals increases significantly by not using DPRE. By analysing some of the real games in more detail it may be concluded that in some situations by not executing appropriate positioning exchanges in defense "holes" appear in the team defense letting the opponents score easily.

Results \ Opp. Teams	TSI	KB	UVA	YOW	FCM	FA	ATT	CMU
PACTS (Simple Active Positioning)								
Goals (Scor.-Conc.)	0-65	0-22	1-38	6-19	0-34	2-2	14-6	21-0
Wins–Draws–Loses	0-0-10	0-2-8	0-0-10	1-3-6	0-0-10	2-6-2	8-1-1	10-0-0
Shoots	1-86	2-36	2-72	13-37	2-61	5-16	26-14	37-16
Attack–Middle-Def.	8-58-34	9-61-30	14-44-42	16-54-30	8-47-45	14-58-28	38-46-16	42-46-12
PACTF (Active Positioning with Static Formation)								
Goals (Scor.-Conc.)	0-48	1-18	2-46	10-16	4-24	4-3	24-4	32-0
Wins–Draws–Loses	0-0-10	0-3-7	0-0-10	3-4-3	0-0-10	3-5-2	10-0-0	10-0-0
Shoots	2-75	3-32	5-89	21-39	6-48	6-20	39-12	54-8
Attack–Middle-Def.	12-56-32	10-58-32	18-47-35	22-50-28	14-55-31	24-50-26	44-42-14	50-43-7
SPAR (Positioning by Attractions and Repulsions)								
Goals (Scor.-Conc.)	0-36	2-12	1-42	8-14	5-25	6-2	33-3	45-0
Wins–Draws–Loses	0-0-10	1-4-5	0-0-10	2-4-4	0-0-10	4-5-1	10-0-0	10-0-0
Shoots	2-55	8-26	12-76	16-42	6-46	8-12	52-10	71-5
Attack–Middle-Def.	10-54-36	13-52-35	19-41-40	25-45-30	16-41-43	29-45-26	50-37-13	58-34-8
SP (Simple Strategic Positioning)								
Goals (Scor.-Conc.)	9-6	12-1	30-7	91-0	81-1	145-0	209-0	236-0
Wins–Draws–Loses	4-3-3	7-3-0	8-1-1	10-0-0	10-0-0	10-0-0	10-0-0	10-0-0
Shoots	23-12	22-6	42-18	132-3	108-10	196-1	264-0	256-0
Attack–Middle-Def.	31-51-18	48-33-19	38-37-25	60-30-10	53-34-13	68-29-3	74-25-1	77-22-1
SBSP(Situation Based Strategic Positioning)								
Goals (Scor.-Conc.)	10-3	19-0	32-6	98-0	76-1	158-0	204-0	246-0
Wins–Draws–Loses	6-2-2	7-3-0	9-1-0	10-0-0	10-0-0	10-0-0	10-0-0	10-0-0
Shoots	16-4	35-2	45-16	146-0	102-3	209-0	253-0	273-0
Attack–Middle-Def.	34-50-16	40-41-19	40-35-25	63-25-12	55-32-13	71-27-1	72-26-2	75-24-1

Table 1. Results Achieved Using Different Positioning Systems

Results \ Opp. Teams	TSI	KB	UVA	YOW	FCM	FA	ATT	CMU
Without using DPRE – Dynamic Positioning and Role Exchange								
Goals (Scor.-Conc.)	8-10	10-3	24-12	76-0	65-6	126-0	187-0	225-0
Wins–Draws–Loses	2-4-4	6-3-1	6-2-2	10-0-0	9-0-1	10-0-0	10-0-0	10-0-0
Shoots	18-14	16-7	34-21	108-2	89-15	170-1	232-0	243-0
Attack–Middle-Def.	27-50-23	40-36-22	34-39-27	55-34-11	47-33-20	62-33-5	69-28-3	78-21-1
Using DPRE – Dynamic Positioning and Role Exchange								
Goals (Scor.-Conc.)	9-6	12-1	30-7	91-0	81-1	145-0	209-0	236-0
Wins–Draws–Loses	4-3-3	7-3-0	8-1-1	10-0-0	10-0-0	10-0-0	10-0-0	10-0-0
Shoots	23-12	22-6	42-18	132-3	108-10	196-1	264-0	256-0
Attack–Middle-Def.	31-51-18	48-33-19	38-37-25	60-30-10	53-34-13	68-29-3	74-25-1	77-22-1

Table 2. Results Achieved with and without using DPRE

8. Conclusions

FC Portugal research on coordination methodologies enabled the definition a model for the strategy of team for a particular soccer game. This model is then implemented over SBSP and DPRE, enabling the team to perform in a very efficient way, as shown by our controlled experiments and competition results, and also in a real-soccer like manner. The development tools that enabled the tuning of the cooperative team behaviour and individual decision and individual skills mechanisms have been presented. These tools are based on general principles that can be of applied to the development of several types of agent in different domains.

FC Portugal achieved more than 15 awards in international RoboCup competitions. These awards included wining the World championships of the 2D simulation league (in 2000), Coach Competition (in 2002) and 3D simulation league (in 2006) and several European championships (including RoboCup Rescue). Its participation in the very recent World Championship of the Physical Visualization league resulted in the 2nd place (in 2007). The collaboration of FC Portugal members with CAMBADA and 5DPO Middle-Size teams resulted in a technology transfer from the simulation league to the real robots leagues, as, for example, the coordination model of the CAMBADA team (showed in the RoboCup 2007 free challenge) directly based in SBSP.

9. References

AGEIA (2007), AGEIA PhysX , http://www.ageia.com/physx/

Azuma, R. (1997), A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments* Vol. 6, No. 4 (August 1997), pp. 355 – 385.

Boost (2004), Boost C++ Libraries, http://www.boost.org

Chen, M;, Foroughi, E.; Heintz, F.; Huang, Z.; Kapetanakis, S.; Kostiadis, K.; Kummeneje, J.; Noda, I.; Obst, O.; Riley, P.; Steffens, T.; Wang, Y. & Yin, X. (2007), RoboCup Soccer Server Manual, http://downloads.sourceforge.net/sserver/manual.pdf (2007). The RoboCup Official Web Site, http://www.robocup.org/

Donaldt, B.; Levey, C. ; McGray, C.; Rus, D. & Sinclair M. (2003), Power Delivery and Locomotion of Untethered Micro-Actuators, *Journal of Microelectromechanical Systems*, Vol. 12, No. 6 (2003), pp. 947-959

Donald, B; Levey, C.; McGray, C.; Paprotny, I. & Rus, D. (2006), An Untethered, Electrostatic, Globally Controllable MEMS Micro-Robot, *Journal of Microelectromechanical Systems*, Vol. 15, No. 1 (2006), pp. 1-15

Ferreira, R.; Reis, L.P. & Lau, N. (2004), Situation Based Communication for Coordination of Agents, Proceedings of Scientific Meeting of the Portuguese Robotics Open 2004, Reis, L.P. (Ed.) pp.39-44, ISBN 972-752-066-9, April 2004, FEUP Edições, Porto, Portugal

Expat (2004), The Expat XML parser, http://expat.sourceforge.net

Kitano, H. et al. (1997), Robocup: The Robot World Cup Initiative, Proceedings of 1st International Conference on Autonomous Agent (Agents97), 1997, The ACM Press, Marina del Ray.

Lau, N. & Reis L.P. (2002), FC Portugal 2001 Team Description: Configurable Strategy and Flexible Teamwork, In: *RoboCup-2001: Robot Soccer World Cup V*, Birk, A.; Coradeshi, S. & Tadokoro, S. (Ed.), Springer Verlag, Berlim, LNAI 2377

LIRC (2007), Linux Infrared Remote Control, http://www.lirc.org/

Mackay, W. & Gold, R. (1993), Computer augmented environments: Back to the real world, *Commun. ACM*, Vol. 36, No.7 (July 1993), pp.24--26,

McGray, C.; Arai, F.; Jacoff, A.; Tadokoro, S. & Gaitan, M. (2007), RoboCupSoccer – Nanogram Competition – White Paper, on line, available at http://www.eeel.nist.gov/812/nanogram/white_paper.pdf

Microsoft (2007), Microsoft Robotics Studio, http://www.microsoft.com/robotics.

Milgram, P. & Kishino, F. (1994), A Taxonomy of Mixed Reality Visual Displays, *IEICE Transactions on Information Systems* ,Vol. E77-D, No. 12 (December 1994), pp. 1321-1329

Obst, O. & Rollmann, M. (2005), Spark - A generic simulator for physical multi-agent simulations. *Computer Systems: Science & Engineering*, Vol. 20, No. 5 (September 2005)

Reis, L. P.; Lau, N. & Oliveira, E. C. (2001), Situation Based Strategic Positioning for Coordinating a Team of Homogeneous Agents, In: *Balancing Reactivity and Social Deliberation in Multiagent Sytems: From RoboCup to Real Word Applications*, Hannenbauer, M.; Wendler, J. & Pagello, E., (Ed.), pp. 175-197, Berlin Springer-Verlag LNAI 2103.

Reis, L.P. & Lau, N. (2001), FC Portugal Team Description: RoboCup 2000 Simulation League Champion, In: *RoboCup-2000: Robot Soccer World Cup IV*, Stone, P., Balch, T., & Kraetzschmar, G., (Ed.), pp. 29-40, Berlin Springer-Verlag, LNAI 2019.

Reis, L.P. & Lau, N. (2002), COACH UNILANG - A Standard Language for Coaching a (Robo)Soccer Team, In: *RoboCup-2001: Robot Soccer World Cup V*, Birk, A.; Coradeshi, S. & Tadokoro, S. (Ed.), pp. 183-192, Springer Verlag, Berlim, LNAI 2377

Riedmiller, M. & Merke, A. (2002), Karlsruhe Brainstormers - a reinforcement learning approach to robotic soccer II, In: *RoboCup-2001: Robot Soccer World Cup V*, Birk, A.; Coradeshi, S. & Tadokoro, S. (Ed.), Springer Verlag, Berlim, LNAI 2377

Riley, P. (2003), SPADES: System for Parallel Agent Discrete Event Simulation, *AI Magazine*, Vol. 24 No. 2, pp. 41–42.

Riley, P. (2003a), SPADES for Parallel Agent Discrete Event Simulation, http://spades-sim.sourceforge.net/

RoboCup (2007a). Physical Visualization Sub-League Official Web site. http://wiki.cc.gatech.edu/robocup/index.php/Physical_Visualization_Sub-League

RoboCup (2007a), Site of Osaka University's Physical Visualization League: http://er04.ams.eng.osaka-u.ac.jp/ecobe-robocup/

Robosoft (2007), RobuDOG simulator for the Robocup http://www.robosoft.com

Ruby (2004), Ruby Home Page http://www.ruby-lang.org

Sedaghat M. & al. (2003), Caspian 2003 Presentation Description,

Smith, R. (2006), Open Dynamics Engine v0.5 User Guide, http://opende.sourceforge.net/,

Stone, P. & Veloso, M. (1999), Task Decomposition, Dynamic Role Assignment, and Low-Bandwidth Communication for Real-Time Strategic Teamwork, *Artificial Intelligence*, Vol. 110, No. 2, (June 1999), pp. 241–273.

Stone, P.; Riley, P. & Veloso, M. (1999), CMUnited-99 source code, 1999, Date Accessed: July 2007, Accessible from http://www.cs.cmu.edu/~pstone/RoboCup/CMUnited99-sim.html.

Stone, P. (2000), *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*, MIT Press, ISBN: 0262194384.

Stone, P.; Riley, P. & Veloso, M. (2000), Layered Disclosure: Why is the agent doing what it's doing?, Proceedings of Fourth Int. Conf. on Autonomous Agents (Agents 2000), 2000, Barcelona.

Teixeira, C.; Lau N. & Reis, L.P. (2004), FC Portugal 2003 Shoot Evaluation Based on Goalie Movement Prediction, Proceedings of Scientific Meeting of the Portuguese Robotics Open 2004, Reis, L.P. (Ed.) pp.149-155, ISBN 972-752-066-9, April 2004, FEUP Edições, Porto, Portugal

Yanagimachi, S. & Guerra, R. (2007), Citizen Micro Robot´s Reference Manual. Osaka University. http://er04.ams.eng.osaka-u.ac.jp/ecobe-robocup/files/robot-manual.pdf

Yao, J.; Chen, J.; Cai, Y., Li, S. (2002), Architecture of TsinghuAeolus, In: *RoboCup-2001: Robot Soccer World Cup V*, Birk, A.; Coradeshi, S. & Tadokoro, S. (Ed.), pp. 491-494, Springer Verlag, Berlim, LNAI 2377

10

Desktop Robot Soccer

Frederic Maire, Joaquin Sitte and Narongdech Keeratipranon
Faculty of Information Technology, Queensland University of Technology
Australia

1. Introduction

Robot soccer pits teams of fast-moving robots in a dynamic environment (Sng et al., 2002). Robot soccer fosters AI and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined (Asada & Kitano, 1999). Today two international robot soccer federations, *RoboCup* (RoboCup, 2007) and *FIRA* (FIRA, 2007), organize competitions in an eclectic range of categories. Those competitions are accompanied with technical conferences. The first international robot soccer tournament MiroSot'96 was held at Korea Advanced Institute of Science and Technology (KAIST), in November, 1996. At the time of writing, we can count more than ten different robot soccer leagues from RoboCup and FIRA.

Taxonomy of the robot soccer leagues could start with the vision system used. The *global vision* group contains all the leagues that allow a global vision system (camera that gives an eye-bird view of the playing field). The image processing is done on a PC that controls the robots via a radio link. Whereas the *local vision* group contains all the leagues that require the vision processing to be done on the robots themselves. In this second group, the robots achieve a higher level of autonomy. Only wheeled robots are used in the global vision group. Whereas, the local vision group can be subdivided into wheeled robots and legged robots. Finally there are simulation leagues that provide a test bed for multi-agent research for those who do not have access to real robots.

Robot soccer not only stimulates robotic research, but also provides a platform for computational intelligence education that allows the development of engaging undergraduate level assignments. However, there are several limiting factors for the widespread use of robot soccer as a research platform or a teaching tool. Most robot soccer leagues like the popular RoboCup Small Size and FIRA Mirosot leagues require a large playing field and a team of several postgraduate students to build the hardware and develop the complex software. The least resource-demanding robot soccer league is the simulation league. Unfortunately, by its very nature, this league does not provide the invaluable experience of real robots. With the constraint of using real robots, the least resource-demanding robot soccer league is arguably the FIRA KheperaSot league. This league represents *Desktop Robot Soccer*, in the sense that the playing field fits on a desktop or a computer laboratory bench.

The regulations of the KheperaSot league impose size restrictions on the robots. The size limitation lowers the entry barrier for participants in relation to other robot soccer

tournaments, making it more accessible to individuals and small teams with modest funding and infrastructure support. The size limitation also poses challenge for hardware technology. It pushes the limits of how much processing and sensing can be put into the small package at a reasonable cost. However the size is not as small as to requiring miniaturisation technology beyond the reach of standard electronics and construction techniques. The KheperaSot league was the first fully autonomous robot soccer league of FIRA.
Section 2 provides an overview of KheperaSot league. Section 3 describes the winner of the 2003, 2004 and 2005 KheperaSot World Cups. Section 4 discusses how robot soccer can be used in the undergraduate curriculum. Section 5 concludes the paper.

2. KheperaSot League

The KheperaSot league has its origin in the 1997 Danish Robot Soccer Championship organised by Henrik Hautop Lund (Lund, 1999). The Khepera robot (Fig. 1) is a two-wheeled cylindrical robot with a diameter of 70 mm, equipped with a ring of 8 IR proximity sensors, wheel encoders and a linear camera turret (Fig. 2) that produces a horizontal linear image of 64 pixels with 256 grey levels. These 64 pixels allow the detection of the ball (a yellow tennis ball), the goal (large black zone), and the opponent robot (wearing a black and white stripped shirt).
The main difficulties of the KheperaSot league reside in the limited computational resources (512K of memory) and the low resolution of the linear camera.

Fig. 1. Khepera II robot

The KheperaSot playing field is 105 centimetres long and 68 centimetres wide (see Figure 3) and is surrounded by grey walls. The goals are openings in opposite walls and are painted black inside. Both goals look identical. A match consists of five rounds of at most four minutes each. A round ends when a goal is scored or when the ball does not move for thirty seconds. The team that scores the largest number of goals is declared the winner. At the beginning of a round the ball is placed at the centre of the field. The players are positioned differently at the start of each round.

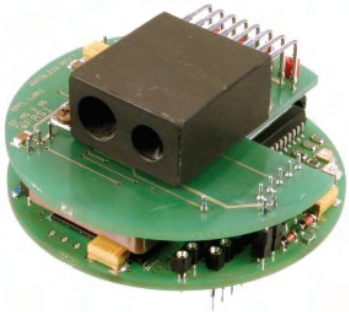


Fig. 2. Linear camera turret

The referee points out 180-degree rotation symmetric starting positions. Each player starts facing its opponent's goal line. A starting position in the opponent's half is possible. The KheperaSot environment is shown in Figure 4.

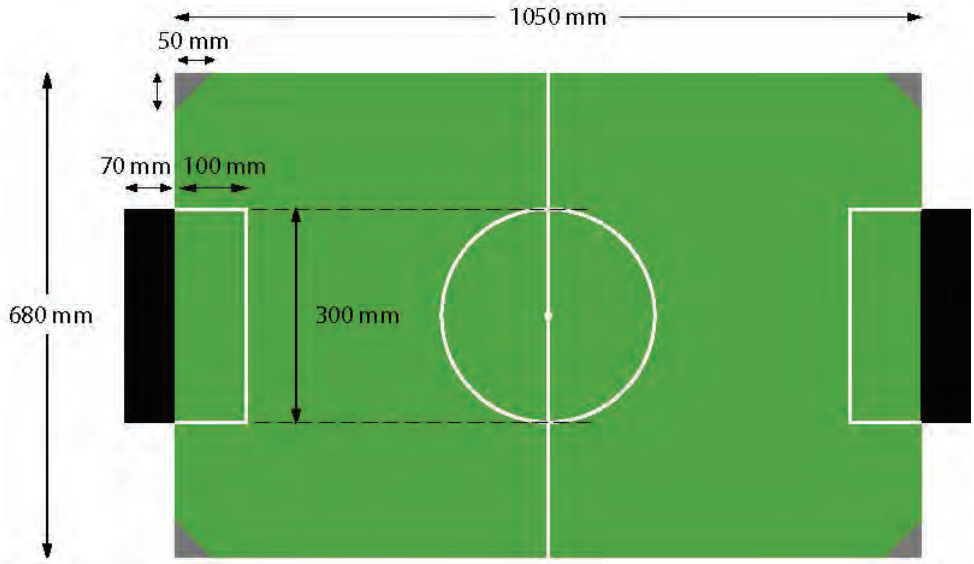


Fig. 3. KheperaSot's floor plan

Unlike leagues allowing a global vision system, the KheperaSot robot has to find it own position in a completely symmetric environment. The only information that the robot can exploit is that it is facing the opponent's goal line at the start of each round. But if the robot get disoriented in the course of a round, it is impossible to determine which goal is the opponent goal from visual clues. Apart from the intrinsic limitations of the odometers, pushing by the opponent can create further odometric errors. It was not uncommon in the first years of the competition to see confused robots score own goals.

The cylinder shape of the robot and the grooves on the tennis ball makes dribbling the ball a challenging task. During a game, the ball might be pushed into a corner. It is not a trivial task to unstuck the ball from the corner.

Fig. 4. KheperaSot's arena

One of the most attractive features of the KheperaSot league is its relatively low cost. Robot soccer leagues which require many robots and a large field such as RoboCup – Small Size League, have budgets of at least 30,000 US$ for the hardware (Peel, 2003). KheperaSot requires a much smaller budget. Moreover the playing field can fit on a desktop. A single person can look after a KheperaSot system, whereas the other leagues have typically teams of half-a-dozen people.

3. Description of Kheperoo

QUT's entry in the KheperaSot league, called *Kheperoo*, has won three KheperaSot World Cups in a row (2003, 2004 and 2005) before retiring. In this section, we give an overview of the system and describe the strategy used.
The software architecture used is a finite state machine with multiple threads running concurrently. Kheperoo's top priority in the game is to get to the ball first and move the ball away from the opponent. If Kheperoo manages to move ball away from the opponent's vision field, the opponent will need some precious time to locate the ball again. During that time, our robot can take advantage of the opponent confusion and push the ball towards the opponent line. Kheperoo deliberately does not try to head for the goal directly, but simply the opponent line. The rationale behind this decision is that the opponent is more likely to be in between its goal and the ball because of the symmetry of the starting condition.

After racing to the ball, Kheperoo dribbles the ball towards the opponent's goal line based on the estimated orientation provided by the wheel-encoders (they play the role of a virtual compass for a short period of time).

If Kheperoo is lucky, the ball may end up directly in the opponent's goal. Most of the time, the ball will get stuck against the opponent's wall. This situation triggers a complex behaviour to push the ball into the opponent goal. An overview of Kheperoo finite state machine is shown in Figure 5.

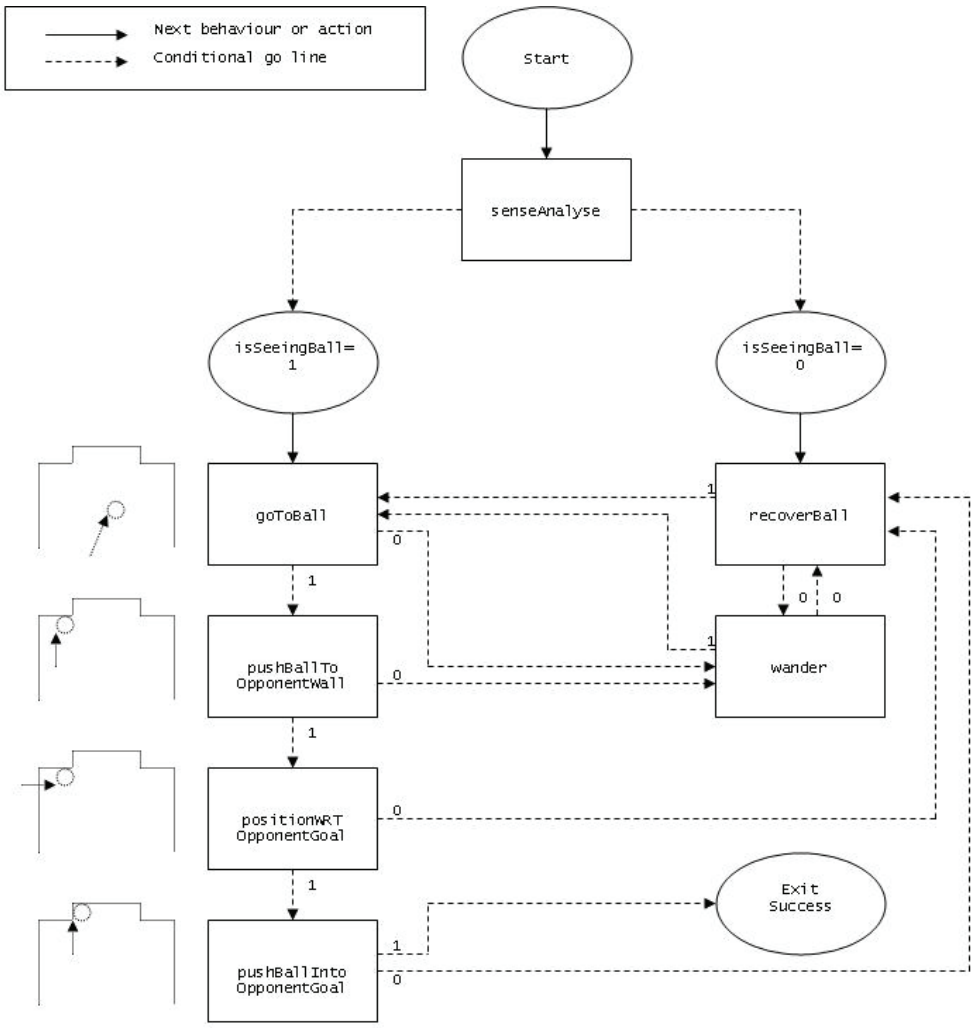


Fig. 5. The finite state machine of Kheperoo

Apart from the main control thread, a watchdog thread is used to monitor the robot's wheels' status. The robot has to be able to detect and stop when it runs against a wall to avoid the wheel slippage problem and prevent any damage to the robot itself. The actual wheel speeds and desired wheel speeds are compared to determine whether a static obstacle is in the way.

To alleviate the problem of accumulative odometry error, Kheperoo will reset its pose when it knows the actual direction base on some specific situations. When the robot moves steadily along the side wall, its direction must be either facing opponent or facing own goal lines, this information and its current pose based on odometry can be used to determine its actual direction. The robot can also calibrate its direction using proximity sensor when it stops in front of the opponent wall.
Complex behaviours such as *unstuck the ball from the corner* also use the watchdog to complete their tasks. To unstuck the ball, first, the robot will position itself carefully with respect to the ball, then push the ball straight to the wall until some resistance is felt. After that, the robot will spin on itself to unstuck the ball. Hopefully, the ball will roll out from the corner.
For dribbling the ball, we use the fact that it is more effective to control the ball direction when the ball is rolling because the ball already has some momentum. One method to make the ball roll straight away before the robot starts dribbling is to give a strong kick to the ball. After the kick, the ball usually rolls in a straight line and the robot can continue to forward dribble the ball. In some case, the ball does not roll in a straight direction and the robot needs to circle around the ball to recover its dribbling direction. But at least after the kick, the ball has moved closer to the opponent side.
We attribute the success of Kheperoo in competition to its speed, robustness to sensor noise and ability to handle deadlock situations.

4. Experiences with desktop robot soccer as a teaching tool

The development of practical sessions and assignments for undergraduate teaching typically requires a compromise between what is achievable by an average student and what engages the interest of a more advanced students in the class. Selecting a suitable compromise is particularly problematic for undergraduate computational intelligence (CI) teaching units which typically attempt to cover a very broad range of AI and machine learning techniques. At our university, the curriculum of the CI unit includes topics like fuzzy logic, neural networks, reinforcement learning and genetic algorithms. We believe that the undergraduate students should experience the non-deterministic characteristics of a real robot environment (as opposed to a simulated one). Students should focus on the interaction of a single robot with its environment before dealing with scenarios that require a multi-agent system. In other words, students should familiarize themselves with sensory noise, motor characteristics and uncertainty before trying to coordinate complex robot behaviours for a team play in a multi-agent framework.
A challenge with teaching abstract concepts is finding illustrative examples for the ideas presented. That is, finding real problems to be solved with the abstract methods provided in the lectures. In the prevailing teaching paradigm, the lecturer instructs the student on how to solve a problem step by step. The lecturer also tells the student what problems are solvable with each problem-solving method. The drawback with this approach is that the lecturer performs the greatest part of the cognitive processing. This is why we have followed a problem based approach. The goal of the assignment is formulated and given to the students who should find suitable solutions to the problem by themselves. It is up to the students to decide which of the CI techniques covered in the lectures are relevant to their problem. In this context of problem based teaching, the lecturer and the tutors become

coaches for groups of students. In problem based learning, it is very important to make goals, assignments and expectations on the students as clear and exhaustive as possible (Ambury, 1992). If the requirements are unclear the students might lose interest in the course or do something totally different than what was expected by the teachers.
Below is the weekly task schedule of the laboratory sessions. The first sessions are relatively structured and served as an introduction to mobile robotics to most of the students (Keeratipranon et al., 2003).

- Session one; get an idea on how we can communicate with the robot
    a. Install KTProject (software to compile and download executables to the Khepera robot)
    b. Get familiar with Tera Term Pro (terminal emulator for serial communication with the Khepera robot)
    c. Communicate with the robot using Matlab
    d. Modify the *obstacle avoidance*, Braitenberg vehicle project such that the robot avoid the black area (robot's IR proximity sensor cannot detect black object as black colour does not reflect well the IR signal back to the robot) using KTProject
    e. Implement a *wall following* behaviour.
- Session two; provide students with a template example
    a. Reverse engineer the template; student will know how to
        1. Read and use proximity sensor values
        2. Control a robot using the keyboard and serial communication
        3. Create a multi threaded program (multi tasking)
        4. Detect that the robot is stuck
        5. Estimate the direction of the robot base on the left and right wheel encoders.
    b. Create a new project using a given template.
    c. Interact with the robot via keyboard control
    d. Modify the *wall following* function so that it can follow a T-shape object.
    e. Brainstorm about what basic behaviours are required in order to create a robot soccer player.
- Session three; implement some basic behaviours
    a. Continue the exploratory work on proximity sensors.
    b. Finish the *obstacle avoidance* and *wall following* behaviours.
    c. Implement a *ball following* behaviour. In this behaviour, the robot will follow the moving ball using only proximity sensors while the ball is in front of the robot at a short distance and moved by a person.
    d. Implement a *circle around the ball* behaviour. In this behaviour, the robot will move around a static ball.
- Session four; implement more complex behaviours
    a. History shows that many teams will not have completed the *wall following*, *ball following*, or *circle around the ball* behaviours. Therefore, this session gives more time for the slower teams to catch-up.
    b. Implement a *dribble the ball* behaviour. In this behaviour, the robot will push the ball forward in a straight direction. The *pushing ball* behaviour is one of the

milestones for the assignment. Teams that cannot demonstrate this behaviour on the real robot by the deadline suffer a penalty in their final marks.

- Session five; milestone week and use of a new sensor: the linear camera
    a. At this stage of the semester, every team should have mastered the proximity sensor well enough to demonstrate a dribbling behaviour.
    b. Get familiar with the linear camera.
    c. Design a *look for the ball* and a *go to ball* behaviours.
- Session six; vision sensor
    a. Implementation of a *look for the ball* behaviour. The aim of this behaviour is to locate the direction of the ball. A tennis ball can be differentiated from other objects as it has a bright colour compared to the wall or the goal.
    b. Implementation of a *go to ball* behaviours. After locating the ball, the robot has to be able to move to the ball. The desired property of this behaviour is speed and reliability.
- Remaining three sessions; free style development
    a. The development of all other behaviours is left to the initiative of the students. They have to decide which of the techniques presented in the lecture they will use (if any). The tutors employ at this stage a problem based learning approach to guide the student groups.
- Final session; friendly competition
    a. Each team has the opportunity to showcase the result of their effort in the end of the semester KheperaSot competition. The assignment marks do not depend directly on the competition results, but observations made during the games are taken into account for some marking criteria.

Outside the laboratory sessions, the students have access to a KiKS, Khepera simulator (Nilsson, 2001).

5. Discussion

As discussed above the KheperaSot in its current form poses challenges in motion control, navigation and self-localisation, as well as in higher level autonomous behaviour design and implementation.

The reason why we support and encourage the KheperaSot league is the complete autonomy of the robot combined with its small size. The autonomy enhances the educational value of the tournament by putting it on par with prospective autonomous mobile robot applications that have to rely entirely on their own sensors to acquire information of the world around them. The autonomous nature of the KheperaSot league also provides a natural evolutionary pathway for the game, allowing it to maintain challenges as the technology and the experience of the players advance.

The size limitation lowers the entry barrier for participants in relation to other robot soccer tournaments, making it more accessible to individuals and small teams with modest funding and infrastructure support. The size limitation also poses challenge for hardware technology. It pushes the limits of how much processing and sensing can be put into a small package at a reasonable cost. However the size is not as small as to require miniaturisation technology beyond the reach of standard electronics and construction techniques.

We have successfully integrated Khepera robot soccer into our computational intelligence course. The laboratory sessions are organized in a standard computer laboratory equipped with rows of computer benches. At the beginning of a session, we bring in the soccer fields. Students have demonstrated that they gain understanding of the many issues with embedded systems programming such as the real time aspects; interrupt driven programming, and completely different program debugging resources.
Student's programming skills have been expanded from the experience with real robots. Students can have a visual feedback through the displayed behaviour of the robot. Unexpected situations have happened such as a failure to track the ball while the robot is moving fast to the ball. This failure is not necessarily due to the low frame rate of the camera but can be due to the backward tilt of the robot with the acceleration (the ball goes below the horizon). These types of real-world problems are not experienced with a simulator.
The most obvious extension of the current KheperaSot is the replacement of the current 1D vision by a 2D colour camera. Digital image sensors of the type used in mobile phones are cheap and widely available, some of them include a microprocessor for low level image processing. Two solutions are already available for the Khepera: the adaptation of the CMUcam by K-team (K-team, 2007) and the 2D camera developed by the Paderborn team (Chinapirom et al., 2004). Both allow vision image processing on the robot. Upgrading the vision system will provide the enriched realism of the game without a large increase in cost. With a 2D vision system the KheperaSot league will have all the features of the humanoid and AIBO leagues, which are all autonomous, with the added complication of legged locomotion. For some time to come, wheeled locomotion will allow much faster games than the legged leagues resulting in more interesting games. Most importantly, 2D vision will facilitate multi-player teams and thereby largely expanding the opportunities for collaborative strategies. An expansion to 3 player teams seems feasible with a moderate enlargement of the playing field and without fully loosing the desktop characteristic of the KheperaSot league. One may argue that increasing the number of players per team also rises the cost. However, because of the autonomy of the robot, teams could be formed by students of different institutions each providing their own robots. Our ultimate vision would be to bring the cost to a level where individual enthusiast could buy their own pocket sized soccer robot for participating in a school or neighbourhood team.
Up to now the league uses Khepera robots, hence its name; however the rules do not exclude other manufacturers. As more powerful small size and low power single board computer (SBC) come on the market at low prices, such as the Gumstix (Gumstix, 2007) boards. We expect alternative robots to be built for the game.

6. References

Ambury, G. (1992). Beginning to Tutor Problem-Based Learning: A Qualitative Investigation of Andragogy in Medical Curriculum Innovation, *Proceedings of Annual meeting of the CASAE*, Ottawa, Canada, June 1992

Asada, M. & Kitano, H. (1999). The RoboCup Challenge, *Robotics and Autonomous Systems,* Vol. 29, No. 1, page numbers (3-12)

Chinapirom, T.; Kaulmann, T.; Witkowski, U. & Rauckert, U. (2004). Visual object recognition by 2D-color camera and on-board information processing for minirobots, *Proceedings of FIRA Robot World Congress*, Busan, South Korea, 26 – 29 October 2004

Federation of International Robot-soccer Association (2007). http://www.fira.net/ , accessed on 24 July 2007

Gumstix Inc (2007). Gumstix - way small computing, 2007. http://www.gumstix.com/, accessed on 24 July 2007

Keeratipranon, N.; Sitte, J. & Maire, F. (2003). Beginners Guide to Khepera Robot Soccer, Brisbane, Australia, Queensland University of Technology., http://www.fira.net/soccer/kheperasot/Guid.pdf

K-team (2007). CMUcam Vision Turret http://www.k-team.com/kteam/index.php?site=1&rub=3&upPage=108&page=17&version=EN , accessed on 24 July 2007

Lund, H. H. (1999). Robot Soccer in Education, *Advanced Robotics Journal,* Vol. 13, No. 8, page numbers (737-752)

Nilsson, T. (2001). KiKS is a Khepera Simulator, Stockholm, Sweden, Umea University.

Peel, A. (2003). The Robots: RoboCup Used for Research and Teaching, *Proceedings of Autonomous Minirobots for Research and Edutainment (AMiRE)*, pp. 173-182, 18-20 February

Sng, H. L.; Sen Gupta, G. & Messom, C. H. (2002). Strategy for Collaboration in Robot Soccer, *Proceedings of The First IEEE International Workshop on Electronic Design, Test and Applications*, pp. 347-351

The RoboCup Federation (2007). http://www.robocup.org/, accessed on 24 July 2007

11

Embedded Behavioral Control of Four-legged Robots

David Herrero Pérez and Humberto Martínez Barberá
University of Murcia
Spain

1. Introduction

The problem of creating a four-legged robotics football team is a very difficult and challenging problem. Regardless of the hardware design and manufacture, there are several fields involved, like low level locomotion, perception, location, behavior development, communications, etc., which should be addressed for developing a fully functional team. In practical terms, this means that the software project to develop a robotics soccer team can be very large, which implies that verification, debugging and monitoring tools are needed and play a very important role in software development time (which uses to be a very expensive resource).

This work is focused on the architecture and behavioral programming model we use to develop a team in the Sony Four-Legged League, which is one of the official leagues of the RoboCup. All the code, examples, and tools we present in this work have been developed for the **TeamChaos** team[1], which has participated in the 2004, 2005 and 2006 editions of the RoboCup and several international competitions, and is a follow up of the former **TeamSweden** team, which has participated in the previous editions (1999 to 2003). In this league, all the teams must use the same physical platform, in particular the commercial four-legged robot AIBO developed by Sony. The most recent model, *AIBO ERS-7*, integrates one CPU (64-bits RISC Processor 576MHz), audio interfaces (speaker and stereo microphones), switch sensors, two infrared distance sensors and a CCD camera (350K pixels) as exteroceptive sensors, accelerometers as internal sensors and wireless LAN for communications.

In the 2007 RoboCup Edition, the teams of four-legged league consist on four robots, which have to operate fully autonomously, i.e., there is not external control, neither by humans nor by computers, thus all the processing must be done on board and for practical reasons it has to be performed in real time, which prevents us from using time consuming algorithms. Because of the fact that the hardware platform is standard, we can consider this as a software only league. The field is also standard; there are many color coded objects in the

[1] http://www.teamchaos.es

field to make easier the sensing, like ball, goals and beacons, and each team wears a colored uniform. However, in a real soccer field there are not characteristic colored cues, therefore, rules of RoboCup are gradually changed year after year in order to push progress towards the final goal, "*by the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team*". The official platform and field of this league for the RoboCup 2007 Edition is shown in Fig. 1.

Fig. 1. Official platform (left) and field (right) of four-legged league in 2007 RoboCup Edition

This league proposes a very demanding scenario, with high uncertainty in perceptions and limited processing capabilities. In addition, having a competition implies that at certain periods the software development and tuning presents high activity peaks. These facts condition the way robots have to be programmed. Our approach to this problem is twofold: we define a software architecture into which all the different modules plug, and we use a programming language that can drastically reduce development time for certain modules, in particular all behavior related modules.

We follow the ThinkingCap architecture (Saffiotti et al., 1995), a two-layer architecture which clearly reflects a cognitive separation of modules. From the conceptual point of view, modules are arranged by the nature of their processing tasks. From a software point of view, the interfaces are clear and well defined, so that replacing or improving modules is not a demanding task. Our current instance of the architecture makes extensive use of the behavioral paradigm, and for implementing those behaviors we have opted for the LUA language (Ierusalimschy et al., 1996). LUA is a simple yet powerful embedded language with a quite portable interpreter. We have integrated LUA in the architecture and have developed a set of tools for the on line edition, monitoring and debugging of control programs. This allows us to develop and modify behaviors while testing robots on the playing field at runtime. Moreover, the behaviors can be tested and verified before the execution on the real robot using those software tools, which dramatically reduces the severity and duration of downtime during development time. In order to simplify and reuse behaviors, we organize them in two types depending on the complexity and functionality:

low-level behaviors, which perform specific actions (go to ball, look for ball, kick ball, etc), mostly reactive, and high-level behaviors, which perform high level tasks (defend, attack, pass, penalty shootout, etc), mostly deliberative. These usually imply the use of some form of state, and we have opted for the Hierarchical Finite State Machine (HFSM) paradigm (Hugel et al, 2005) to design them. We have developed a visual editor for HFSMs (based on Hugel's code), which automatically generates LUA code directly executable by our architecture.

This chapter is organized as follows. Section 2 describes the control architecture. Section 3 describes the LUA language and how it has been incorporated into the architecture. Section 4 and 5 describe low-level and high-level behaviors respectively. Section 6 presents some conclusions and future work.

2. Control Architecture

2.1 The ThinkingCap Model

The architecture used by each robot is an instance of the ThinkingCap architecture (Saffiotti et al., 1995). Fig. 2 shows the main elements of this layered architecture; the lower layer provides the interface to the actual hardware, the middle layer maintains a consistent representation of the environment around the robot and provides the reactive robot control, the higher layer maintains the representation of the objects in a world frame and performs decisions considering the global information, and the communication layer provides the interface to share information with the other members of the team.

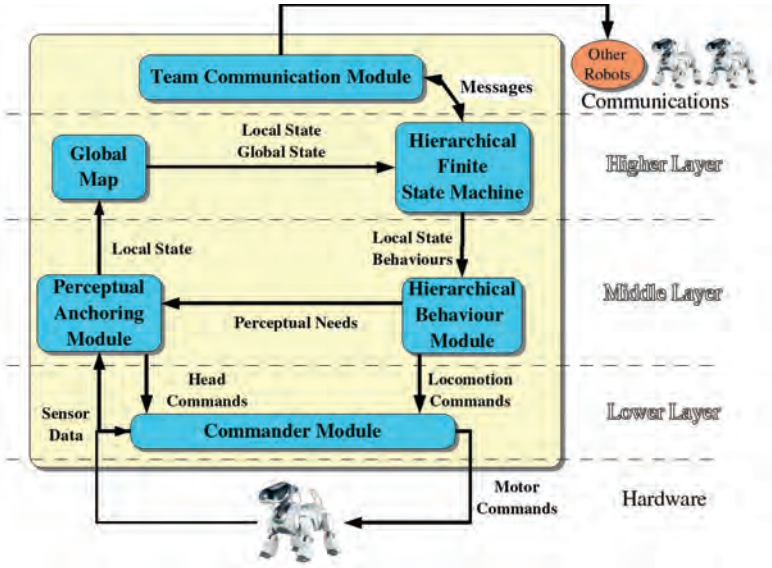


Fig. 2. Instance of the Thinking Cap architecture for the four-legged league

The **lower layer** provides an abstract interface to the sensori-motor functionalities of the robot (Commander Module – CM). This module accepts abstract commands from the upper layer, and implements them in terms of actual motion of the robot effectors. In particular, it receives set-points for the desired linear, lateral and angular velocity, and translates them to an appropriate walking style by controlling the individual leg joints.

The **middle layer** maintains a consistent representation of the space around the robot (Perceptual Anchoring Module - PAM), and implements a set of robust tactical behaviors (Hierarchical Behavior Module – HBM). The PAM acts as a short-term memory of the location of the objects around the robot: at every moment, this module contains an estimate of the position of these objects based on a combination of current and past observations with self-motion information. The PAM module is also in charge of camera control, by selecting the fixation point according to the current perceptual needs (Saffiotti & LeBlanc, 2000). The HBM implements a set of navigation and ball control behaviors.

The **higher layer** maintains a global representation of the field (Global Map - GM) and makes real-time strategic decisions based on the current game state, situation assessment and role selection (Hierarchical Finite State Machine - HFSM). Self-localization in the GM is based on fuzzy logic (Buschka et al., 2000), (Herrero-Pérez et al., 2004). The HFSM implements a behavior selection scheme based on finite state machines (Hugel et al, 2005). In addition, global ball sharing is also based on fuzzy logic (Cánovas et al, 2004).

Radio **communication** is used to exchange position and coordination information with other robots (Team Communication Module - TCM) using customs protocols over UDP/IP.

Intercommunication between the different modules is implemented by data structures interchange. The two more important data structures are those representing the world state, either in a local or global frame. The information stored in these structures can be related to either static objects (nets, landmarks) or dynamic objects (ball, teammates, opponents). The local state represents the objects that have been recently perceived by the robot camera in a robot centric frame. The data structure that represents the local state is called *Local Perceptual Space* or LPS (Saffiotti et al., 1995), which consists on an array of Local Perception Objects or LPO. Each one of these LPOs includes polar positioning information (ρ and θ) and an anchoring value, which somehow represents the reliability of that precise perception and decreases over time (Saffiotti & LeBlanc, 2000). At each control cycle, all the objects that have not been re-perceived are rotated and translated according to the odometry estimation. The global state represents all the objects that can be in the environment in a world frame, in our case the field viewed from the goalkeeper position. The data structure that represents the global state is called *Global State* or GS, whose contents is the result of fusing information from different sources (time aggregation of own camera perceptions and/or other robots global states). The most important information for soccer game play is that related to self-positioning (implemented by filtering the different camera perceptions and odometry estimates) and ball positioning (implemented by filtering the local position of the ball from the different team robots and their absolute position estimations). The first one is important for zonal behaviors (i.e. keeping the goalkeeper in its own area), while the second is important when the robot does not sees the ball or it is too far to get good distance estimation.

From the control point of view, the behaviors use the information contained in the LPS (typically low-level behaviors of the HBM) and the GS (typically high-level behaviors of the HFSM) to perform reactive or strategic decisions, which finally translate into movement

commands (forward, lateral and angular velocities or kicking actions) or perceptual commands (object necessities or needs). While the former are more or less standard in any soccer-playing robot (and certainly in any four legged robot), the later are specific to this instance of the ThinkingCap architecture (Saffiotti & LeBlanc, 2000). The needs are stored in an array, and they represent the priority to actively look for an object in the environment. The PAM uses this array to determine at each cycle which object it should look for, typically the one with the highest priority. In case that two or more objects share the highest priority, the one selected is the one less recently seen. In any case, when looking for an object, if any other is perceived it is incorporated into the LPS. In addition to this, behaviors can also share information with teammates (ball booking, role information, etc).

2.2 Integration with OPEN-R

The API for programming and debugging code for the AIBO platform, Sony OPEN-R SDK, is merely an interface to develop software for the Aperios OS, which is the real-time operating system used by the entertainment robots of the company. This API discloses the specifications of the interface between the *system layer* and the *application layer*. The software developed using these specifications is object-oriented and modular, where each module is an OPEN-R object. In practice, OPEN-R objects are threads running concurrently with many inter-threads connections which invoke methods of the OPEN-R objects, i.e., the way to manage the concurrency is event-oriented programming. The software to control the robots consists on multiple objects with various functionalities running concurrently and communicating each other via inter-object communication. The programming language supported by this API is C++, including its functionalities.

The Thinking Cap architecture has been programmed using OPEN-R. The natural way to implement the different modules of the architecture is by using one OPEN-R object for each module of the architecture, being the connections between objects implemented by the event-oriented inter-object communication of Sony OPEN-R SDK. Moreover, this implementation provides effective modularization as well as clean interfaces, making it easy to develop different parts of it. Furthermore, it allows the execution of each module in a computer, using RP-OPEN-R (Remote Processing OPEN-R) which is a tool for compiling and executing OPEN-R objects on x86-based CPUs or AIBO robots. For instance, the low level modules can be executed on-board and the high level modules can be executed off-board, where some debugging tools are available. However, due to the real-time constraints, this distributed implementation generates serious synchronization problems, which cause delays in decisions and the robots cannot react fast enough to dynamic changes in the environment.

Because of the reasons above mentioned, we have favored two particular implementations of the Thinking Cap architecture using OPEN-R objects, which we call distributed and monolithic respectively. The **distributed** version is composed of three OPEN-R objects: low level control (ORLRobot), high level control (ORHRobot) and communications (ORTcm). The low level object contains the functionalities of the robot CM and the PAM module in charge of the head or camera, which allows executing this module independently for debugging. The high level object contains all the behavioral part of the system, the HBM and the HFSM, and the GM, the global representation of the field, which allows executing this module off-board receiving or simulating the data from the low level object. The

communications object implements an interface to communicate the robots using customs protocols over UDP/IP. The trade-off between synchronization problems and modularity for debugging is met using this distributed version of only three OPEN-R objects, instead of one object for each module. The **monolithic** version has two OPEN-R objects only: robot control (ORRobot) and communications (ORTcm) objects. The robot control object contains all the functionalities of the robot included in the ORLRobot and ORHRobot modules described above, while the communications object is exactly the same.

In order to maintain both implementations, the different software modules are programmed using standard C++ code, and at compilation time it is decided whether it will be a distributed or a monolithic version. Fig. 3 shows the two possible implementations of the architecture at compilation time, each thread or OPEN-R object contains many software modules of the Thinking Cap architecture. As mentioned above, the distributed implementation allows running modules independently, which facilitates drastically the debugging, and the monolithic implementation avoids the synchronization problems of the inter-object communications of OPEN-R objects, mainly due to concurrency issues.

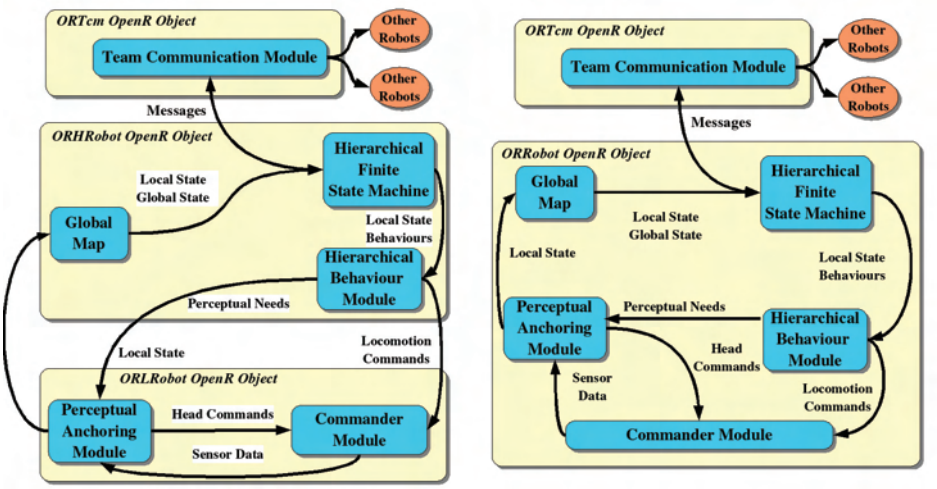


Fig. 3. Implementations of the architecture: distributed (left) or monolithic (right)

3. The LUA Interpreter

3.1 The LUA-based Development Cycle

Because the AIBO exhibits a closed and restricted programming system, the only way to incorporate new software to the robot is by way of OPEN-R objects coded in C++. The typical **on robot** development cycle is as follows: a C++ program is edited and compiled in a desktop computer using a cross compiler for the AIBO CPU. The generated binaries are then written in a memory stick (usually attached to the computer using an USB card reader). The memory stick is then inserted into the robot. Finally the robot is switched on and the program is verified or validated. There are alternatives to this process, some of them

involving using FTP to send the new binaries to the robot, but it is always needed to reboot or switch on the robot, which usually takes a couple of minutes). The typical **off robot** development cycle is much simpler, because the C++ programs can be compiled to the desktop computer CPU and then executed there. Unfortunately, this procedure can only be used to check for the completeness of software modules, but not for the verification and validation of actual robot execution (despite the use of a good simulator, which are not freely available). For these situations the real robot is needed, and the debugging is quite tedious. In particular, this is more exacerbated during competitions, in which during a short time, a lot of behaviors and their parameters have to be modified and checked (and even created from scratch).

In this situation, it is close to be mandatory the use of a series of tools that can boost development and debugging time on robot. Therefore, many teams working with the AIBO platform are using different high-level languages in their behaviors designs. For example, some teams use a reduced version of Perl (Upenn team, from University of Pennsylvania) or Python (UNSW team, from University of New South of Wales) interpreters to implement their high-level behaviors and for rapid development using scripts, which are compiled into intermediate opcodes for efficient performance. Other teams have developed specifics languages to engineering the behaviors of multi-autonomous agents in complex and dynamic environments, like the *Extensible Agent Behavior Specification Language* (XABSL) (Loetzsch et al., 2006). All of these approaches have their pros and cons. XABSL is a XML based language, which without the proper editing tool is very difficult to develop. Current XABSL tools only work with GermanTeam[2] architecture and are very specific for that system. On the other hand, Perl and Python are general use languages, but because of the OPEN-R and hardware constraints, only a reduced interpreter can be used. Perl code readability and maintenance is difficult, and learning curve is quite high. Python is a much cleaner language, but its features (base libraries) are far more complex than what is needed for developing behaviors. In addition Python interpreter footprint is large compared to Perl and other options. For these reasons, we have adopted LUA[3] as the programming language of the behaviors of our architecture. LUA (Ierusalimschy et al., 1996) is a free and open-source multi-paradigm programming language, extremely compact and primarily used as a scripting language or an extension to another language, mainly C/C++. It is primarily considered an extension language although it can be categorized as extensible, interpretive, iterative, logic-based, multi-paradigm, object-oriented, reflective, or as a scripting language. The main characteristic is that it utilizes meta-mechanisms instead of implementing various features directly. This means that the core of the language is fairly restrictive because it is embedded, but it can be extended to include other desired features. Because the language does not include these extensions by default, it avoids the overhead for unused functions, streamlining the code to make it optimal for embedding within another program. Nowadays, this language is primarily used in video games because of its versatility.

In order to enable the LUA interpreter to interact with the sensory and motor routines implemented in the OPEN-R modules, the interpreter was extended so that it is able to call C functions exported by the modules, namely a C extension module for LUA. Basically, this

[2] http://www.germanteam.de

[3] http://www.lua.org

extension module is an interface to access data variables in the OPEN-R routines and activate functions that set sequence of motions, behaviors, team messages, etc. The data variables are the structures defined in the control architecture of the robotic team, which is depicted in Fig. 2. Both the HFSM and the HBM are implemented using LUA scripts with this interface, being the difference the input/output data variables and the activation functions. In the HFSM, the data variables are used to coordinate and make decisions considering the team, the global state of the game and the messages of the teammates, and the decision is the activation of a low level behavior in the HBM. In the HBM, the behaviors only depend on the local state around the robot, being the actions the activation of perceptual needs and locomotion commands, including kicks.

With the use of LUA, the **on robot** development cycle is as follows: a LUA source file is edited in the desktop computer, and then it is sent to the running robot (the development tools make some validation and verification checks, and the file is only sent if it contains no errors). From this moment on, successive calls to the edited script will use the new version. This is clearly a much nicer and helpful approach than the C++ one. In addition, although syntax check is performed, the new code can have potential errors. Because of the lack of LUA pointers, in any case, the new code cannot crash or hang the AIBO CPU, which is a quite common situation while debugging C++ code. In order to simplify code management, each behavior (be it low-level or high-level) is coded in a single LUA source code file

3.2 Integration of the LUA Interpreter

Because of LUA runs by interpreting bytecodes for a register-based virtual machine, we have performed different experiments to evaluate its computational cost because this is a critical point in real-time applications, like robotics soccer is. While interpreting LUA bytecodes is extremely fast, the integration of the LUA virtual machine with the legacy C++ code presents some difficulties. Because we use LUA source files that can be edited and replaced at any time, we do not store the bytecodes. The control cycle of our architecture is quite tight given the hardware platform, and typically both a low-level and a high-level behavior are executed every 100 ms. The time consumed by the interpreter should be minimum in order to leave as much CPU as possible for the vision process (PAM). For the integration we have tried and evaluated three different strategies:

- Each time a script is called it is loaded from the file and then interpreted. This is the simplest approach, and the very first to be implemented. While it is simple, the overhead of interpreting two files every control cycle is too high.
- The first time a script is called it is loaded and kept into memory, and then interpreted at every successive call. The advantage of this approach is that the loading time (accessing the memory stick is quite time consuming) is reduced, but when the behavior is modified the system must signal the interpreter to reload it.
- The first time a script is called it is compiled to bytecodes and the kept into memory, and then executed at every successive call. The advantage of this approach, much like the just-in-time compilers (JIT) standard to the Java world, is that both the loading time and the interpretation time are reduced. In this case it is needed to signal the interpreter when a behavior has been modified to reload and recompile it. This is by far the more

complex approach (from an implementation point of view) and the one that has finally being selected.

Fig. 4 shows a plot of the time in microseconds consumed in several calls to a high-level LUA behavior. The curve labeled as original is the simple approach, and the curves labeled as Optimization 1 and Optimization 2 are the two successive improvements. In average the three approaches consume approximately 40 ms, 17 ms and 7 ms respectively. The variance in execution time is due to the execution time of the other concurrent processes, being it larger when the task consumes more CPU. The measured time is the difference between the end and start times of the interpreter process obtained from the real time clock (RTC). As it can be seen, the third approach is by far the less time consuming (by a factor of nine with respect to the simple approach), and is the one that it is currently in use in our system.

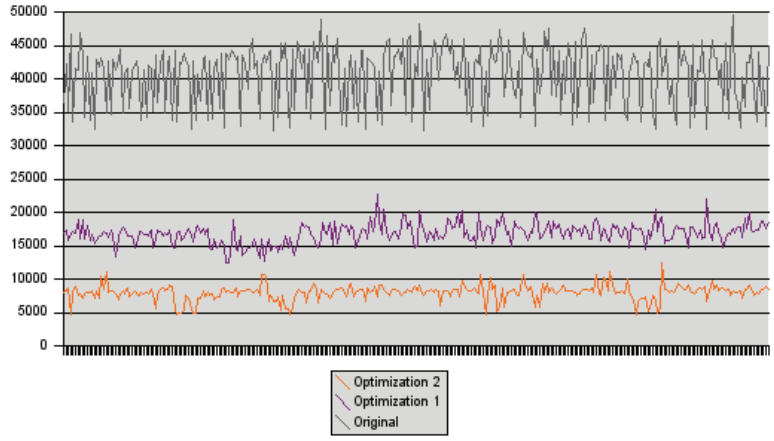


Fig. 4. Execution time (μs) for the three integration approaches of the LUA interpreter

3.3 ThinkingCap Services of the LUA Interpreter

Because behaviors need to access to information provided by other modules of the architecture, we have extended the LUA interpreter to interact with the routines programmed using OPEN-R; in particular, we have implemented a C extension module to call C functions exported by the OPEN-R modules. Basically, the library to interact with the Thinking Cap architecture, named *chaoslib*, is an interface to access data variables in the OPEN-R routines and activate functions that set sequence of motions, behaviors, team messages, etc. We will describe the most important methods of the library for a better understanding of the different examples shown below. The *chaoslib* methods can be grouped into: data access methods, control action methods, coordination methods, and persistence methods.

Data access methods allow accessing all the different information that is generated by the different modules of the architecture, mainly the LPS and the GS. The most important methods are:

- *chaos.lps_getLpo(index).* Returns a table containing the values of the LPO stored at position 'index' of the LPS. A set of convenience constants is available (BALL, NET1, NET2, etc).
- *chaos.gsGetMyPos()*. Returns a table containing the estimation of the location of the robot from the GS.
- *chaos.getBallVel()*. Returns a table containing a vector with the estimation of the velocity of the ball. The goalkeeper typically uses this to decide a defensive action.
- *chaos.getGlobalBall()*. Returns a table containing the estimation of the location of the ball from the GS.
- *chaos.getMates()*. Returns a table containing the estimation of the location of the teammate members.

Control action methods send commands to the other modules of the architecture, like locomotion commands, object necessities and kicking routines activation. The most important methods are:

- *chaos.setVlin(vlin).* Sets the desired robot's linear velocity.
- *chaos.setVrot(vrot)*. Set the desired robot's rotation velocity.
- *chaos.setVlat(vlat).* Set the desired robot's lateral velocity.
- *chaos.setKick(kickid)*. Performs a kicking routine with identifier 'kickid'.
- *chaos.setNeeded(index, value).* Sets the necessity of LPO object at LPS position 'index' to a value of 'value'.
- *chaos.trackLandMarks().* Special method that sets necessities for localization specific objects.
- *chaos.setBehavior(behavior).* Executes a behavior with name 'behavior'.

Coordination methods allow the robot to interact with its teammates. These can be related to the current role of the player, implemented with a dynamic role allocation method (Agüero et al., 2006a), and with the booking of the ball, implemented with a distributed mutual exclusion method (Agüero et al., 2006b). The most important methods are:

- *chaos.getRole()*. Returns a table containing the current role of the robot.
- *chaos.haveBookedBall().* Returns a table containing a truth-value representing the ball booking state for the robot.
- *chaos.bookBall().* Ask the other robots that we want to book the ball.
- *chaos.releaseBookedBall()*. Tells the other robots that we no longer want to book the ball.

Persistence methods allow the behavior to obtain information about previous activations. This is due to the fact that consecutive activations of a behavior imply loading bytecodes into the LUA virtual machine. In this way, a behavior can share with itself some form of behavior state, and know information related to its activation. Data storage is implemented in a hash table. Because LUA does not support types, the global hash table has parameters to

specify data types. Current supported types are string, integer and float. The most important methods are:

- *chaos.getBehaviorInfo()*. Returns a table containing the information about the current behavior. This information includes a behavior execution timer and a flag indicating if it is the first time the behavior is executed.
- *chaos.setGlobal(key, type, value)*. Stores the variable with name 'key' of type 'type' with the value 'value' into the global hash table.
- *chaos.getGlobal(key)*. Gets the value of the variable with name 'key' from the global hash table.
- *chaos.getCurrentState()*. Gets the current state of the game as set by the external referee.

4. Low-level Behaviors

4.1 The HBM Model and Tools

Behavior-based systems are increasingly used in many robotic applications, including mobile units, manipulators, entertainment robots and humanoids. Behavior-based systems were initially developed on mobile robots, where complex tasks were achieved by combining several elementary control modules, or *behaviors* (Brooks, 1986). In most of these systems, however, arbitration between behaviors was crisp, meaning that only one Behavior was executed at a time, resulting in jerky motion. In other systems, several behaviors are executed concurrently and their outputs are fused together, resulting in smoother motion during switching of behaviors (Cameron et al., 1993), (Saffiotti, 1997), (Saffiotti et al., 1993).

The use of behavior-based system for more complex plants than a wheeled unit needs a framework which is able to handle several DOF (Kim et al., 2001) uses fuzzy rules to control a 6 DOF arm, and (Lever et al., 1994) describes an automated mining excavator that uses concurrent behaviors. However, the complexity of these systems makes the design process very demanding: (Kim et al., 2001) uses 120 rules to perform a Pick Up task, and (Lever et al., 1994) uses a neural network for behavior arbitration, thus giving up the readability of the arbitration rules.

We follow an approach to building behavior-based systems that can be applied to control plants with many DOF. Complexity is addressed by a hierarchical decomposition (Saffiotti & Wasik, 2002) of the overall task into simple behaviors. These behaviors are encoded by small sets of fuzzy rules. In addition, fuzzy meta-rules are used to encode the way behaviors are combined together: this makes it very easy to re-configure the system for different tasks.

We define a set of **basic behaviors**, that is, behaviors that perform elementary types of actions, most of them common to all players independently of their role in the field. In our domain, these include turning toward the ball, going to the ball, or moving to a given position. These behaviors constitute the basic building blocks from which more complex types of actions are obtained by hierarchical composition. The behaviors, which are packaged in the HBM module, are defined by way of fuzzy rules (Saffiotti et al., 1993), (Saffiotti et al., 1995). The input space used by all behaviors is the local state provided by the PAM, which contains the current estimates of the position of all the objects in the field. The output space consists of the velocity set-points which are transmitted to the CMD module.

An additional control variable is used to indicate which kicks are applicable in the given situation.
Thus behaviors are coded by using fuzzy rules of the form:

```
if <predicate>                then <action>
```

where predicate is a formula in fuzzy logic that evaluates a series of properties of the local state. This can be done without the need of an analytical model of the system or an interaction matrix, which may be difficult to obtain for complex plants and tasks, as the RoboCup case is. It is also worth noting that the uncertainty in fuzzy system is taken in account as well.

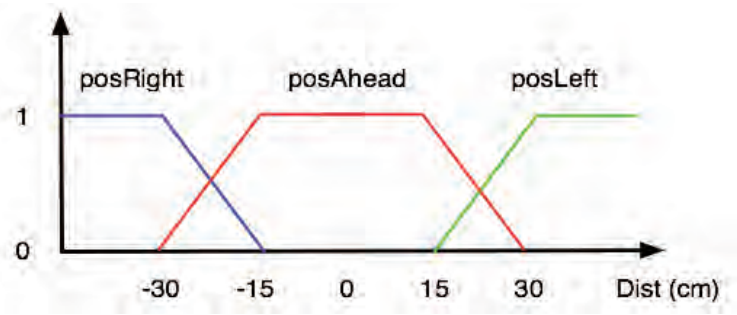


Fig. 5. Membership functions for the *posLeft*, *posAhead* and *posRight* conditions

In order to give a more concrete impression of how behaviors are implemented, we show here the *gkcutb* behavior. For sake of clarity, all the rules shown in this section are given in pseudocode and in a slightly simplified form with respect to the actual rules implemented in the robot. The rules are actually implemented using different LUA methods. The goalkeeper's *gkcutb* behavior uses three rules to control the lateral motion of the robot (action *strafe*) to locate it on the trajectory from the ball to the centre of the net. It also uses three rules to control the orientation of the robot to point the head toward the ball (action *turn*). This behavior does not use the forward motion, and thus it is always set to zero (action *go*). Finally, it controls the type of kick to be applied (action *kick*). If the ball is close enough and the robot is pointing to a safe area it tries to kick it using its both arms and the chest.

```
if posLeft                          then strafe RIGHT
if posAhead                         then strafe NONE
if posRight                         then strafe LEFT
if headedLeft                       then turn RIGHT
if headedAhead                      then turn NONE
if headedRight                      then turn LEFT
if and(freeToKick,  ballClose)      then kick FRONTKICK
always                              go STAY
```

The aim of this behavior is to keep the goalkeeper on the trajectory of the ball to the net. The rule conditions are evaluated from features of the local information, like Ball<ρ, θ> (distance and orientation to the ball), and produce a truth value between 0 and 1. Fig. 5 shows the truth value, or *membership function*, of the first three conditions as a function of the *x* position of the robot with respect to the ball-to-net trajectory. These functions are defined by the designer, and depend on how the robot should try to cut the ball. The consequent of the rules indicate which control variable should be affected, and how: the first three rules involve lateral motion of the robot, while the three following rules involve rotational motion. Each rule affects the corresponding control variable by an amount that depends on the truth value of its condition: smaller or larger adjustments to the robot motion will be generated depending on how much the robot is close to the ball-to-net trajectory. Rules are evaluated and combined according to the standard Mamdani approach.

Behaviors also may incorporate perceptual rules used to communicate the perceptual needs of active behaviors to the perceptual anchoring module (PAM), by using fuzzy rules of the form:

```
if <predicate>                then need <object>
```

whose effect is to assert the need for an *object* at a degree that depends on the truth value of *condition*.

In order to give a more concrete impression of how perceptual behaviors are implemented, we show here the *trackbnet1* behavior. This behavior is useful when the robot tries to store a goal in the opposite's net. In this situation it will always concentrate attention on the ball, but when it is close to the ball, it might want to also get information on the relative position of the net, in order to fine motion to head to the net. This can be accomplished with only the following two rules:

```
always                        need BALL
if ballClose                  then need NET1
```

where *always* is a condition which is always true. In a situation where the ball is at 400mm from the robot, the truth-value of ballClose is 0.7, and these rules assert a value of needed of 1.0 for the anchor BALL and of 0.7 for NET1. Behaviors are dynamically activated and deactivated according to the current task and situation, and several behaviors can be active at the same time. The needed values stored in the *S* state are those asserted by the active behaviors, combined by the max operator. This guarantees that perceptual anchoring only depends on the currently active behaviors, hence on the current task.

We build **complex behaviors** by combining simpler ones using fuzzy meta-rules, which activate concurrent sub-behaviors. This procedure can be iterated to build a full hierarchy of increasingly complex behaviors. The mechanism used to perform behavior composition is called Context-Dependent Blending (Saffiotti, 1997). Thus, under the CDB paradigm flexible arbitration policies can be obtained using fuzzy meta-rules of the form:

```
if <predicate>                then use <behavior>
```

For each such rule, the controller evaluates the truth value, in range [0,1], of <condition> and activates <behavior> at a level that corresponds to this value. Several behaviors may be active at the same time: in this case, their outputs are fused by a weighted combination according to the respective activation levels. Fusion is performed on each control variable independently. The use of fuzzy logic to fuse the outputs of concurrent behaviors provides a smooth transition during switching between behaviors (Saffiotti, 1997), which would be difficult to achieve in architectures based on crisp arbitration like subsumption architecture (Brooks, 1986).

As an example, the following meta-rules implement the *gkclearb* behavior, which uses four rules to decide what action to take: turning until the robot faces the ball, moving the robot to approach the ball location, kicking the ball or moving the robot between the ball and the opponent's net. The behavior also controls the type of kick to be applied; depending on the orientation of the robot, it uses either arms or the head.

```
if not(ballSeen)                              then use faceball
if and(ballSeen, ballClose, freeToKick)       then use dokick
if and(ballSeen, ballClose, not(freeToKick)) then use alignbnet1
```

One key characteristic of our behaviors combination technique is that there are well-established techniques to perform fuzzy fusion of the output of behaviors (Saffiotti, 1997). The rule-based approach and hierarchical organization allows us to design, implement and test very complex behaviors, like the goalkeeper behavior, with a limited amount of effort. The goalkeeper behavior involves the use of navigation, manipulation, and perceptual actions in a highly dynamic and unpredictable environment. The full goalkeeper has been decomposed into 12 behaviors, which involve more than 70 fuzzy rules (including those in the basic behaviors) plus 12 perceptual rules. The development of these rules required about 4 weeks of work by one person (Martínez & Saffiotti, 2003).

We have implemented a Java based editor for HBM behaviors and a monitor. The HBM editor (Fig. 6b) is a simple text editor with an integrated LUA interpreter, which is used for detecting syntax errors. If the edited behavior contains no syntax errors, it can be sent to the robot at any time, without the need of stopping or rebooting it. The HBM monitor (Fig. 6b) is a visual tool that shows information very helpful for debugging behaviors. It includes: the table of objects (ball, nets, landmarks) with their anchoring value and relative positions, the current estimated robot position, the current output of the HBM (velocities), the current active behavior, a graphical display of the most recently viewed objects in robot centric coordinates, and a graphical display of the current global robot position with its uncertainty. When debugging single behaviors, we can easily contrast what the robot is actually doing and what it should be doing, and knowing what it actually perceives. A very difficult to debug issue is checking the different preconditions of an action with the real robot because the uncertainty in the perception and localization systems. With this approach we can produce a preliminary version of a behavior with several parameters estimated, then put the robot on the field and execute a single behavior. We then monitor the execution of the behavior, modify the LUA code accordingly and send it to the robot. We repeat this process until we are satisfied with the result. We then can execute the full system allowing for the change of the active low-level behavior by the high-level behaviors.

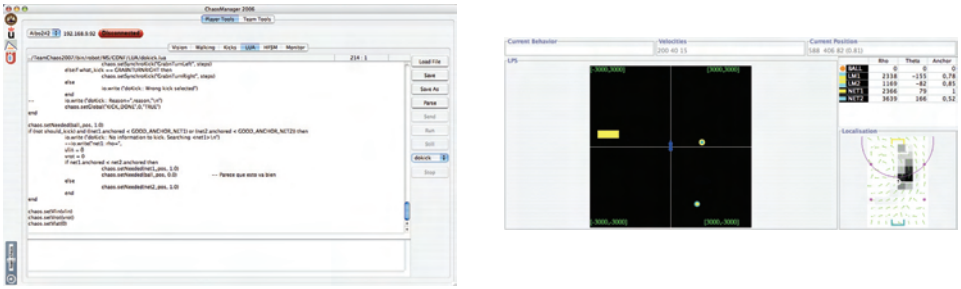


Fig. 6. The HBM tools, code editor (left) and monitor (right)

4.3 Goalkeeper Example

We can use complex behaviors to form even more complex ones, thus creating a *behavior hierarchy*. A simplification of the goalkeeper's strategy is exemplified in Fig. 7. The squared boxes indicate basic behaviors, that is, behaviors that directly control motion and do not call any sub-behaviors.

There is a set of behaviors that are common to all players and are also used by the goalkeeper.

- *lookball*: Turns the robot until it directly sees the ball.
- *go2ball*: Moves the robot to the ball location.
- *dokick*: Moves the robot towards the ball and applies a kick.
- *alignbnet1*: Moves the robot until it is aligned with both the ball and the opponent's net.

While most basic behaviors are coded by means of fuzzy rules (as described in the previous section), there are some cases in which they are not needed. This is the case of the *lookball* behavior. It is intended for finding the ball by means of turning on place. The behavior code is as follows:

```
slowTime       = 400

local ball     = chaos.lps_getLpo(BALL)
local info     = chaos.getBehaviorInfo ()

if info.isNew > 0 then
        sgn = ball.theta / math.abs (ball.theta)
        chaos.setGlobal("BALL_DIRECTION",INTEGER,sgn)
end
sgn = chaos.getGlobal("BALL_DIRECTION")

vrot = 0
if info.timer < slowTime then
        vrot = 50 * sgn
else
        vrot = 75 * sgn
```

```
end

chaos.setNeeded(BALL,1.0)
chaos.setVlin(0)
chaos.setVrot(vrot)
chaos.setVlat(0)
```

For performance reasons, if the robot is not seeing the ball, it is best to turn towards the place where the robot last saw it. This is accomplished using the LUA based state methods, using the global variable BALL_DIRECTION. In addition, during the first four seconds the robot turns slowly (the ball might be close to the robot) and then turns faster (to cover the maximum area per time).

There are some basic behaviors which are specific for the goalkeeper:

- *gktracklms*: Select the least recently seen landmark as a desired perceptual goal. It directly calls the LUA special method *trackLandMarks()*
- *gkkeepout*: Turns the robot slowly moving until it is outside its net.
- *gkkeeparea*. Put the robot facing forward and then moves it to the goalkeeper area.
- *gkkeepbarea*. Put the robot facing backwards and then moves it below the penalty area.
- *gkcutb*. Turn and move sideways in order to intercept the ball trajectory.

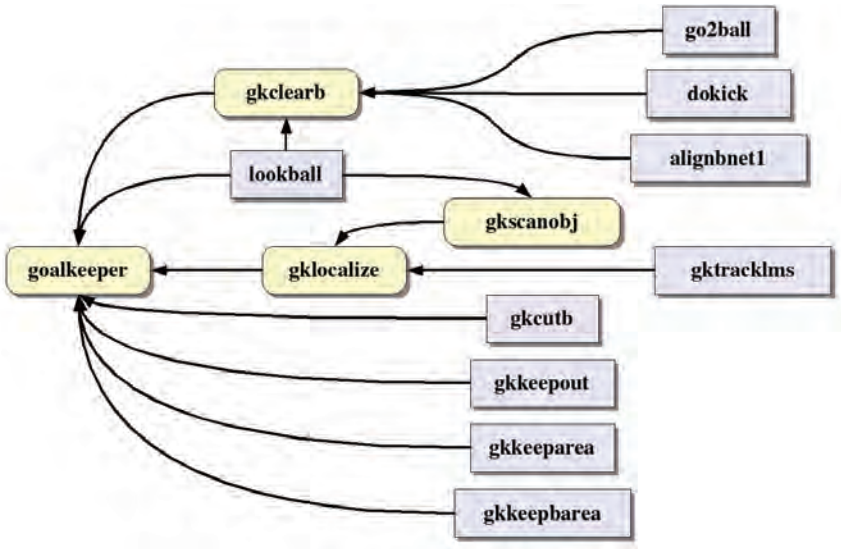


Fig. 7. Behavior hierarchy for the goalkeeper

The other behaviors in the hierarchy are complex behaviors intended to perform the following goalkeeper tasks:

- *goalkeeper*: Top-level goalkeeper behavior.
- *gkscanobj*: Scan the field on place until a given object is found. Look at landmarks and nets cyclically.
- *gklocalize*. Get a better position estimation through looking for the closest landmarks.
- *gkclearb*. Turn and move forward in order to kick the ball out of the goalkeeper's area.

Some of the above behaviors express specific perceptual needs by way of perceptual rules. For instance, most behaviors express a need for the ball position. The *gkkeepout* and *gkkeeparea* behaviors both need to have accurate information about the robot's own location in the field, and hence they express a need for the most probably visible landmarks, including the opponent's net. Moreover, *gkcutball* and *gkclearb* behaviors both need to have accurate information about the ball position in addition to the robot's own accurate location in the field (to avoid self scoring). In general, the overall perceptual needs of the *goalkeeper* behavior depend on which sub-behaviors are activated at every moment, and are used to direct perception.

5. High-level Behaviors

5.1 The HFSM Model and Tools

For specifying and implementing high-level behaviors we make use of the Hierarchical Finite State Machine (HFSM) paradigm (Hugel et al, 2005). In some way or another, the notion of state is usually implied on the execution of a high-level behavior: there is a necessity on knowing what was the state of execution between two successive invocations of the behavior. In the RoboCup, most teams implement some form of state machines, be they ad hoc implementations or the output of formal tools like XABSL (Loetzsch et al., 2006) or Petri Nets (Ziparo and Iocchi, 2006).

An HFSM consists on a set of states, meta-states, which are state machines, and transitions between states and/or meta-states. When the robot is in a state, it executes the corresponding state's code, which is standard LUA code accessing to local perceptions (ball, nets, landmarks, etc.), global information (global ball position, own location, etc.) and shared messages (teammate positions, etc.) from other robots. The states usually invoke low-level behaviors (*faceball*, *go2ball*, etc). The transitions between states and/or meta-states define the conditions to change from one to the other state by the initial and final conditions of the states or meta-states. The meta-states are automata in their self and must carry out all the preconditions for an automaton; they must have an initial state, which is executed first, and cannot contain itself, i.e., an automaton can contain several meta-states (but not itself) and these meta-state can be referenced in different places in our automaton or from other automaton without duplicating code. This simple yet powerful paradigm allows us to specify and reuse machines than can be used inside others, avoiding the repetition of code and allowing for a better code management. For instance, if a typical set of actions is modeled using a meta-state called *Score*, whenever the conditions for scoring are satisfied and no matter in which state we are, we can always call that meta-state. Thus we write code

for scoring once, and we can invoke it in different situations. This is much like a subroutine is in programming.

In our implementation, the main HFSM is a meta-state. Meta-states can be referenced from different states without duplicating code. The transitions are implemented defining two conditions to change between states: test code and priority. When the robot is in a state, it checks the test conditions from all transitions from this state. If some of these tests are satisfied, the new state for the robot is the final state of that transition otherwise the robot continues executing the current state. In the case many transitions come out from the same state, the priority associated to the transition is used to decide the final state. In practice, transition code is checked considering the priority, and when the conditions of a transition are satisfied, the robot's state is changed. When the transitions are checked, not only the transitions from the state (inside the meta-state) are checked but also the transitions from the meta-state. In fact, transitions from the meta-state are first checked, that is, transitions from meta-states have more priority than transitions from states.

The HFSM mechanism is also used for role assignment and execution (Agüero at al., 2006a), so that field players can play different roles in different game conditions. For example, if a defender goes to the ball it can change its role to attacker and another robot should change to its own role to defender. This can be easily achieved defining several meta-states and sharing information between the robots in order to know when to change. Fig. 8 shows an example of the HFSM of field players using three roles: attacker, defender and supporter. Each one of these roles is implemented as a meta-state, and the transitions reflect the conditions for switching from a given role to another.

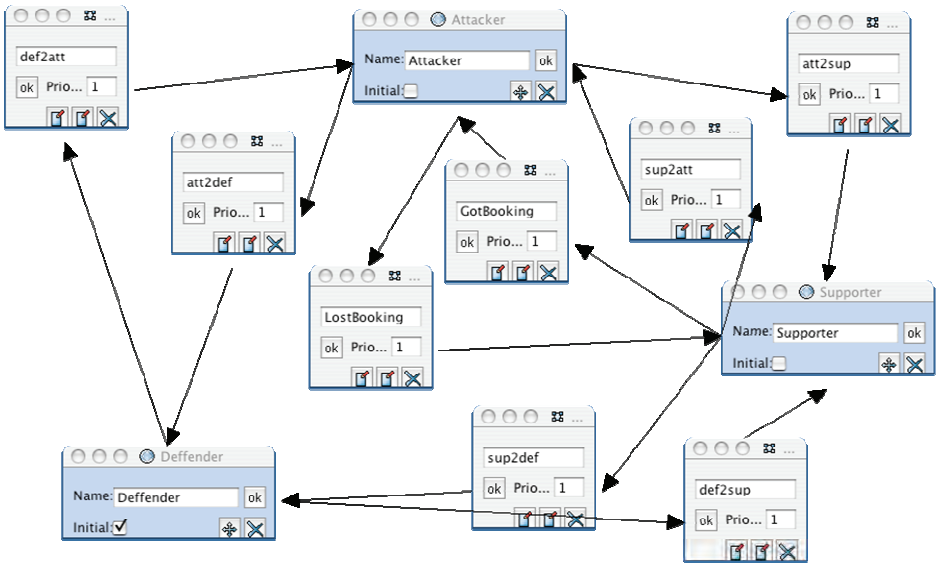


Fig. 8. A sample HFSM with meta-states and transitions

A very important feature of the HFSM model is that a visual tool can be easily produced, so that code development is greatly sped up. We have implemented a Java based visual editor

for HFSM which is based on a early implementation of the tool by the Université de Versailles (Hugel et al, 2005). We have redesigned the GUI and in addition to generating C++ code, we also generate LUA code. The development process with the HFSM tool is as follows. The user starts by creating a set of states and meta-states with their corresponding transitions. Then the corresponding LUA code for the states and transitions is edited. Finally, the user can generate LUA code using the corresponding menu and then transfer it to the robot. The generated LUA code is included in a single LUA file. This process can be repeated over time without the need of stopping or rebooting the robot. The GUI includes a tree view of the whole HFSM (to get an overview of all the states, meta-states and transitions) and a visual view of the selected meta-state (Fig. 9a). When the code of a state or transition is edited, a programming window is open with a syntax-coloring editor and the list of available behaviors that can be invoked (Fig. 9b).

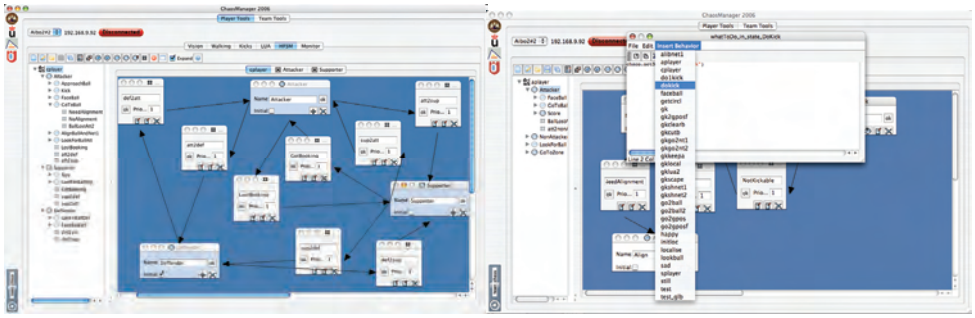


Fig. 9. The HFSM editor, for meta-state edition (left) and code edition (right)

5.2 Attacker Example

In order to show how to apply HFSM to robotics soccer players, we present and describe a simple attacker, quite similar to the one currently used for competitions, being the major difference the lack of role negotiation (which is implemented in a higher level state machine, as shown in Fig. 8). The *Attacker* HFSM (shown in Fig. 10) is composed of three states: *FaceBall*, *GoToBall*, and *Score*. This HFSM should be activated only when the ball is in direct view of the robot (this is something that a higher level state machine should be take care for). The rationale behind the Attacker is to turn towards the ball (by calling the *faceball* behavior from the *FaceBall* state), then move towards the ball (by calling the *go2ball* behavior from the *GoToBall* state), and finally pushing the ball towards the net. This last action is complex enough to be divided into some stages, and thus the state *Score* is in fact a meta-state.

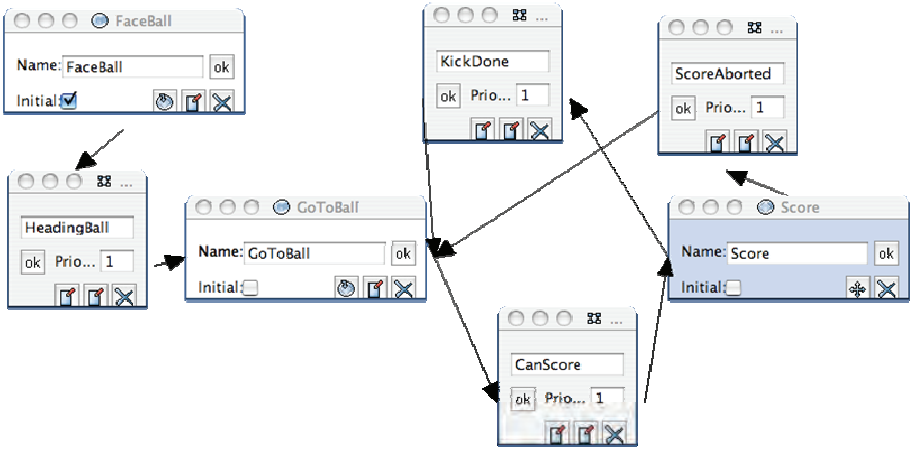


Fig. 10. The *Attacker* HFSM

The *Score* meta-state (shown in Fig. 11) is composed of three states: *ApproachBall*, *Align*, and *DoKick*. This meta-state should be activated only when the robot is close enough to the ball (approximately 50 cm). The rationale behind the Score is to move in a more precise way towards the ball (by calling the *go2ball* behavior from the *ApproachBall* state) while taking care than the robot, the ball and the net are more or less aligned (by calling the *alibnet1* behavior from the *Align* state), and finally produce a kicking movement that pushes the ball into the net (by calling the *dokick* behavior from the *DoKick* state).

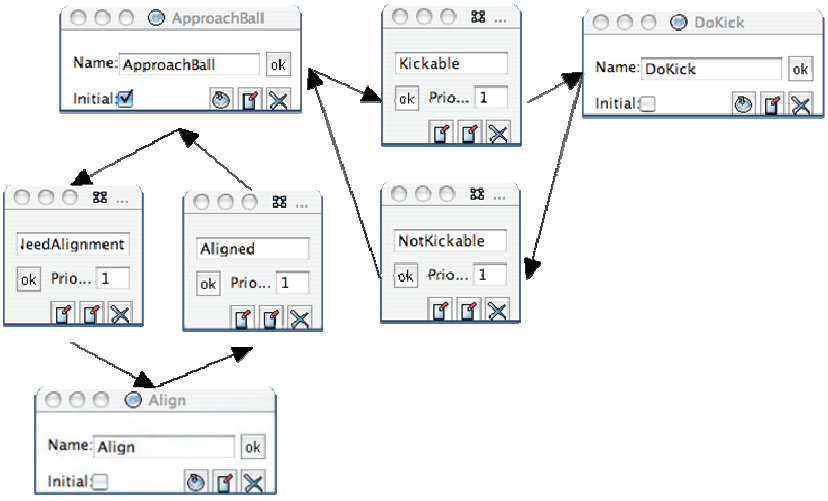


Fig. 11. The *Score* meta-state

In general, the LUA code embedded in the previously described states is very simple, and consists in a call to the corresponding behavior, because each state has been designed to correspond to a single behavior. For instance, the *DoKick* state contains the code:

```
chaos.setBehavior("dokick")
```

There are some states that need a little more complex code. For example, the *GoToBall* state not only calls the corresponding single behavior *go2ball* for robot movement, but also takes care of calling the appropriate localization related tasks, to avoid that in long displacements the robot misses the landmarks. This is the main difference between the state *GoToBall* and *ApproachBall*: the later does not perform localization tasks to concentrate the visual focus of the robot on the ball. The *GoToBall* state LUA code is as follows:

```
local gs       = chaos.gsGetMyPos()
local ball     = chaos.lps_getLpo(BALL)
local net1     = chaos.lps_getLpo(NET1)
local net2     = chaos.lps_getLpo(NET2)

if (gs.quality < 0.6) then
       chaos.trackLandMarks()
end
if (net1.anchored < 0.5) and (net2.anchored < 0.5) then
       chaos.setNeeded (NET1, 1.0)
end

chaos.setBehavior("go2ball")
```

The code performs three activities. If the robot is not properly localized (its localization quality goes below 0.6) a special method *trackLandMarks()* is called, which sets necessities for the most relevant landmarks or nets. In addition, if one of the nets has not been perceived for some time, it sets the necessity for the opponent's net, which might be helpful when attacking. Finally, it always invokes the *go2ball* behavior.

Besides having LUA code in the states, there is also LUA code in the transitions, which basically check for the preconditions of activation of the corresponding state. These usually imply simple tests of either the local or global state (LPS or GS). For instance, the *Kickable* transition code from state *ApproachBall* to *DoKick* (Fig. 11) is as follows:

```
local ball = chaos.lps_getLpo(BALL)

if ball.rho < 600 then
       return true
else
       return false
end
```

This code tests if the distance of the robot to the ball is less than 60cm, in which case the ball can be kicked.

6. Conclusions

This work has presented the architecture and behavioral programming model used to develop a team of the Sony Four-Legged League, which is one of the official leagues of the RoboCup. This league is a very demanding scenario, with high uncertainty in perceptions and limited processing power. In addition, having a competition implies that some dates the software development and tuning presents high activity peaks. These facts condition the way robots have to be programmed. Thus, our main goal has been improving productivity as much as possible while being able to correctly develop all the required behaviors.

The approach consists on adopting a programming architecture that reflects a cognitive separation of modules and allows for an efficient management of modules code. Because the standard programming mode of the AIBO robots makes use of OPEN-R and C++, the on robot behavior development control cycle is very unproductive, and this is typical task that sooner or later must be done (typically in dates close to or during the competition). We have then opted to use an embedded language to make much easier the behavior development task and have selected the LUA language for its many features, being the more important benefits its reduced footprint, its clear language and its execution speed. We have shown different examples of behaviors coded with the language and the main methods of the custom library to access the architecture from LUA.

In order to organize behaviors in the architecture, we divide them into two types, low-level and high-level behaviors, which are implemented using the HBM and the HFSM model respectively. The HBM model allows us to define behaviors by way of fuzzy rules and fuzzy meta-rules in order to cater with uncertainty in both perceptions and actions. The HFSM model allows us to define behaviors by way of state machines in order to sequence high-level tasks. The combination of these two models allows for the combination of the conceptual expressiveness of state machines and the robustness of fuzzy controllers, with the added benefit of being programmed in LUA, which allows for a very productive development cycle.

Although all the work presented has been directed towards robotics soccer, it is important to note that the methodology and tools presented can be used in other scenarios in which the on robot behavior development is an important or crucial task. Two future lines open on this work: porting all the software and technology to a humanoid robot, and incorporating the methodology in the development of a prototype unmanned boat.

7. Acknowledgement

This work has been supported by CICYT project DPI2004-07993-C03-02. The development of the different software pieces could not have been possible without the contribution of many individuals. Without being extensive, we would like to acknowledge: Alessandro Saffioti (University of Örebro) for his implementation of the ThinkingCap architecture and the HBM, in which the one described is based on, Vincent Hugel (Université de Versailles) for

his early implementation of the HFSM in which ours is based on. Last but not least, we would like to thank some members of the TeamChaos (Vicente Matellán, Miguel Cazorla, Francisco Bas, Carlos Agüero and Francisco Martín) for implementing, testing or debugging different pieces of software.

8. References

Agüero, C.; Matellán, V.; Cañas, J.M. & Gómez, V. (2006a) Switch! Dynamic Role Exchange among Cooperative Robots. *Proceedings of 2nd International Workshop on Multi-Agent Robotics Systems*, Setúbal, Portugal

Agüero, C.; Martín, F.; Matellán, V. & Martínez-Barberá, H. (2006b) Communications and Simple Coordination of Robots in TeamChaos. *Proceedings of 7th Workshop on Physical Agents (WAF-06)*, Gran Canaria, Spain

Books, R.A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, March 1986, 14-23.

Buschka, P.; Saffiotti, A. & Wasik, Z. (2000). Fuzzy Landmark-Based Localization for a Legged Robot. *Proceedings of International Conference on Intelligent Robotic Systems (IROS)*, pp. 1205-1210, Takamatsu, Japan.

Cameron, J.M.; MacKenzie, D.C.; Ward, K.R.; Arkin, R.C. & Book, W.J. (1993) Reactive Control for Mobile Manipulation, *Proceedings of International Conference on Robotics and Automation*, pp. 228-235, USA.

Cánovas, J.P.; LeBlanc, K. & Saffiotti, A. (2004). Robust Multi-Robot Object Localization Using Fuzzy Logic. *Proceedings of RoboCup 2004*, pp. 247-261, Lisbon, Portugal.

Herrero-Pérez, D.; Martínez-Barberá, H. & Saffiotti, A. (2004). Fuzzy Self-Localization using Natural Features in the Four-Legged League. *Proceedings of RoboCup 2004*, pp. 110-121, Lisbon, Portugal.

Hugel, V.; Amouroux, G.; Costis, T.; Bonnin, P. & Blazevic, P. (2005). Specifications and Design of Graphical Interface for Hirarchical Finite State Machines. *Proceedings of RoboCup 2005*, pp. 648-655, Osaka, Japan.

Kim, C.S.; Seo, W.H.; Han, S.H. & Khatib, O. (2001). Fuzzy logic control of a robot manipulator based on visual servoing. *Proceedings of International Symposium on Industrial Electronics*, pp. 1597-1602.

Ierusalimschy, R.; de Figueiredo, L. H. & Celes, W. (1996). LUA-an extensible extension language. *Software: Practice & Experience,* Vol. 26, No. 6, pp. 635-652.

Lever, P.J.A.; Wang, F-Y. & Chen, D. (1994). A fuzzy control system for an automated mining excavator. *Proceedings of International Conference on Robotics and Automation*, pp. 3284-3289.

Loetzsch, M.; Risler, M. & Jüngel, M. (2006). XABSL - A Pragmatic Approach to Behavior Engineering. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5124-5129.

Martínez-Barberá, H. & Saffiotti, A. (2003). On the strategies of a 4-legged goal-keeper for RoboCup. *Proceedings of the 6th International Conference on Climbing and Walking Robots (WLAWAR-03)*, pp. 745-752, Catania, Italy.

Saffiotti, A.; Ruspini, E.H. & Konolige, K. (1993) Blending reactivity and goal-directedness in a fuzzy controller. *Proceedings of the 2nd IEEE International Conference on Fuzzy Systems*, pp. 134-139.

Saffiotti, A.; Konolige, K. & Ruspini, E.H. (1995). A Multivalued-Logic Approach to Integrating Planning and Control. *Artificial Intelligence*, Vol. 76, No. 1-2, pp. 481-526.

Saffiotti, A. (1997). Fuzzy logic in autonomous robots: behavior coordination. *Proc. of the 6th IEEE International Conference on Fuzzy Systems*, pp. 573-578.

Saffiotti, A. & K. LeBlanc (2000). Active perceptual anchoring of robot behavior in a dynamic environment. *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3796-3802, San Francisco, USA.

Saffiotti, A. & Wasik, Z. (2002). Using hierarchical fuzzy behaviors in the RoboCup domain. In Zhou, Maravall, and Ruan, editors, Autonomous Robotic Systems, pp. 235-262.

Ziparo, V.A. & Iocchi, L. (2006) Petri net plans. *Proceedings of Fourth International Workshop on Modelling of Objects, Components, and Agents (MOCA)*, pp. 267–290, Turku, Finland.

12

A Comprehensive Framework for Perception in Robotic Soccer

Luciano Oliveira[1], Augusto Loureiro[2] and Leizer Schnitiman[2]
[1]*Institute of System and Robotics*
University of Coimbra,
Portugal
[2]*Programme of Master in Mechatronic*
Federal University of Bahia,
Brazil

1. Introduction

The main difficulties of perception systems, mainly to be applied to robots which play soccer, may be listed as: recognition of objects in a very short time and provision of accurate information to the control system. To accomplish these tasks, many researches in the sensor fusion field have been carried through with the objective of determining complementary or redundant information of the world.

Throughout the years, several works have been proposed to tackle the problem of integrating sensor data to enhance the recognition performance of robotic systems (Bai et al, 2003; Fanny et al, 2004; Lanthier et al, 2004). Bai et al (2003) propose a fusion strategy based on Gaussian distribution over the space of robot position, with the main goal of estimating the robot position, using only range sensors. As the proposed method is based on an asynchronous sensor fusion, and it depends essentially on robot's dead reckoning, the strategy fails whereas the robot runs long distances. Ferrein et al (2005) combines sensor fusion techniques to estimate ball position in a robot soccer field, namely, a weight grid and a Kalman filter strategies. A comparative study was made and the proposed method overperformanced traditional ones, although the work is very Robocup domain-specific. Lanthier et al (2004) use data from inexpensive sensors (sonars and infrared (IR) sensors) to enhance the accuracy of a stereo-based system, whose strategy of data fusion relies on an occupancy grid technique and a Kalman filter.

All those works propose different ways to combine sensor data in order to improve sensing performance. Considering all these elements, a generic framework which copes with the problem of perceiving objects in front of a mobile robot is then proposed. The framework has been applied in Robocup domains, but it can be easily transferred into other robotic fields, under just few adjustments.

The aim of this chapter is to present the proposed framework, which tries to establish an as much as possible tradeoff between accuracy and on-the-fly information. The system consists of a vision servoing module, presented on the top of the robot, in charge of perceiving the

relevant objects (ball and robots), and a set of IR distance sensors whose data are fused with the vision information in order to achieve the spatial location of objects in front of the robot. In the vision system, a cascade of boost rejection (Viola & Jones, 2001) with a Support Vector Machine (SVM) (Vapnik, 1995), at the final stage, are used to guarantee a more accurate classification, in a short time. To integrate all sensor data, a Takagi-Sugeno (TS) fuzzy logic based system tries to balance the better situation in which each sensor data may be used or integrated, and gives the final spatial location of the objects in the soccer field. The proposed framework brings threefold contributions: i) a low computation cost perception system; ii) to the best we know, a novel calibration system, with the use of a regression SVM to obtain a mapping between the world and image coordinate systems, without the need of a rigid transformation scheme, and iii) a fusion system which provides a robust sensor integration. A thorough analysis of each module has been carried through and results have been shown to highlight the proposed framework characteristics.

The rest of the chapter is structured as follows: in Section 2, the system architecture of the soccer robot used and the overall structure of the perception framework are given; in Section 3, a brief overview of the image classification methods is presented; in Section 4, the fuzzy engine used into the framework is discussed; in Section 5, the novel calibration method and its performance analysis are detailed; Section 6 shows some experimental results. Finally, Section 7 draws some conclusions and future works.

2. System Architecture of the Soccer Robot

In Fig. 1, the hardware architecture of the soccer robot, which have been worked with, is depicted. It is essentially a three wheel robot base with a set of IR sensors and a two degrees of freedom (DOF) servoing vision head.

Fig. 1. Hardware architecture of the soccer robot

So that robots play soccer, integration architecture of the perception, control and navigation modules is necessary. Thereby, the architecture proposed in (Costa & Bittencourt, 1999) is used as dorsal spine of aggregation of all robot modules. This architecture is illustrated in Fig. 2.

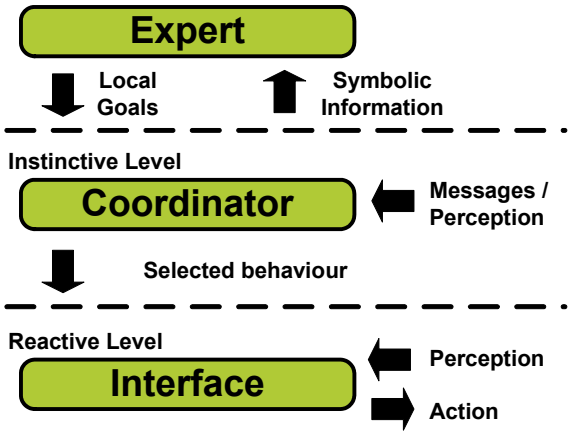


Fig. 2. Software architecture of our soccer robot

The *cognitive* module is a system based on symbolic knowledge which handles information come from *instinctive* level, as well as, asynchronous messages received from other robots (autonomous agents). This module gives global and local goals to the robot, as output. The *instinctive* level is in charge of identifying environment states and choosing the more appropriate behaviour for the robot's current state and goals. The *reactive* level communicates directly to the perception system, receiving a frame containing information about spatial location and velocity of the targets (ball and robots) in front of the robot. Fig. 3 illustrates how this process works. Detailed information of the frame contents (Fig. 3) given by perception system to the control system is discussed in Section 4.

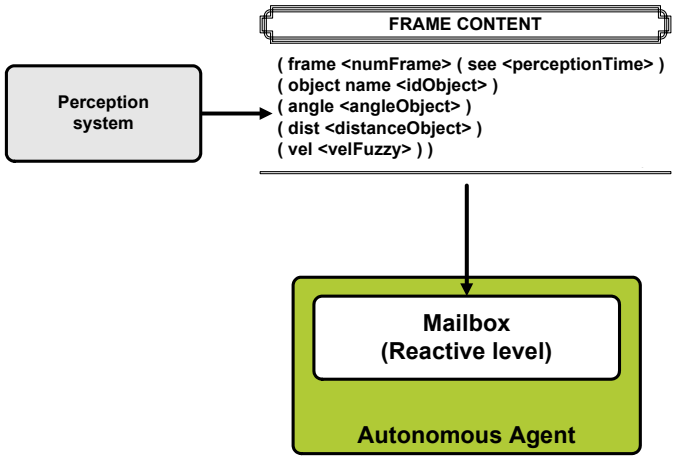


Fig. 3. Communication protocol between the perception system and the reactive level

2.1 The Perception Framework: Overall Architecture

The perception system consists of two threaded modules. After acquiring sensor data, each frame and a respective set of distance data are processed in the specific modules. The objective of this step is to prepare these simple data for the data fusion stage. Fig. 4 shows this process.

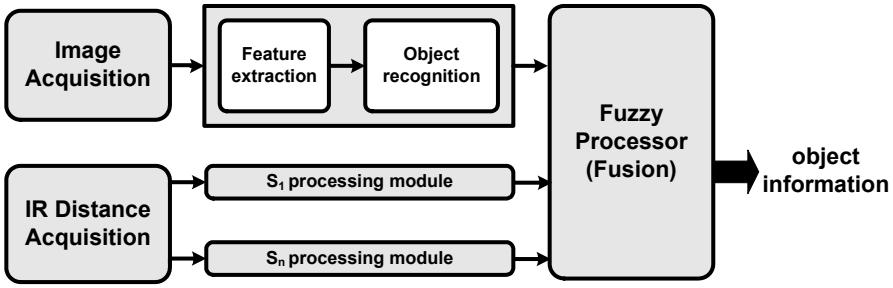


Fig. 4. Communication protocol between the perception system and the reactive level

Particularly, in the vision-based module, each acquired image goes to a feature extractor in order to achieve Haar-like features (Viola & Jones, 2001). These features are, then, classified using a cascade of weak classifier (Adaboost), with an SVM at the end, validating all the classification process. Then, after converting the raw information come from the IR distance and synchronizing it by a timing line strategy, all this information goes to a fuzzy processor, giving object information in form of frames (Minsk, 1975), illustrated in Fig. 3.
In the next sections, detailed information about the perception system, as well as, for completeness, a brief overview of the classification methods used is given.

3. Overview of the Classification Methods

In this section, an overview of the image classification methods applied in the vision system is highlighted, as well as, the way these methods have been combined in order to provide a more reliable object recognition system. This new approach represents an advance with respect of our first implementation in (Oliveira et al, 2005).

3.1 Adaboost

In (Viola & Jones, 2001), a complete system for face recognition is presented. Motivated by this approach, we decided to use Haar-like features in the same way to recognize the ball and robots in the soccer field by means of an SVM, at the end, in order to turn the vision task more robust. With the use of an SVM classifier is intended to decrease the high number of false alarm the Adaboost classifier, against Haar-like features, is prone to. For completeness, we present a brief description of the method and the way it has been used.
Haar wavelate templates (or Haar-like features) have been firstly used in (Oren et al, 1997), along with an SVM classifier, in order to recognize people. The set of these templates was

further expanded in (Viola & Jones, 2001) to incorporate other kinds of contrast differences. Fig. 5 illustrates the common set of the templates.

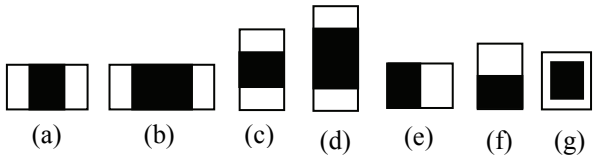


Fig. 5. (a), (b) and (c) are templates for line features; (e) and (f) are for edge features and (g) is the center-surrounded feature

These feature templates are regard to contrast differences in the image pixels. Fig. 6 shows how these templates are applied in the proposed system.

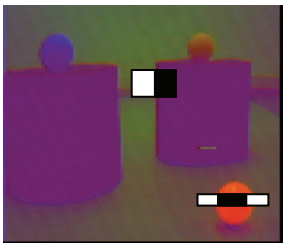


Fig. 6. Templates used in the image pixels

Haar-like features are extracted from the image in an overlapping way. Hence, this approach leads to an overcompleted set of features feasible to be applied in object categorization tasks (Viola & Jones, 2001). For each feature, in the image, a weak classifier $h_j(x)$ is trained and has the following form:

$$h_j(x) = \begin{cases} 1, & if\ p_j f_j < \varphi_j \\ 0, & otherwise \end{cases}, \tag{1}$$

where x is an m×n image subwindow which consists of a feature f_j, a threshold θ_j and a parity p_j which indicates the direction of inequality. Each of $h_j(x)$ reacts to a Haar-like feature. The final Adaboost classifier is composed by all weak classifiers and it is represented as in Fig. 7.

As in Fig. 7, information about all subwindows extracted from the feature extraction module is goes through a cascade of 15 weak classifiers (C1…C15).

As mentioned before, at the end of the rejection cascade, an SVM classifier has been employed in order to reinforce the decision made previously and decrease the number of Adaboost false alarms.

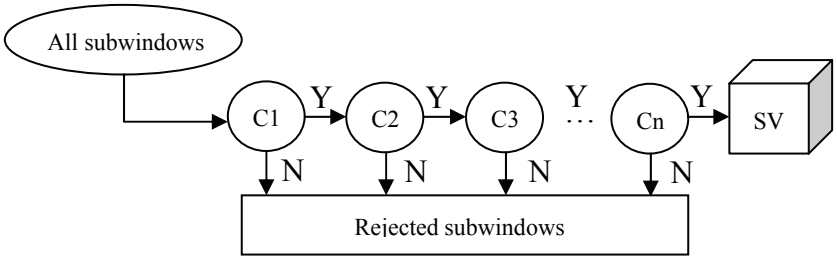


Fig. 7. A cascade of rejection classifiers with an SVM classifier at the end

3.2 Support Vector Machine

SVM is a deterministic learning machine which employs linear discriminant functions into the raw input vector, in case of linear SVMs, or into a high dimensional feature space, in case of non-linear SVMs (Vapnik, 1995). As a supervised method for data classification, its structures embodies a training and prediction stages. In the training stage, algorithms provided by optimization theory are applied in order to learn how to separate the input space; when necessary, a mapping to a higher dimensional feature space is accomplished to guarantee the linear separability of data in any circunstance. In prediction stage, the classifier has the following forms in (2) and (3).

$$\begin{matrix} w_i x_i + b, d_i = +1 \\ w_i x_i + b, d_i = -1 \end{matrix}, \tag{2}$$

Considering a training sample $\Omega = \{(x_i, d_i)\}$, where i= 1...N samples, x_i is the i^{th} input element and d_i is the i^{th} desired output, represented by the set {+1, -1}. Then, the discriminant function for a linear SVM is given by (2), where +1 represents an object and -1, a non-object. In case of non-linear SVM, (2) is written in the form.

$$\begin{matrix} \phi(w_i x_i) + b, d_i = +1 \\ \phi(w_i x_i) + b, d_i = -1 \end{matrix}, \tag{3}$$

where ϕ represents a kernel function which implicitly maps, by means of a inner product, the input space to a higher dimensional space. The most usual kernel function can be listed as:

1. Polynomial: $\phi(x_i x_j) = (\gamma x_i^T x_j^T + r)^T, \gamma > 0$
2. Gaussian: $\phi(x_i x_j) = \exp(-\gamma \left\| x_i - x_j \right\|^2), \gamma > 0$
3. Sigmoid: $\phi(x_i x_j) = \tanh(\gamma x_i^T x_j^T + r), \gamma > 0$

A function is only considered a kernel if it satisfies the Mercel's condition (Cristianini & Shawe-Taylor, 2003; Kecman, 2001), *i. e.*, must be semidefinite and positive.

4. Sensor Fusion Using Fuzzy

In this section, the fusion processor, shown in Fig. 4, is described. This processor is based on a Takagi-Sugeno (TS) fuzzy engine (Takagi & Sugeno, 1985), responsible to decide which data from the sensors are to be taken into account in order to guarantee the most accurate object information to the robot. The TS fuzzy system has been built from the spatial sensor location on the soccer robot, and is depicted in Fig. 8. The main idea of the system is to provide spatial location of objects in front of the robot in polar coordinates (θ, d) and every location is regard to the mass center of the soccer robot. This information comes as from the IR distance sensor as from the camera (according to a calibration scheme, described in Section 5).

Hence, the decision of taking the angle or distance information from the IR distance sensor, from the camera, or combining both of them, must be made by the TS fuzzy system.

According to Fig. 8, the further the object is from the IR distance sensor (S1 to S5), the more accurate is the determination of the object angle. This is verified in the following way: the spread of each IR distance sensor is made by way of a 3 cm wide cylinder (distance between receptor and emitter); for instance, an object between A_1 and A_1' will have a less accurate angle determined by the distance sensors than an object between A_2 and A_2', if only IR distance sensors are considered. In other words, the angle provided by the IR distance sensor will be more accurate in region A_2 and A_2' than in region A_1 and A_1', since the first one is narrower.

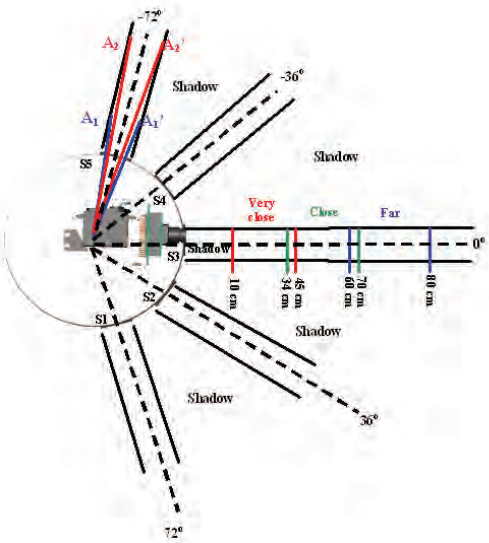


Fig. 8. Determination of the fuzzy structure based on sensor location in the robot

Nevertheless, considering the precision scale, fuzzy sets are defined, according to Fig. 9, with the aim of determining the best angle information to the soccer robot

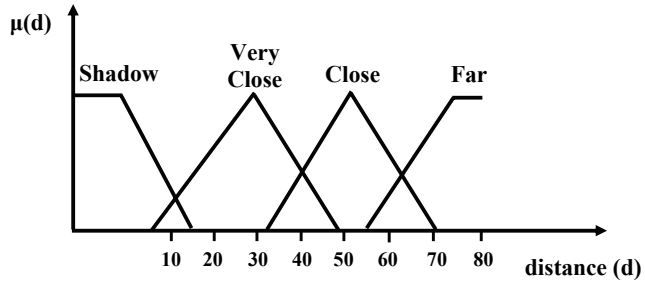


Fig. 9. Fuzzy sets determined from sensor physical disposal in Fig. 8. d is measured in cm

From these fuzzy sets, eight rules were proposed:

R1: IF distance = SHADOW THEN distance = d(camera)
R2: IF distance = SHADOW THEN angle = a(camera)
R3: IF distance = VERYCLOSE THEN distance = d(Si)
R4: IF distance = VERYCLOSE THEN angle = a(camera)
R5: IF distance = CLOSE THEN distance = d(Si)
R6: IF distance = CLOSE THEN angle = a(camera)*0,5 + a(Si)*0,5
R7: IF distance = FAR THEN distance = d(Si)
R8: IF distance = FAR THEN angle = a(Si)

The functions a(.) e d(.) represent, respectively, the angle and distance obtained by the camera and IR distance sensors. The real values of distances and angles, after evaluation of the rules, are determined by:

$$S = \frac{\sum \psi_i z_i}{\sum \psi_i}, \tag{4}$$

where $\}_i$ is the T-norm of each antecedent and z_i is the result of the function *f(x, y)*, responsible for describing the relationship between the fuzzy sets of the antecedent.

At the end of the fusion process, each object is identified and located by means of a internal representation frame structure (Minsky, 1975):

(frame <numFrame> (see <timePerception>)
(object_name <idObject>)
(angle <angleObject>)
(dist <distanceObject>)
(vel <fuzzyVel>))

For each image frame <*numFrame*>, all objects are located and identified by an object name <*idObject*> and three supplied characteristics: angle relative to the center of the base of the robot <*angleObject*>, object distance regards to the front of the robot <*distanceObject*> and fuzzy velocity of each object <*fuzzyVel*>.

The fuzzy velocity <*fuzzyVel*> is determined by (5). Fig. 10 shows the fuzzy sets used.

$$\text{<fuzzyVel>} = [\mu_l(\text{difP}), \mu_m(\text{difP}), \mu_h(\text{difP})], \tag{5}$$

where $\mu_i(difP)$ are membership functions, and i represents each fuzzy set of linguistic variable velocity (*low, medium and high*). difP is the difference between centroid location of an object in relation of frames n and $n - 1$.

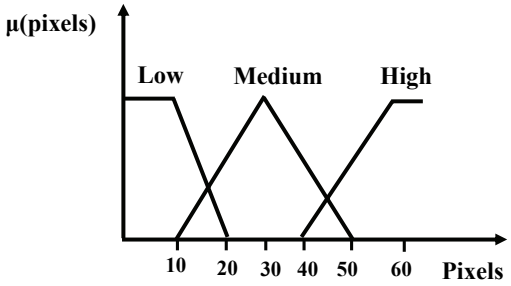


Fig. 10. Fuzzy sets for fuzzy velocity (fuzzified by pixel information). This information is just fuzzified and passed by the control module as an estimative

This velocity information is not intended to be accurate since the control module will translate it into a treatable knowledge within other fuzzy controllers. Hence, the universe of discourse represents the diagonal of the image frame given by the camera sensor. This information is just fuzzified and passed to the control module as an estimative.

5. Calibration Method

Based on the features extracted from the image (height, width, area, centroid), object location is determined in polar coordinates (θ, d), where θ represents the angle and d is the distance relatives to the robot mass center.

To determine these polar coordinates, two regression SVMs have been employed: with respect to θ, a mapping function between the pixel coordinate of the centroid of the object and the concerned physical angle; with respect to d, a mapping between the height of the object in the image and the respective distance between the robot and the concerned object.

To obtain the mapping function centroid-angle, *radii* and parallel lines are drawn in a white paper, according to Fig. 11. Objects with a determined centroid pixel have been placed in each intersection in order to gather information to train the regression SVM.

Firstly, a set of pairs (centroid pixel, angle) is achieved from the centroid of each object in front of the robot and the relative angle to the mass center of the robot.

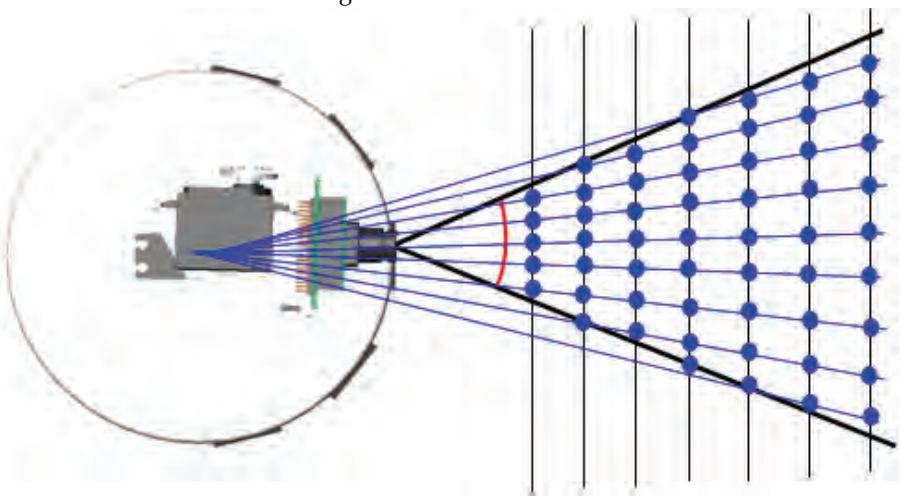


Fig. 11. Calibration map used to extract angles and distances from the image

The vision head is composed by two servo-motors, and, thus, it has two DOF: pan and tilt movements. In order to taper the angular distortion effect in the image, which occurs when the vision head takes different positions related to *z* axis, an angle value is added to the previous pair (centroid pixel, angle), according to Fig. 12.

In resume, in each angle (20, 35, 50 and 65 degrees), a set of pairs (centroid pixels) is determined and a tuple (centroid pixel, angle, angle of head) is given to the regression SVM.

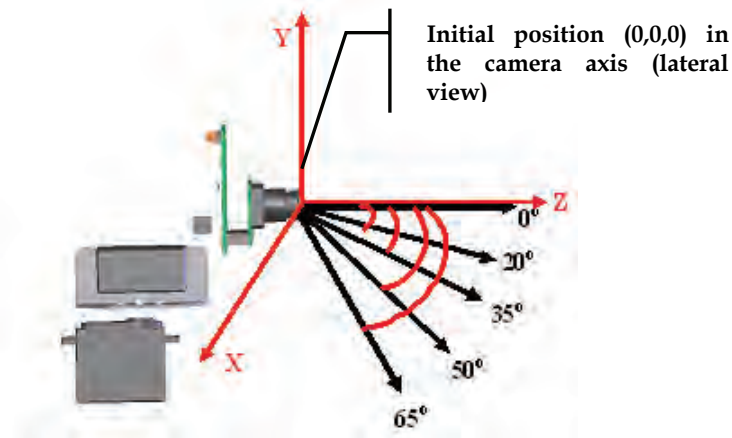


Fig. 12. Angles of the servoing vision head used to enhance the regression SVM performance. The position (0, 0, 0) corresponds to the initial position in which the vision head starts when it is switched on

Information about distance is then extracted to complete the perception information when the objects are found in shadow areas. To obtain more accurate information of the distances, in case of a shadowed object, the robot may rotate its body to correct the estimated distance given by the image. This information is achieved by a simple relation between object height and distance.

5.1 Performance Analysis

Evaluation of the system has consisted to measure the difference between the true and the obtained values. To do this, the calibration map, in Fig. 11, was used once, but, now, with specific angles (-16.5, -8, 0, 8 and 16.5) and distance values (25, 30, 35, 40, 45, 50 and 55) and for each angle shown in Fig. 12. Results are, then, summarized in Table 1.

Table 1 shows the angles obtained by the regression SVM. The angles was gathered through four different angular positions of the vision head, and with objects in different places in the classification map shown in Fig. 11.

After data have been gathered to populate Table 1, analysis of the system error has been carried through and Table 2 summarizes the results. In Table 2, the absolute deviation of the measurements is placed regards to the mean values found in Table 1. The mean values of the absolute deviation were computed and they corresponds to the mean deviation obtained in each angular position of the vision head, as in Table 3.

It is worth noting a systematic positive error in the final correction values found, which indicates a higher error to the right side of the vision system. Hence and according to the final **mean error** of the system (see Table 3), the value of 1.88 degrees has been added to the angles obtained by the regression SVM. The final error could be explained by one or more of the items in the list below:

- **Radial distortion of the camera lens –** higher distortion of the camera lens to the right side;
- **Calibration error –** an error in building the calibration map;
- **Regression SVM –** lack of sufficient generalization or information to the regression SVM.

The values in the Table 3 can be better visualized with the respective values of the standard deviation in each point as shown in Fig. 13.

Vision head = 20 degrees (angles obtained by regression SVM)								
True angle	**In 25 cm**	**In 30 cm**	**In 35 cm**	**In 40 cm**	**In 45 cm**	**In 50 cm**	**In 55 cm**	**Mean**
-16.5º	-16.0º	-15.0º	-15.0º	-15.0º	-15.0º	-15.0º	-15.0º	-15.1º
-8º	-9.0º	-10.0º	-9.0º	-9.0º	-9.0º	-9.0º	-9.0º	-9.1º
0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º
8º	5.0º	4.0º	4.0º	5.0º	5.0º	5.0º	5.0º	4.3º
16.5º	14.0º	14.0º	14.0º	14.0º	14.0º	14.0º	14.0º	14.0º
Vision head = 35 degrees (angles obtained by regression SVM)								
-16.5º	-15.0º	-15.0º	-16.0º	-16.0º	-16.0º	-16.0º	-16.0º	-15.9º
-8º	-10.0º	-10.0º	-10.0º	-10.0º	-10.0º	-10.0º	-10.0º	-10.0º
0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º	-1.0º
8º	5.0º	4.0º	4.0º	6.0º	6.0º	6.0º	6.0º	5.6º
16.5º	14.0º	14.0º	14.0º	14.0º	14.0º	14.0º	14.0º	14.0º
Vision head = 50 degrees (angles obtained by regression SVM)								
-16.5º	-18.0º	-18.0º	-17.0º					-17.7º
-8º	-11.0º	-10.0º	-11.0º					-10.7º
0º	-2.0º	-2.0º	-2.0º					-2º
8º	4.0º	5.0º	7.0º					5.3º
16.5º	13.0º	12.0º	12.0º					12.3º
Vision head = 65 degrees (angles obtained by regression SVM)								
-16.5º	-17.0º	-17.0º						-17º
-8º	-8.0º	-9.0º						-8.5º
0º	-1.0º	-2.0º						-1.4º
8º	4.0º	4.0º						4º
16.5º	13.0º	12.0º						12.5º

Table 1. Results for each vision head angle assumed in the training phase (20, 35, 50 and 65)

Vision head = 20 degrees		
True angle	**Mean**	**Absolute deviation**
-16.5º	-15.1º	-1,4º
-8º	-9.1º	1,1º
0º	-1.0º	1,0º
8º	4.3º	3.7º
16.5º	14º	2.5º
Vision head = 35 degrees		
-16.5º	-15.9º	-0.6º
-8º	-10.0º	2.0º
0º	-1.0º	1.0º
8º	5.6º	2.4º
16.5º	14.0º	2.5º
Vision head = 50 degrees		
True angle	**Mean**	**Absolute deviation**
-16.5º	-17.7º	1.2º
-8º	-10.7º	2.7º
0º	-2.0º	2.0º

8º	5.3 º	2.7º
16.5º	12.3 º	4.2º
Vision head = 65 degrees		
-16.5º	-17.0 º	0.5º
-8º	-8.5 º	0.5º
0º	-1.4 º	1.5º
8º	4.0 º	4.0º
16.5º	12.5 º	4.0º

Table 2. Absolute deviation between the mean of the angle values obtained by the regression SVM and the real angles (calibration map)

Vision head angle	**Erro correction**
20º	1.4º
35º	1.46 º
50º	2.56º
65º	2.1º
Mean	**1,88º**

Table 3. Mean error correction of each vision head angular position

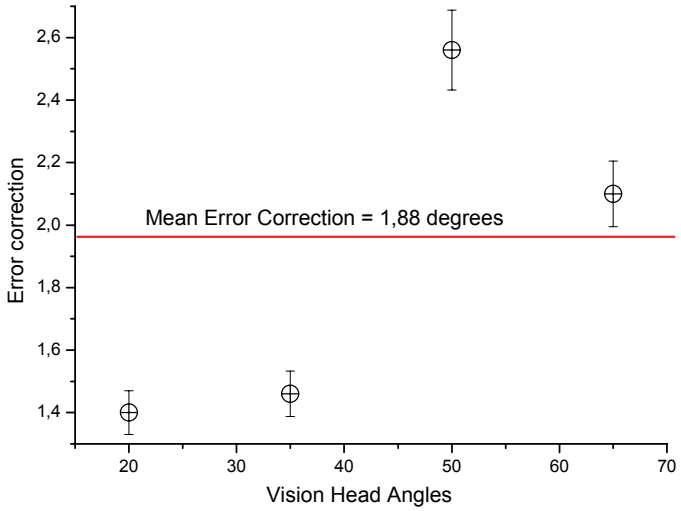


Fig. 13. Graph of the overall error correction

6. Experimental Results

A version of the proposed framework using an SVM multi-classifier to recognize the objects in the soccer field (same team robot, the other team robot and ball) can be found in (Oliveira

et al, 2005). To evaluate this system, Table 4 summarizes results over different illumination values, according to Robocup rules for F180 robot competition (Robocup, 2007).

Illumination (lux)	Classification Rate (%)
570	87.75
660	84.08
780	84.87
800	86.36
920	87.72
970	90.00

Table 4. Classification rate over different illumination values, measured by means of a luximeter

A luximeter was used to evaluate the illumination over the objects in the field. In order to decrease the illumination changing effect, the YCrCb colour space was used. After the object pixels have been classified, a cluster algorithm was applied in order to give the centroid of the object, considering: for the robots, the colours of the balls on the top of the robot, which discriminates the same team and the other team robots; and the ball.

Considering different kernels, Table 5 shows the overall performance classification rate over 800 luxs of illumination.

Kernel type	Classification Rate (%)
Linear	77.6
Gaussian	**84.6**
Sigmoid	78.5
3 deg. polynomial	64.2

Table 5. Use of different kernel types and its respective classification rate in 800 luxs

It is worth noting that Gaussian kernel has given the best classification rate and has motivated its use for classification and the results in Table 4.

Considering all the aforementioned, and motivated by the speed and good results in object recognition (Viola & Jones, 2001) by the boost classifiers, we have decided to apply a new classification approach, described in Section 3.1.

The performance of the system was again evaluated in the same conditions. Table 6 shows the last results in different illumination values (in *luxes*). As can be seen in Table 6, the classification performance rate increased with a respective increasing in speed (from 5 to 10 fps), since the Gaussian SVM is only applied in the final stage of the cascade rejection of the Adaboost (15 stages used. For more information, see Section 3.1).

Illumination (lux)	Classification Rate (%)
570	88.01
660	90.34
780	91.23
800	91.67
920	93.45
970	94.06

Table 4. Classification rate over different illumination values, measured by means of a luximeter

7. Conclusions

A framework for perception in robotics soccer has been presented. The proposed framework has shown been effective in recognize coloured objects in the soccer field but it might be slightly changed to be used in different approaches, through a new set of training samples and fuzzy rules.

A TS fuzzy engine has been used to integrate information from different sensors, particularly, an IR distance sensor and a camera. To integrate these sensor data, a novel calibration method, based on a regression SVM, was developed and it has shown a robust mapping between the calibration map and the obtained values in camera space, with a low average error of 1.88 degrees.

Also, the vision system was evaluated, and the new scheme, with an addition of a cascade of boost rejection and an SVM, has given better performance than in (Oliveira, 2005). The Adaboost classifier decreased the computation cost for the object recognition task and the SVM, used at the last stage of the cascade, reinforce the decisions taken by the Adaboost.

Future work has been conducted to a temporal image fusion by means of a tracking system, which will allow a robot to analyze of the behaviour of the robots and the ball. Moreover, this system will help to enhance the classification performance by decreasing yet more the number of false alarms.

8. References

Bai, T . ; Gu, J. ; Cheng, G. ; Madjalawieh, O. & Liu, P. (2003) A New Landmark Framework for Mobile Robot Localization and Asynchronous Sensor Fusion. *International Symposium on Computational Intelligence in Robotics and Automation*, IEEE, pp. 1451–1456

Costa, A & Bittencourt, G. (1999) From a concurrent architecture to a concurrent autonomous agents architecture. *Third International Workshop in RoboCup* (IJCAI'99), pp. 85–90, Springer, Lecture Notes in Artificial Intelligence.

Cristianini, N.; Shawe-Taylor, J. (2003) An Introduction to Support Vector Machine an Other Kernel-based Learning Algorithms, *Cambridge University Press*.

Ferrein, A. ; Lakemeyer, G. & Hermanns, L. (2005) Comparing Sensor Fusion Techniques for Ball Position Estimation. *RoboCup*.

Kecman, V. Learning and Soft Computing (2001), *The MIT Press*.

Oliveira, L.; Costa, A. & Schnitman, L. (2005) An Architecture of Sensor Fusion for Spatial Location of Objects in Mobile Robotics. *Progress in Artificial Intelligence*, v. 1, pp. 462-473, Springer, Lecture Notes in Computer Science.

Lanthier, M. ; Nussbaum D. ; Sheng A. (2004) Improving Vision-Based Maps By Using Sonar and Infrared Data, *10th Int. Conf. on Robotics and Applications*, IASTED, pp. 118-123.

Minsky, M. (1975) A framework to represent knowledge. *In: The Psychology of Computer Vision*, *McGraw-Hill*, pp. 211–277.

Oren, M.; Papageorgiou, C.; Sinha, P.; Osuna, E. & Poggio, T. (1997) Pedestrian Detection Using Wavelet Templates. *Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, pp. 193–199.

Robocup (2007) [http://small-size.informatik.uni-bremen.de/rules:main].

Takagi, T. & Sugeno, M. (1985) Fuzzy Identification of Systems and Its Applications to Modelling and Control. *Transactions on Systems, Man and Cybernetics*, v. 28, p. 15–33, IEEE.

Vapnik, V. (1995) The Nature of Statistical Learning Theory. *Springer Verlag*.

Viola, P. & Jones, M. (2001) Rapid Object Detection Using a Boosted Cascade of Simple Features. *Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, pp. 511–518.

13

Task Management for Soft Real-Time Applications Based on General Purpose Operating Systems

Paulo Pedreiras and Luís Almeida
LSE-IEETA/DETI, University of Aveiro
Portugal

1. Introduction

Many embedded systems, such as those found in planes, trains, cars, robots and machine tools, exhibit specific timeliness, predictability and precedence constraints that must be respected. In many cases they are built on top of COTS microprocessor boards, possibly PCcompatibles, e.g., PC104, SBCs and Mini-ITX, and using multi-tasking operating systems or kernels, both real-time (RTOS) and general purpose (GPOS) despite the profound architectural and functional differences exhibited by these two classes of software infrastructures (Gopalan, 2001).

In fact, GPOS are typically time-shared multi-processing systems, optimized to manage heterogeneous classes of resources such as CPU, memory, disk, network interface, etc. The performance criteria for these systems are mostly associated with average throughput and fairness, the typical applications are not strictly time-constrained and may exhibit unpredictable blocking and execution times and activation latencies. Conversely, RTOS favor the timely execution of the activities they support. However the predictability delivered by this class of OSs comes at a price, which usually takes the form of additional information and constraints on the application tasks, e.g., bounded worst-case execution time, minimum inter-arrival time or activation period, relative phases and precedence constraints, and on operating system primitives, e.g., bounded blocking times, predictable synchronization primitives, suitable schedulers and admission control.

The architectural dichotomy presented by RTOS and GPOS leads to significant differences at the application implementation level, and thus the choice of using an RTOS or a GPOS may not be completely dictated by the timeliness and predictability aspects, only. For example, in soft real-time applications, which tolerate occasional failures in the time domain, GPOS may be preferred since they deliver sufficient real-time performance and generally outperform RTOS in practical aspects like hardware support, price, availability and diversity and quality of development tools.

Nevertheless, GPOS lack some features that are commonly required by embedded applications, e.g., support for automatic activation of recurrent tasks with enough precision, phase control and precedence constraints. These difficulties have been perceived in the scope of the CAMBADA middle-size robotic soccer team (Cambada), developed at the

This work was partially supported by the European Commission (NoE ARTIST2, IST-2 004527) and by IEETA - Universidade de Aveiro, Portugal

University of Aveiro, Portugal, and led to the development of a user-space simple process manager library, called PMan, which extends the native services provided by the underlying GPOS, Linux in this case. The PMan services are currently used to automatically trigger processes, adapt the Quality of Service (QoS) delivered to each process according to the operational environment, and to enforce phase and precedence constraints between distinct but related processes.
This chapter describes the PMan library implemented in Linux and presents results that confirm the usefulness of the services provided. It is organized in the following way: Section 2 discusses related work; Section 3 addresses the internals of the process manager layer (PMan); Section 4 presents a case study including practical experiments and Section 5 concludes the chapter.

2. Related work

Since the mid 90s that several attempts were made to achieve real-time performance with time-sharing general purpose operating systems. One approach that received substantial attention from the research community was the use of CPU reservations (Lee *et al*, 1996) (Jones *et al*, 1997) according to which a task could establish a contract with the CPU, reserving a given amount of time units every given period. These reservations would have priority over the time-sharing tasks. However, this model was not adequate to provide low jitter execution, could lead to large blocking unless a consistent system-wide reservation scheme was applied to all resources, and did not account for variable execution time. This aspect caused particular inefficiency in multimedia applications, thus (Chu and Nahrstedt, 1997) proposed supporting a new class of periodic but variable execution time tasks, which was completely built at user-level, thus highly portable, but required a kernel that could provide reserves.
This same idea of providing real-time support to applications running in user space was also the motivation for the development of the LXRT module in RTAI/Linux (LXRT). This module permits executing hard real-time tasks in user-space context, with a positive impact in the application development effort and in the system integrity, since programming errors caused by the real-time tasks do not jeopardize the overall Linux kernel sanity. This feature comes at a cost of degraded real-time performance, namely latency and jitter, particularly when Linux system calls are used by the tasks. The Xenomai project (Xenomai) presents several resemblances with RTAI/LXRT, however has a greater focus in facilitating the developers migration from RTOS to GNU/Linux based environments, by providing an emulation layer that supports diverse RT-APIs via skins, e.g., for VxWorks, POSIX and µITRON. Both of these approaches provide good timing performance but require adequate kernel level support that is not provided by the Linux kernel alone.
A different path was followed by Chu and Nahrstedt (1997) in which soft real-time operation was achieved with a simple user-space scheduler based on the fixed priorities defined within POSIX.4. We also follow this approach, providing a user-space scheduler that can simply be executed as a normal application in a Linux GPOS, without need for kernel patches or specialized kernels. With respect to (Chu and Nahrstedt, 1997), we take a step further adding support for off-sets and precedence constraints.

3. The process manager layer - PMan

The core of our proposal is the processor manager layer, called PMan, which aims at facilitating the development of soft real-time applications, extending the native services provided by the underlying GPOS in the following aspects:

- automatic activation of recurrent tasks;
- settling of relative phase control, allowing to establish temporal offsets among tasks;
- precedence constraints, conditioning the release of processes to the conclusion of a set of predecessors;
- on-line process management and QoS adaptation, allowing adding and removing processes at run-time as well as changing dynamically the temporal properties of the executing ones, without service disruption.

The time management within PMan is associated to a periodic tick whose source is userconfigurable and can be generated with a timer or an external event. For example, in the CAMABADA project the PMan tick is associated with the arrival of image frames, in order to minimize the latency between the image acquisition and the activation of the related processing tasks.

The PMan operation relies on certain data concerning the processes, which is kept within the PMan table. A process record in this table is shown in Table 1. The process name and process pid fields allow a proper process identification, which is used to associate a table entry with a particular process and to send OS signals to the processes, respectively. The period and phase fields are used to trigger the processes at adequate instants. The period is expressed in number of **PMan ticks**, allowing each process to be triggered every n ticks. The phase and delay fields permit de-phasing the processes activation, for example to balance the CPU load over time, with potential benefits in terms of process activation jitter. The deadline field supports a basic reflection mechanism permitting the process, when necessary, to carry out sanity checks or recovery actions in case of process misbehavior. A user specified process is automatically activated upon occurrence of a deadline miss event. The following section of the PMan process record is devoted to the recollection of statistical data, which can be useful for profiling purposes. Finally, the status field keeps track of the current process state.

The library of services associated to the PMan layer is summarized in Table 2. The layer is initialized via the **PMAN_init** service and terminated with **PMAN_close**. The process registration in the PMan table is carried out with **PMAN_procadd**. After registering it is necessary to bind the process OS pid using **PMAN_attach**. This separation allows having a process registering itself autonomously or having a third party managing the registration and properties of other processes. Process entries may be removed from the PMan table by calling **PMAN_procdel**. **PMan_detach** dissociates a process from a PMan table entry.

Process identification		
	PROC_name	Process ID string
	PROC_pid	OS process ID
Generic temporal properties		
	PROC_period	Period (ticks)
	PROC_phase	Phase (ticks)
	PROC_delay	Execution delay (ms)
	PROC_deadline	Deadline (μs)
Precedence constraints		
	PROC_pred_name	Predecessor list
QoS management		
	PROC_qosdata	QoS attributes
	PROC_qosupdateflag	QoS change flag
Statistical data		
	PROC_laststart	Activation instant of last instance
	PROC_lastfinish	Finish instant of last instance
	PROC_nact	Number of activations
	PROC_ndm	Number of deadline misses
Process status		
	PROC_status	Process status

Table 1. PMan process record

PMAN_init	allocates resources (shared memory, semaphores, etc.) and initializes the PMan data structures
PMAN_close	releases resources used by Pman
PMAN_procadd	register a process in the PMan table
PMAN_procdel	removes a process from the PMan table
PMAN_attach	attaches the OS process id to an already registered process, completing the registration phase
PMAN_detach	clears the process id field from a PMan record
PMAN_prec_add	specifies a precedence constraint between to processes
PMAN_prec_rem	removes the precedence constraint between two processes
PMAN_QoSupd	changes the QoS attributes of a process already registered in the PMan table
PMAN_TPupd	changes the temporal properties (period, phase, deadline or delay) of a process already registered in the PMan table
PMAN_epilogue	signals that a process has terminated the execution of one instance
PMAN_query	allows to retrieve statistical information about one process
PMAN_tick	called upon occurrence of a tick event, incrementing time within PMan and triggering the activation of processes

Table 2. PMan services

Attaching/detaching processes can be carried out online to allow, for example, selecting one from a set of processes to carry out a given action according a desired cost function, i.e. implementing functional alternatives that can be useful during CPU overloads.
A specific feature of PMan is the support for precedence constraints among processes. For example, in a classical sampling-controller-actuation loop it is necessary to guarantee that these functions are executed strictly in this order to have the end-to-end latency minimized. With the recent advances in computing hardware, e.g., hyper-threading and multi-core CPUs, simpler techniques such as those based on fixed priorities are no longer enough to enforce the right sequencing, being necessary to use explicit synchronization primitives. These ones become hard to use in non-trivial situations with many-to-many dependencies

between processes or when the set of processes and, consequently, their precedence relations, vary dynamically. To cope with this difficulty the PMan library manages automatically the precedence constraints among processes. Applications declare precedence relationships using **PMAN_precadd** and, conversely, **PMAN_precdel** to remove previously established relationships. PMan checks all related precedences before actually releasing a process. The status of precedence constraints is updated whenever processes terminate.
The **PMAN_QoSupd** call allows changing the QoS allocated to each process at runtime. The QoS attributes depend on the underlying OS. The current implementation over Linux considers the OS priority as a QoS parameter. The application processes are assigned realtime priorities, with SCHED_FIFO scheduler, via the **sched_setscheduler** system call. The process priority, supplied as argument to **PMAN_QoSupd**, must be within the range [sched_get_priority_min, sched_get_priority_max]. Similarly, the temporal properties of one process can also be updated dynamically using **PMAN_TPupd**. The ability to change the QoS of processes at runtime is particularly useful when the environment is highly variable and/or hard to characterize, allowing the application to dynamically adapt itself, allocating more resources to the processes that, in each instant, have higher impact on the global performance.
The **PMAN_epilogue** call must be issued by every process managed by PMan, just before termination. This service is required for internal PMan management, namely verification of deadline violations and updating precedence constraints. **PMAN_query**, on the other hand, allows accessing the statistical data of each registered process, which can be useful, for example, for profiling and load management. Finally, **PMAN_tick** carries out temporal management in PMan, incrementing the tick count, activating processes, checking task deadlines, etc. This service must be requested periodically either with a system timer or with an external event. The latter mode, which is not commonly found, supports a transparent synchronization of the application execution with an external event stream, such as the arrival of image frames from a camera with automatic image capture.

4. Process synchronization with PMan: a case study

4.1 The CAMBADA vision subsystem architecture

As stated above, the PMan library was developed to address some difficulties perceived in the scope of the CAMBADA RoboCup middle-size robotic soccer team (Cambada). Currently, the vision subsystem architecture (Fig. 1) uses one catadioptric configuration implemented with a low cost Fire-wire web-camera (BCL 1.2 Unibrain camera) and a hyperbolic mirror. The camera delivers 640x480 YUV images at 30 frames per second. When a new frame becomes available, the image handling process is automatically triggered and the frame is placed in a shared memory buffer. The Color_Class[i] set of processes (Fig. 1) will then analyze the acquired image for color classification, creating a new one with color labels, i.e., an 8 bit per pixel image. This image is also placed in a shared image buffer, which is afterwards analyzed by the Obj_track[i] object detection processes (Fig. 1). The output of the detection processes is placed in the real-time database (RTDB) which can be accessed by the other processes on the system, such as the control action and world state update.

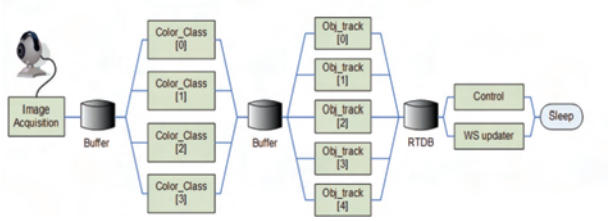


Fig. 1. CAMBADA vision subsystem architecture

4.2 Motivation for explicit precedences and relative offsets

The activation of the image-handling processes is carried out by the PMan manager right after the arrival of each new image frame. In earlier versions (Pedreiras *etal.* 2006; 2007) the PMan triggered sets of related tasks simultaneously, using priorities to enforce precedence constraints. This approach worked well with a single CPU and in the absence of hyperthreading but it failed to enforce such constraints when the computing platform of the robots was updated to Intel™ Core2Duo™ processors with two CPUs and hyper-threading. This is a common problem in real-time applications, which depend on specific features of the underlying hardware platform. When the platform is replaced, previous assumptions may fail leading to a poor application performance and possibly to a system failure. Therefore, the **PMAN_precadd** and **PMAN_precdel** primitives (Table 2) were added to the PMan library to deal with precedence constraints explicitly, as referred in Section 3.

Another identified problem was the need to adjust the control parameters individually, for each robot, due to differences in CPU processing power and thus differences in end-to-end image handling latencies. Moreover, the highly variable nature of the execution time of image processing activities further complicated the controller tuning. To solve these problems the actual release instant of the control process was decoupled from the termination of the preceding object tracking processes and it was set with a predetermined offset with respect to the image reception. This technique substantially reduces the dependence of the control performance on the underlying hardware used and it was added to the current version of PMan.

4.3 Experimental results

To experimentally verify the implementation and assess the performance of the PMan library several experiments were carried out using the CAMBADA architecture depicted in Fig. 1. Table 3 shows the process set used in the experiments.

Process	Period	Precedence list	Delay (ms)	Description
readFireWire	1	{}	0	Interface with camera. Issues the tick event.
Color_Class[0..3]	1	readFireWire	0	Color classification. Each process scans 1/4th of the image.
Obj_Track[0..4]	1	Color_Class[0..3]	0	Object tracking (ball, obstacles, blue goal, yellow goal, lines).
Control	1	Obj_Track[0..4]	20	Issues control commands.
WSUpdater	1	Obj_Track[0..4]	20	Updates the RTDB with the processed data.

Table 3. Experimental process set

Two hardware platforms were used, one based on an Intel Pentium M Processor at 1.6GHz (Asus A3N notebook PC) and another based on an Intel P4 DualCore at 2.6GHz (Asus Pundit desktop PC). Both run the Linux 2.6.22 kernel, with the timer frequency set to 1000 Hz and the High Resolution Timer Support enabled. The first experiment aimed at assessing the overhead induced by the PMan layer, namely by executing the PMAN_tick service. Table 4 shows the latencies measured from the activation of the *readFireWire* process, which calls PMAN_tick, to the start of execution of Color_Class[0], the process that executes right after. Two different scenarios have been considered. The first scenario (Immediate) corresponds to a slight modification of Table 3 in which the Delay parameter is made equal to 0 for all processes. This means that all processes execute as soon as possible, i.e., when the CPU is available and the precedence constraints are met. The other scenario (Deferred) corresponds to the parameters specified Table 3. Both scenarios differ only in the activation of the Control and WSUpdater processes, which is deferred by 20ms after the trigger in the latter case.

	Control and WSUpdater activ.	Max. (µs)	Min. (µs)	Avg. (µs)	St.dev. (µs)
PM, 1.6GHz	Immediate	40	5	12.56	4.61
	Deferred	93	15	34.30	10.75
P4 Dual Core, 2.6GHz	Immediate	35	8	10.77	1.68
	Deferred	42	19	24.18	1.78

Table 4. Upper bound on the execution time of **PMAN_tick**

The measured latencies vary between 5µs and 93µs in the notebook platform and between 8µs and 42µs in the desktop platform. As expected, the average values are lower for the desktop PC due to the higher CPU processing power. It should also be remarked that the deferred execution incurs an additional execution penalty, resulting from the need to create a wake-up thread and an extra call to the nanosleep primitive. Nevertheless, in any of the scenarios analyzed the additional overhead and latency induced by the PMAN_tick event are negligible in face of the PMan tick period considered (33ms).

Table 5 shows the results of a second experiment, aiming at verifying the effectiveness of the deferred activations, namely those of the *Control* process. When the activation is immediate, i.e. with *Delay=0*, the activation latency is highly variable (with the notebook) and hardware dependent, reaching an absolute jitter of almost 18ms in the worst-case, which is over 50% of the sampling period, a non-negligible value from the control performance point of view. With deferred activation, i.e. *Delay=20ms*, the absolute jitter is substantially reduced, being below 3.5ms for the notebook and below 35 micro-seconds for the desktop PC. Note that in the experiments the precedence constraints are always enforced and thus part of the jitter observed in the notebook platform results from the predecessor tasks requiring more than 20ms to complete thus pushing the activation of the control process.

	Control process activation	Max. (µs)	Min. (µs)	Avg. (µs)	St.dev. (µs)
PM, 1.6GHz	Immediate	23143	5810	15448	5592
	Deferred	23465	20037	20224	414
P4 Dual Core 2.6 GHz	Immediate	5963	2168	3296	573
	Deferred	20078	20046	20054	3

Table 5. Control process activation delay

As expected, enforcing a deferral in the activation of processes that are subject to precedence constraints can have a noticeable impact on the respective regularity practically eliminating the respective jitter. Furthermore, the activation latency and associated jitter become nearly hardware independent. This increased execution predictability facilitates the tuning of feedback controllers, such as those used within the Control process, resulting in improved control performance.

Fig. 2. presents a histogram of the control process activation latencies both with deferred and immediate execution in the notebook platform. The reduction in the jitter figures is clear. With the deferred execution near 90% of the process activation latencies occur within a vicinity of 1ms of the desired value while with immediate execution the activation pattern is substantially enlarged, with latencies ranging from near 6ms to around 23ms and two clear peaks. It should be remarked that with immediate execution the start time of the control process depends solely on the completion of its predecessors and that this instant depends on the amount of processing they required. This amount depends on the richness of the images, i.e., on the number of regions that have to be checked, a parameter that depends on the environment (colors of the objects surrounding the playfield, illumination intensity and nature, etc.) and is, to a large extent, unpredictable and highly dynamic. Therefore, to allow a fair comparison among the diverse scenarios a fixed synthetic workload was generated and used throughout the experiments.

Fig. 3 presents a histogram similar to that in Fig. 2 but referring to the desktop platform. The major distinction between the results with both platforms is the strong reduction in the jitter achieved with the desktop PC caused by its substantially higher processing power. Particularly, it is worth noticing the high accuracy with which the control process is triggered in the desktop PC with deferred execution. When this process is triggered the precedence constraints are always already met and thus the residual jitter is due to the handling of OS events, only. This behavior is confirmed quantitatively by inspecting Table 5. The control process activation for the desktop platform varies between 20078micro-seconds and 20046micro-seconds, with a standard deviation of 3 micro-seconds.

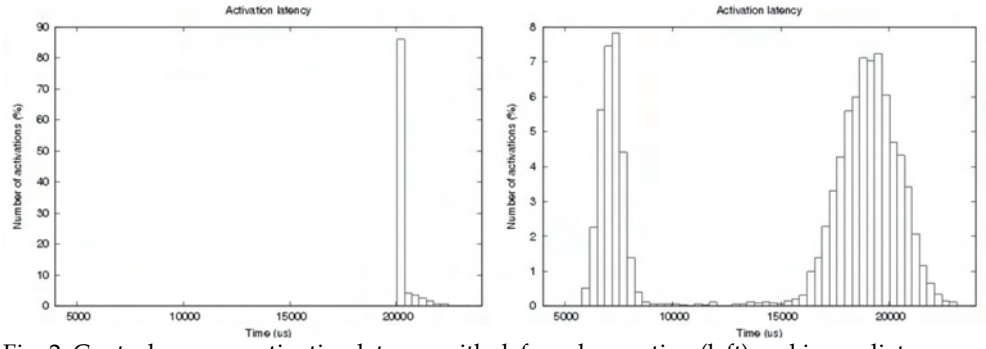


Fig. 2. Control process activation latency with deferred execution (left) and immediate execution (rigth), notebook platform

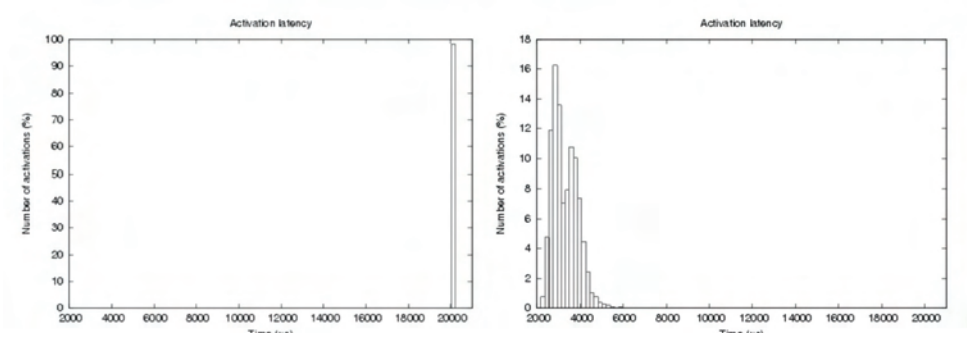


Fig. 3. Control process activation latency with deferred execution (left) and immediate execution (right), desktop platform

Fig. 4 shows an excerpt of an execution timeline in the notebook PC. Process activations are indicated by small circles. The four *Color classification* processes and the five *Object tracking* processes execute in sequence, inheriting the execution jitter of their predecessors. However, the *Control* and *World State update* processes, on top, have a deferred activation that absorbs the execution jitter of the predecessors, resulting in a higher activation regularity. Notice, nevertheless, that these processes also have precedence constraints, which are always enforced, even if the execution of the predecessors takes longer than the specified deferral delay.

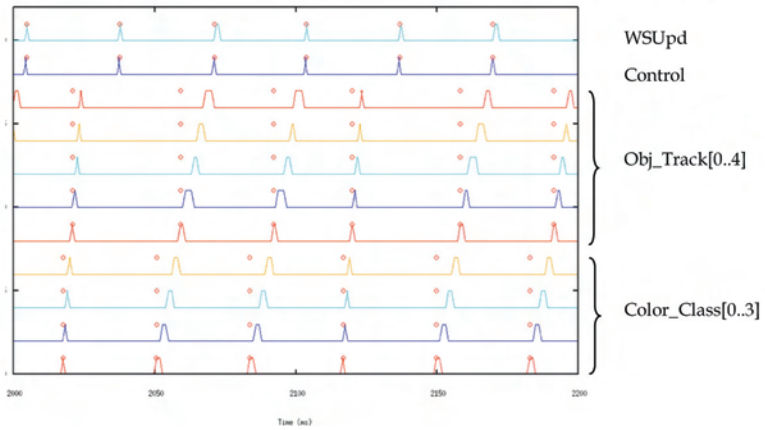


Fig. 4. Process execution timeline. Notebook PC, deferred execution

5. Conclusions

Using general purpose operating systems for soft real time applications has several advantages related with low costs and the abundance of device drivers and software tools. However, such applications still require adequate timing services, for process activation and synchronization.

In this chapter we presented a process management library that provides such services with substantial hardware independence and executes completely within user-space, being thus very flexible to deploy and use. In particular, this library provides support for periodic process activations, possibly with relative offsets and explicit precedence constraints, and also dynamic adaptation of temporal parameters and QoS attributes.
The library was developed within the scope of the CAMBADA RoboCup middle-size soccer robots. Using this application as a case study, the chapter presents several practical experiments that show the low overhead induced by the process management structure and the effectiveness of its support for precedence constraints and relative offsets with hardware independence.

6. References

Chu, H. & Nahrstedt, K. (1997). A soft real time scheduling server in UNIX operating system. *Proceedings of European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*. Also In *Lecture Notes In Computer Science*, pp. 153-162, ISBN:3-540-63519-X , Darmstadt, Germany, September 1997.

Chu, H. & Nahrstedt, K. (1999). CPU service classes for multimedia applications. *Proceedings of IEEE Conference on Multimedia Computing and Systems* (MCS'99), pp. 296-301, ISBN: 0-7695-0253-9,. Florence, Italy, June 1999.

Gopalan, K. (2001). Real-Time Support in General Purpose Operating Systems. *ECSL Technical Report TR92, Experimental Computer Systems Lab, Computer Science Dept,* WSUpd Control Color_Class[0..3] Obj_Track[0..4] *Stony Brook University, Stony Brook, NY - 11794-4400*.

Jones, M. B.; Rosu, D. & Rosu, M. (1997). CPU reservation and time constraints: Efficient, predictable scheduling of independent activities. *Proceedings of 16th ACM Symp. On Operating Systems Principles* (SOSP'97), St. Malo, France, October 1997.

Lee, C.; Rajkumar, R. & Mercer, C. (1996). Experience with CPU reservation and dynamic QoS in real-time Mach. *Multimedia Japan*, March 1996.

Pedreiras, P.; Teixeira, F.; Ferreira, N.; Almeida, L.; Pinho, A. & Santos, F. (2006). Enhancing the reactivity of the vision subsystem in autonomous mobile robots using real-time techniques. RoboCup Symposium: Papers and Team Description Papers, RoboCup-2005: Robot Soccer World Cup IX, Lecture Notes in Artificial Intelligence, Bredenfeld, A.; Jacoff, A.; Noda, I.; Takahashi, Y. (Eds.) pp. 371-383, ISBN: 3-540-35437-9 , Springer Berlin / Heidelberg , Springer, 2006.

Pedreiras, P.; Teixeira, F., Ferreira, N.; Almeida, L.; Pinho, A. & Santos, F. (2007). A Real Time Framework for the Vision Subsystem in Autonomous Mobile Robots. In *Vision Systems Applications,* G. Obinata and A. Dutta (Eds.), pp.83—100, ARS, ISBN 978-3-902613-01-1, Vienna, Austria.

Cambada. Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture. University of Aveiro, Portugal. Online, http://www.ieeta.pt/atri/cambada/

LXRT. Online, http://www.fdn.fr/~brouchou/rtai/rtai-doc-prj/doxyapi/group__lxrt.html Xenomai. http://www.xenomai.org/index.php/Main_Page

14

Integrating Autonomous Behaviour and Team Coordination into an Embedded Architecture

Bernd Kleinjohann, Lisa Kleinjohann, Willi Richert and Claudius Stern
University of Paderborn, C-LAB
Germany

1. Introduction

Robotic soccer is a challenging research and application field for the combination of real time embedded systems design with intelligent autonomous behaviour as well as tema coordination and learning. The joint investigation of these themes is pursued by the development of the Paderkicker robots described in this paper.
The Paderkicker team (Richert et al., 2006) consists of five robots (Fig. 1) that already participated successfully in the German Open competition in 2004, 2005 and 2007, the Dutch Open 2006 and the RoboCup 2006 World Championship. Our platform asks for the whole range of research issues needed for a successful deployment in the real world. This includes embedded real-time architectures (Beier at al., 2003; Esau at al., 2003b; Stichling, 2004), real-time vision (Stichling & Kleinjohann, 2002a, 2002b, 2003) learning and adaptation from limited sensor data, skill learning (Richert & Kleinjohann, 2007) and methods to propagate learned skills and behaviours in the robot team (Richert et al., 2005). However, our goal is not to carry out research for specific solutions in the robotic soccer domain, but to use and test advanced techniques from different research projects. The Paderkicker platform serves as a test bench for the collaborative research center 614 (funded by the Deutsche Forschungsgesellschaft). Furthermore, the knowledge in vision, motion and object tracking was used in the AR PDA (Bundesministerium für Bildung und Forschung) project (Reimann, 2005).
This paper describes various aspects of the Paderkicker robots. Section 2 gives an overview of the robots' construction. First the actor systems for driving, ball handling and vision system are described followed by the sensor systems for odometry, landmark and ball recognition. Section 3 focuses on various aspects of the modular robot design. Functional design, hardware, software and process architecture are covered as well as the message format used for communication between different robots and robot components. Section 4 describes the real-time image processing module developed for the Paderkicker robots. Our algorithms for colour segmentation and recognition of objects like ball, field or lines are presented. Furthermore, we propose a DCT-based (discrete cosine transform) pre-processing step improving the subsequent colour segmentation. In Section 5 the behaviour based realization of the Paderkicker robots is explained starting from team coordination issues down to simple reactive behaviour. Section 6 deals with learning capabilities we want to introduce into our

robots. Section 7 concludes the paper with a short resume and an outlook to future developments.

2. Robot outline

The robots of the Paderkicker team were constructed from scratch, since they also should be used for demonstrations in environments which are less restricted concerning lightning and underground conditions than the RoboCup environment. Furthermore they should provide a test platform for different research projects. Nevertheless their main purpose is playing robotic soccer in the midsize league. The Paderkicker robots are mainly developed in the course of student education projects necessitating a modular design approach enabling students to familiarize with the Paderkickers within about two months, after which they should be able to productively contribute to the Paderkickers' development. As already mentioned the main focus of this development is the design of embedded hardware and software and the application of artificial intelligence algorithms in real-time environments.
Size and form of our robots were mainly determined by the RoboCup regulations. The maximum height of 80 cm was chosen for providing the vision system with good overall viewing capabilities. Robot size was motivated by the upper boundary for the area occupied by the entire robot team, which led to the design objective of keeping the robots as small as possible, in order to allow an additional robot in the team. In our case we arrived at an area of 36 cm x 36 cm for each robot.
The carrying structure consists of standard quadratic aluminium profiles with edge length of 2 cm (Fig. 2). At each side the profiles are equipped with T-shaped channels for connecting profiles or fastening functional units for driving, ball handling and orientation. These functional units are realized as mechatronic functional units (MFU) and controlled by separate hardware controllers (see Section 3). The construction of these MFUs, which are built mainly from standard model craft components, will be described in more detail below because of their crucial role in robot soccer.

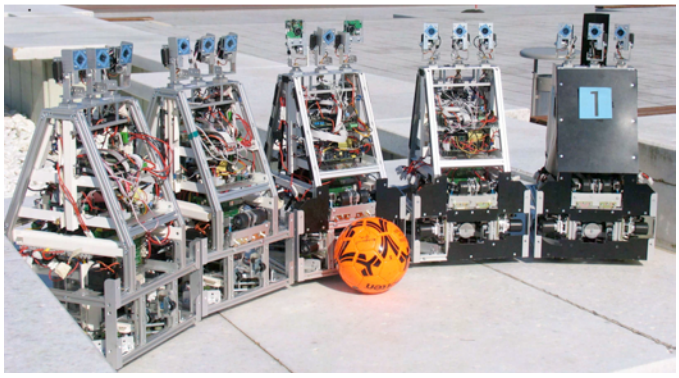


Fig. 1. The Paderkicker robot team

Fig. 2. Robot construction

2.1 Driving system

The Paderkicker robots are equipped with omnidirectional drives (omniwheels) since they allow for simultaneous translational and rotational robot movements while supporting all three degrees of freedom on a plane. Omniwheels are actively driven by a motor into one direction. Orthogonally omniwheels consist of passive rolls with small rolling friction. Since this depends on their footprint, we selected wheels with a small diameter of 6 cm. With hindsight these rolls are capable of driving over very small splits only. Splits like those between lift shaft and cabin are for instance at the upper boundary. Presently, his attribute does not mean a restriction for RoboCup, but it could become one if for instance sport halls should be used for RoboCup tournaments without a carpet that covers installations for fastening sports equipment on the floor. We decided to use four (instead of three) omniwheels per robot to increase stability. When using four driven wheels rotated by 90 degrees each, all four motors contribute to a translational movement. This enables the use of smaller dimensioned motors in contrast to the use of only three omniwheels, because in that case only two of them contribute to a straight movement. Also the assembly angle is less advantageous for three wheels. However special care has to be taken to guarantee that each wheel touches the ground whereas this feature is automatically guaranteed for three wheels. For this purpose we mounted the wheels movably in rubber blocks. Thus all wheels are pressed onto the ground by a robot's own weight. Fig. 3 shows one omniwheel including attached sensors which are described in Section 2.4.

2.2 Vision system

Our vision system contains three independent pan-tilt cameras that are used for active viewing (Fig. 4) in contrast to omnivision systems that are currently used by many other teams. Each camera may independently focus and track a different object of interest like ball, corner, goal or marker. Proper choice of the aperture angle (up to 120 degrees) guarantees a sufficient view in all directions. Our experience showed that for the two outermost cameras a shorter focal distance is preferable to provide an overview of the playing ground while a standard focal distance for the medium camera allows focusing single objects appropriately. A major advantage of our approach, especially for larger soccer fields, is its greater resolution and easier calibration, since it is not necessary to account for the average conditions of the entire soccer field. According to our experience the processing effort for our vision algorithm uses about 15% of the processing capabilities of a Pentium M ULV running at 1GHz.

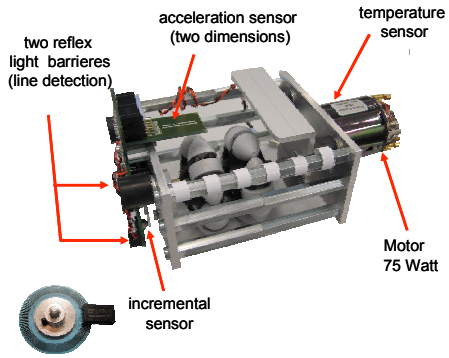


Fig. 3. Omniwheel with attached sensors

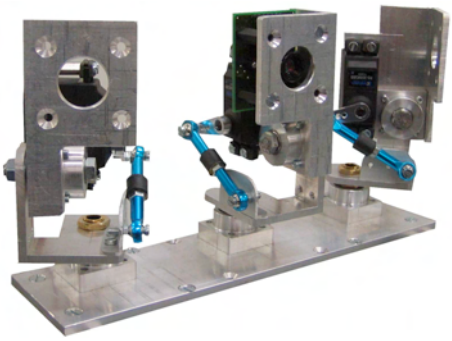


Fig. 4. Active vision system

2.3 Ball handling system

Soccer robots must be able to move the ball in a controlled manner, e.g. for passing it to another robot, taking it away from another one or kicking it towards the goal. The way how they may do it, i.e. how much force may be given to the ball or how long they may touch the ball is restricted by the RoboCup rules to a large extent. For instance, it is not allowed to fix the ball or to enclose more than one third of its diameter. Therefore we developed a mechanism for ball kicking and another one for ball dribbling which is described next.

For ball dribbling we developed a mechanism that allows giving the ball a rotational pulse in diverse directions for rolling it into a desired direction without keeping it in touch with the robot. For this purpose we use three rollers that may act upon the ball in different directions when being in touch with it. Two side rollers are attached horizontally at both sides of the ball handling mechanism and one front roller in the middle as depicted in Fig. 5. The rotational axes are parallel to the ground (side rollers) and parallel to the robot front (front roller). The rollers are fixed at joints supporting an expansion of the side rollers and a lowering of the front roller hereby varying the rotational axis about +/- 45 degrees. Even when driving backwards the rollers can keep the ball rolling backwards and in touch with the robot, although the rules allow this only for a short time. For kicking the ball an electromagnetic solenoid (Fig. 6) is integrated nearly in the bottom center of the robot's chassis. A plunger transfers a directed impulse with adaptable strength to the ball. By adjusting the side rollers correspondingly, the ball may also be hit at its sides and not only in its center, thus allowing a targeting of the ball within an angular range of about 30 degrees. Via this mechanism the robots can kick the ball on the floor. In the future also higher kicking shall be realized.

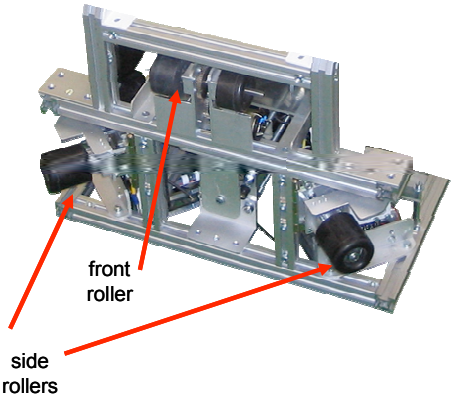


Fig. 5. Rollers for ball dribbling

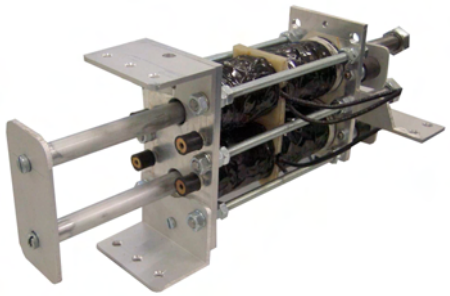


Fig. 6. Solenoid for ball kicking

2.4 Sensory system

Besides the active vision system described above the Paderkicker robots are equipped with several further sensors for monitoring their own status and perceiving their environment. They are located near the actors for ball handling and driving in the bottom layer of the chassis as schematically depicted in Fig. 7.

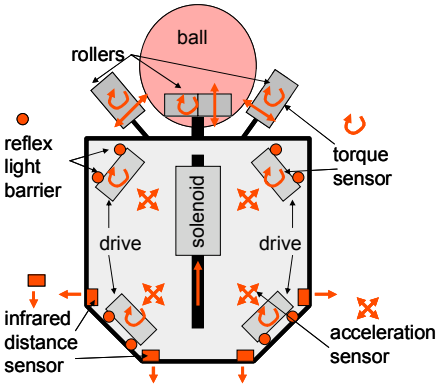


Fig. 7. Sensor and actor location

2.4.1 Motor status

The number of revolutions is monitored for each motor as input for its controller. With help of this value and the current consumption the actual torque of a motor can be estimated. The torque of a motor allows to decide whether for instance a robot is driving against an obstacle (omniwheel motors), whether the ball rollers touch the ball (roller motors) or even during longer driving periods about the floor characteristics and maximum driving velocity on the actual floor. By measuring current and number of revolutions it can also be decided whether one of the omniwheels is spinnning. Furthermore the number of revolutions is used for odometric calculations described next.

2.4.2 Odometry

The sequence of angular movements of the four driving motors is used to calculate a robot's translational and rotational movement on the plane. Together with information about the starting position this allows to estimate the current robot position. It has to be noted that the respective incremental sensors are located near the wheel side and not near the motor (see Fig. 3). They have a resolution of about 1.5 mm. Due to considerable slipping of omniwheels (about 15 to 20%) these calculations are very error prone. Another source for calculation errors are pushes by opponent robots that can be observed frequently although RoboCup rules prohibit them. Since we use four instead of three omniwheels the odometric equation system is over-specified which we use for improving the quality of sensor data.

Further information about a robot's movement is obtained by acceleration sensors at each wheel. They measure the real acceleration over the wheels into two directions entirely independent from the driving motors. Also manual moves are considered. If a wheel spins this leads to faulty large acceleration values. Unfortunately also these measurements are very error prone. Therefore we try to improve odometric data by fusioning the results of several acceleration sensors located at different places within the robot and the data received from the incremental sensors at the wheels. In future the position calculation should be further enhanced by a camera based odometric system that calculates driving direction and velocity from video sequences.

2.4.3 Landmark recognition

Position calculation is most exact if known landmarks can be detected in the environment. For RoboCup lines on the soccer field or goals can be used for this purpose since their position is given. However, if lines are only determined from camera images, the line is usually not detected with sufficient accuracy. This is mainly due to the frame rate of 20 to 30 frames per second and the driving velocity up to 3 m/s that allows only a relatively small resolution regarding time and location. Therefore, the Paderkickers are equipped with two reflex light barriers at each wheel. They are directed towards the floor and can accurately detect driving over a white line on the green floor. A millisecond sampling rate of the eight reflex light barriers' outputs allows to improve the precision of positions determined by odometry considerably.

The goal keeper can further improve its position calculation by taking into account the position of the walls building the goal. If a robot moves within the goal or near to the goal it can measure its distance to these walls. This is particularly helpful since the robot's viewing field is quite restricted in these cases. Therefore, the Paderkicker robots are equipped with four infrared sensors for measuring the distance of the walls into backward direction and to both sides.

3. Modular Paderkicker design

3.1 Functional architecture

The main functional units for driving, ball handling and vision were designed in a modular way as mechatronic functional units (MFU). Each sensor-actor-group is controlled by a separate microcontroller. Originally we planned dedicated designs of mechanical, electronic and embedded system for each MFU to be realized in an independent physical component. Due to cost and time restrictions however, requirements for control of the functional units and electrical motor drivers were gathered and only one microcontroller and driver board was developed to be "universally" used for actor/sensor control and motor driving. Hence, MFUs could not be realized as single physical components. However, logically each microcontroller and driver board belongs to exactly one MFU, which allowed their independent development since resource conflicts are avoided per se.

A Paderkicker robot now consists of a central behaviour module and system control module realized on an embedded Mini-ITX PC board and a microcontroller and the MFUs for driving, ball handling and vision mentioned above, which are further divided into sub-modules as depicted in Fig. 8. The figure also shows the information flow between these modules.

The *driving module* consists of five submodules, one for each of the four wheels and one for central driving coordination. Each wheel is controlled by a separate AVR microcontroller to which also sensors and drivers belonging to this wheel are connected. This microcontroller handles the inputs from several sensors. The actual current consumption and the temperature of the motor as well as the acceleration over this wheel in two axes are calculated. Also two reflex light barriers for recognition of white lines on the soccer field are connected to this board. For goal keepers the AVR controllers for the back wheels also process the inputs from the infrared distance sensors. At the output side an H-bridge is connected for producing the PWM (pulse width modulated) signals driving the motor. This module communicates with its environment via a CAN bus commonly used also in the automotive domain.

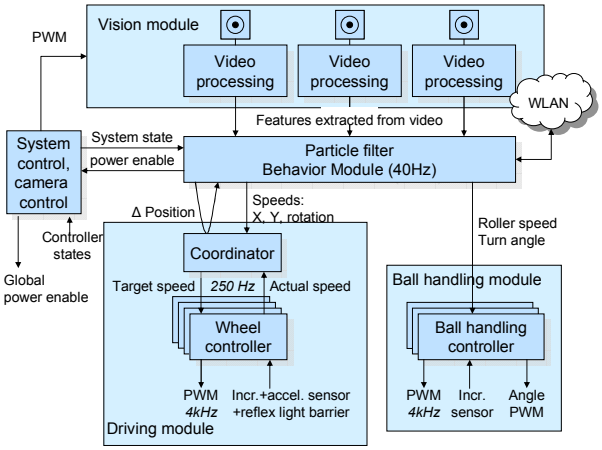


Fig. 8. Paderkicker module structure

The driving coordination submodule is realized on the fifth AVR microcontroller, which is connected to the central PC board via USB. It receives the measurements of the four wheels from the respective microcontroller via CAN, calculates odometric and sensor data and transmits them to the PC via USB.

The *ball handling module* is divided into three submodules two for controlling the two side rollers and a third one for handling the front roller together with the kicking solenoid. For these submodules a separate AVR microcontroller exists one for the side rollers and one for the front roller and solenoid. The third microcontroller (responsible also for kicking) is connected via an H-bridge to a cascaded voltage doubler that generates the fourfold supply voltage (about 100 – 120 Volt). This voltage is needed to load a capacity of 12 mF for moving the plunger. Via an IGBT device the solenoid can be fired. Thus the solenoid cannot only be fired; also its kicking force can be influenced by switching off the power supply while discharging.

The *vision module* has two tasks. On the one hand it separately processes the images perceived by the three cameras as described in Section 4. These results are fusioned by a particle filter. Video processing and particle filter are realized on the PC board. On the other hand it is responsible for generating the PWM signals controlling the pan-tilt units of the cameras which are realized by model craft servo motors. The same technique is also used for controlling the servo motors that adjust the joints (and hence position) for the ball handling rollers. For generating the six PWM signals for all three cameras the same AVR microcontroller is used, which is also responsible for the central system control.

3.2 Hardware architecture

The functional structure described above is mapped onto a hardware architecture as depicted in Fig. 9. The central processing unit is a Pentium M ULV PC running under Linux. Here the vision algorithms (see Section 4), the particle filter and the behaviour-based system (see Section 5) are realized. For control purposes of the modules described above in total eight AVR microcontroller boards are used. This board was developed only once for all modules (Fig. 10, Fig. 11). It is equipped with an AT90CAN128 microcontroller, which al-

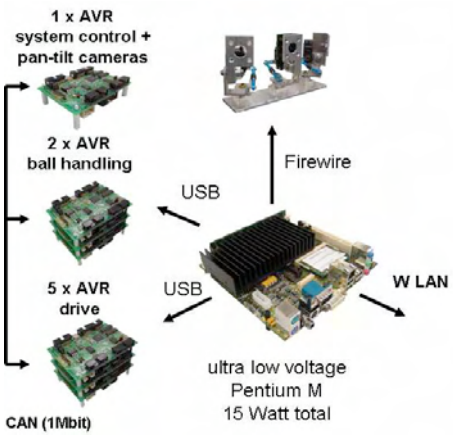


Fig. 9. Paderkicker hardware architecture

ready comes with a CAN bus interfaces. Two USART outputs provided by the microcontroller are connected to an interface component (FTDI) which realizes an USB 1.1 slave interface. This interface is handled by the Linux PC on the Mini-ITX board like a usual file interface. All digital inputs and outputs of the AVR microcontroller are connected via opto couplers (Opt.) or drivers respectively for security reasons. For debugging the status of all inputs and outputs is visualized by LEDs.

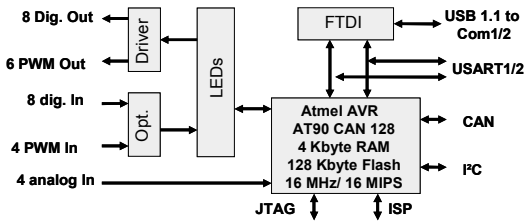


Fig. 10. AVR board (principle structure)

Fig. 11. AVR board (photo)

3.3 Software architecture

3.3.1 Team communication

In a soccer team the robots do not only act autonomously but they have to coordinate their actions. Therefore they need a communication infrastructure. Furthermore they have to react on the referee's commands and their parameters have to be adapted to the actual soccer field and its environmental conditions. The Paderkicker robots use a central team server for this purpose (Fig. 12). Each robot has to register when it is ready to play. Also different master panels (e.g. the referee box) concerning the team as a whole or concerning the initialization or operation of a specific robot can connect to the central team server. The communication is realized by a TCP/IP network with fixed addresses (LAN and WLAN). Particular attention has to be given to a robust WLAN connection, since this connection tends to

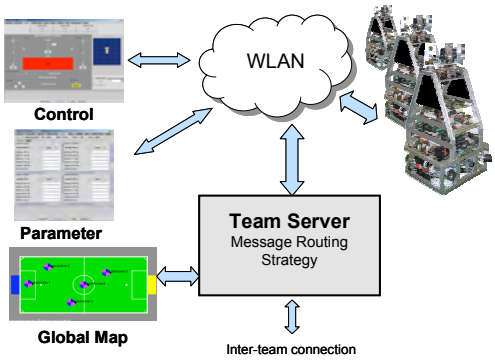


Fig. 12. Paderkicker communication structure

break several times during one halftime of fifteen minutes. A robot that has to restart its entire system from scratch each time it looses the WLAN connection, will not be able to actively participate in the game. Therefore the Paderkickers were designed very robustly against WLAN break down. They carry on with their actual autonomous behaviour and try to re-connect with the network without restart.

3.3.2 Logical robot architecture

The software architecture of a single Paderkicker robot can be described from two viewing points: the processes and threads responsible for processing the different kind of data and the logical or functional separation of tasks and the according data flow.

The *logical software architecture* is structured according to the triple tower model of Nilsson (Nilsson, 1998) that distinguishes between *perception, model* and *action tower* (Fig. 13).

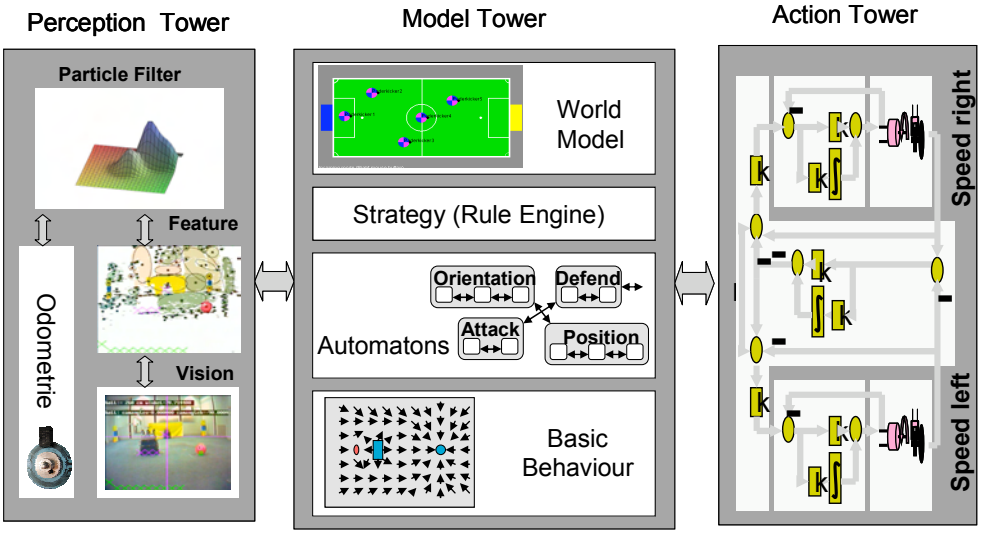


Fig. 13. Paderkicker software architecture

The *perception tower* is responsible for processing and fusioning the sensor inputs and providing them to the model tower in a more compact problem specific way. At the lower level features are extracted from camera images. Also information from the driving system's sensors and from the touch sensors of the ball handling mechanism are evaluated. At the next higher level the sensor information is pre-processed. For instance the debouncing of touch sensors or the different stages of image processing (see Section 4) determining the position of various objects like goal, lines etc. in relation to a single robot belong to this layer. Also the odometric calculations belong to this layer.
Since the Paderkicker robots have four omniwheels, an over-specified system of equations has to be solved. The distance driven by each wheel is sampled with a rate of 100 Hz. Via a table driven model determined by a modelling tool according to a robot's geometry the actual translation and rotation angle of a single robot is calculated. At the next higher level this data is fusioned with the positions calculated for other objects by a particle filter (Bayes filter).
A typical characteristic of the model craft servo motors used in the Paderkicker robots is that they do not deliver any feedback about their actual position. Hence we use a model driven approach to estimate the previously set angle of the servos. This model driven position estimation uses sensor information at various levels of abstraction. If for instance the ball has to be at a proper position for the ball handling system, this position estimation component needs the most recently perceived position information for the ball before it is filtered by the particle filter. In other cases where for instance the reliability of the ball position is more crucial due to its larger distance the filtered position may be preferred although it is received with a small delay. This information may for instance be needed if a robot has to drive towards the ball. Therefore, it has to be decided separately for each behaviour to which degree its sensor information has to be preprocessed.
The *action tower* realizes the control of the actor systems. Here the physical signals are generated for the nominal values in the form needed by the actors. Via specific hardware components on the AVR board PWM signals are generated or analogue values for voltage or current are measured for controlling the motors. Simple control loops are only used for the driving motors and the roller motors. We use cascaded PID controllers for current and revolution control. Their nominal values are calculated by the driving coordination module on the basis of the actual translation and rotation angle for each wheel.
For the future it is desirable that slight changes of the driving system mechanics resulting from maintenance or reconstruction are handled by an automatic calibration. Also adaptive control techniques should be used for adjusting controller parameters automatically. This is particularly desirable, since the maximum speed of a robot depends on the actual underground and its characteristic rolling friction which may change frequently. As always a certain reserve has to be granted by the controller in order to allow for robot turns also at maximum speed, a controller that automatically adapts to the actual rolling resistance would be advantageous.
The *model tower* is responsible for planning and selecting appropriate actions at various layers of abstraction. The lowest level, the so called security level, takes care that a robot is switched off in critical situations, e.g. if due to overheated motors or empty accumulators a robot might damage itself. On top of this security level a reactive layer is realized using the motor schemes developed by Arkin (Arkin, 1998) (see Section 5). This reactive layer is controlled by the next higher level that determines action sequences for a robot. At the topmost

level the playing strategy of a robot is coordinated with its teammates. Via WLAN the information about the positions of goals, opponents or other obstacles are distributed in the Paderkicker team. This information is used to determine which role (attacker, defender, etc.) a robot should take (see Section 5).

3.3.3 Process architecture

From the process point of view there is a main process called Brain running on the ITX PC board that opens several threads. The main process is responsible for action planning. One thread works as internal router for exchanging messages between other processes or threads. Per interface connected to the Brain process another thread is opened. There exist three interfaces one to the driving system, the ball handling and the pan-tilt unit of the vision system. Each of these interfaces is realized via a USB connection of the ITX PC to the respective microcontroller. In principle each microcontroller can be viewed as one additional process, since no operating system is used on the microcontrollers. Among each other the microcontrollers are connected via CAN bus.
For each camera one process is started for feature and object recognition. This information is passed to the router thread of the Brain process. Also the particle filter is realized as separate process that communicates to the Brain process via the router thread.

3.4 Message format

For integration of the separate Paderkicker components that communicate over different bus systems (CAN bus, USB, TCP/IP for communication between robots) a homogeneous message format was designed. Also the different master panels use this format. If a robot component is changed or a new panel should be integrated, a new message has to be defined, if the existing ones should not cover this functionality. For the Paderkicker robots an XML format was defined where all messages are registered. These data is used to generate access routines in different programming languages (C, C++, Java, Python) in a semi-automatical way. The need for a message format that is both robust and computationally cheap led to the following design (Fig. 14).

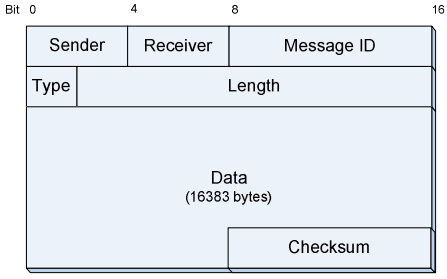


Fig. 14. The Paderkicker message format

Every message is preceded by a message start delimiter. Thereby, each subsystem is able to figure out the starting point of messages in continuous data streams with low processing overhead. Fields to uniquely specify sender and receiver follow. By these fields the different subsystems are addressed, as e.g. the behaviour system, the particle filter, or the TeamServer. The message ID specifies the actual meaning of the following data bytes. Thereby it

is also possible to extend the current message set by not yet foreseen message possibilities. The type describes the nature of the message, which provides the following possibilities: *set*, *request*, *info*, and *poll*. This is needed for certain message IDs where it is possible to set or get the according data, or even request the specified data to be delivered recurrently. Finally, the checksum provides means to recognize errors in the message.

4. Vision

Research regarding computer vision is done mainly in the area of real-time image processing. An optimized algorithm for low latency real-time colour segmentation (Stichling & Kleinjohann, 2002b) now was adapted to run on a PC under an ordinary Linux system. It even performs well on a PDA and was formerly implemented on a Trimedia TM 1100 video processing board running at only 100 MHz. A more detailed description follows later on. Three digital Firewire cameras are mounted on pan-tilt units to cover the whole 360° view being used as an active vision system. This configuration leads to an overall higher resolution and to a larger viewing area in terms of visual depth, compared to an omnivision system. Hence, the robots are capable of seeing distant objects much better. Higher visual coverage of the environment will become more important in the near future when the field will likely be extended in size again. The Firewire cameras are directly connected to the Mini-ITX board running a Linux system where the according video streams are processed simultaneously. In the following first our image processing algorithms will be described. Afterwards we deal with the recognition of RoboCup specific objects like ball, field, lines, etc. and describe a future improvement of image processing based on the discrete cosine transform.

4.1 Colour segmentation

In the environment of RoboCup special colour codes are used to make sure that objects are distinguishable from each other. Therefore a real-time colour segmentation of the camera streams is essential for the system. The above mentioned algorithm is optimized for an extremely low latency, so the overall latency decreases as the depending behaviour system can react faster. The main difference of the used colour segmentation algorithm compared to other segmentation algorithms is its linear processing. The algorithm processes an image line by line and occurring data dependencies refer only to already obtained data. Assuming adequate processing power, the latency of the colour segmentation is the transmission time of one image plus the computing time of the processing steps for the last line.

In a first step the algorithm does a so called "Line-based Region-Growing" in which regions with similar colour are grown while stepping through the individual pixels of the line. In a second step the just created regions are merged together if they meet defined colour-similarity and spatial distance criteria.

As this method was planned to run on embedded devices, the computational need of the algorithm had to be as low as possible. Therefore the underlying data structure has to be simple, especially in terms of computability. In computer vision algorithms moments are often used (Prokop et al., 1992) as they are significant and easy to compute. In this case moments $M(p,q)$ up to second order are used as region descriptors. For a region, the moment $M(p,q)$ is defined as follows:

$$M(p,q) = \sum_{(x,y)\in region} x^p y^q \tag{1}$$

The moments *M(0,0)*, *M(1,0)*, *M(0,1)* describe the number of pixels in x and y direction and the centre of a region. Additionally, to get the orientation of the moment, the central moments *C(p,q)* with the according centre (c_x,c_y) of the region are needed.

$$C(p,q) = \sum_{(x,y)\in region} (x-c_x)^p (y-c_y)^q \tag{2}$$

The central moments *C(1,1)*, *C(2,0)*, *C(0,2)* together with the above mentioned three moments describe one region. For visualization purposes they can be used as a representation for an ellipse, but as ellipses are computationally difficult to handle this is not used for the algorithm's calculations. Additionally the average colour of the regions is part of the descriptor, whereby the YUV colour space is used. The regions are stored as a connected list, whereby the region-growing step assures an ordering of the regions by the y-coordinate of the uppermost pixel of one region. This spatial ordering speeds up the following region-merging step. In this step the regions created before are merged together if they are spatially near to each other and are similar in colour (see Fig. 15). As the moments can be visualized

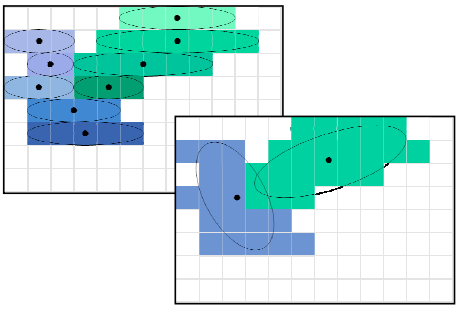


Fig. 15. Region-growing and region-merging

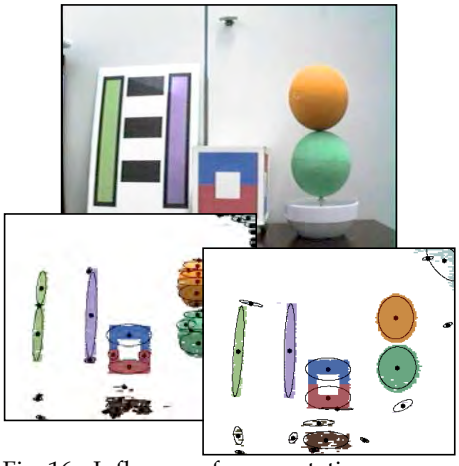


Fig. 16. Influence of segmentation parameters

as ellipses, spatial near can mean overlapping ellipses. The YUV colour space used for region-growing is not very suitable for region-merging because higher Euclidian distances in the colour space are treated as similar. In this step one would like to merge regions of similar colour disregarding the shading, therefore the HSI colour space is suitable for this purpose.

So, regions are merged together if their Euclidian distance in the HSI colour space falls below a specified threshold and if they overlap. The actual region is compared to regions whose lowermost pixel is not higher than the uppermost pixel of the actual region. Hence only the data from the line-based region-growing step of the actual image line and the pre-

viously created regions are needed to do the region-merging. Fig. 16 shows three different resulting images of the colour segmentation algorithm: In the background the original image, above two colour-segmented images. The lower left image shows segmentation with already found regions, but with a lower threshold for the allowed colour difference of regions to merge. In the other segmented image the elimination of shadows and of highlights can be observed.

4.2 General object detection / recognition

The detection and recognition of objects in general is based on the above described colour segmentation algorithm. The coloured regions found by the algorithm and the corresponding two-dimensional moments are checked for defined properties to identify objects like the ball, the goals or other robots. The pre-defined object properties are stored on an USB flash drive which is located at the on-board Mini-ITX board. Every time the robotic system is started the parameters are loaded from the USB flash drive. They can be read out and modified over a control applet which connects over a wireless connection to the central server where all robots are logged in. At the moment the parameters are loaded into the Trimedia respectively into the software vision module on the robots which are equipped with Firewire cameras. Recognized objects are checked for plausibility and deleted if necessary (e.g. when more than one ball has been detected or if the detected ball is outside the field).
In the application area of RoboCup special premises regarding the computer vision apply, e.g. defined colours for different objects. So, some colours are of more relevance than others. The above described colour segmentation algorithm does not benefit from this a priori knowledge, as it only uses the *achromatic threshold* to identify pixels of interest. For this reason, colour lookup-tables were implemented to make use of the prior knowledge of relevant colours. As a result fewer regions are extracted from the image, so that even smaller ones can be taken into account for object recognition. Thus, edge markers can be detected from a greater distance.
Using these lookup tables leads to a robust and fast possibility to assign pixels to an object without the colour space conversion from YUV to HSI for every pixel. This significantly improves the speed of the colour segmentation algorithm resulting in higher and more constant frame rates.

4.3 Field detection

Detecting the field starts at pixel-level of the camera image. Again, the before mentioned YUV lookup tables are used to decide whether a pixel belongs to the field. The camera image is divided into 36 x 44 fragments for which the number of field-pixels is counted. A threshold decides whether this fragment is part of the field. Another matrix of the same size is used to mask the outer dimensions of the field. This method allows detecting the field even if it is hidden by parts of a robot. Another advantage of the field detection is that it can be used to mask out objects outside the field (like ball-like red shoes of a viewer). As this detection runs in parallel to the actual vision system, it does not interfere with it, leading to more stable frame rates.

4.4 Landmark detection

For a robust localization the detection of landmarks is very helpful in particular if the landmarks are unique. In the RoboCup environment landmarks have defined colours and dimensions and additionally they are assigned to a certain side of the field. In Fig. 17 two different landmarks are shown. They normally mark corners of the field. In this vision-system-test-case the two different landmarks are located on the same side of the field and in addition no goal is present.

The vision system draws rectangles around recognized landmarks and writes the corresponding internal number aside. Landmarks are recognized using the coloured moments from the colour segmentation algorithm by successively processing the list of coloured moments, so that landmarks can be detected on-the-fly. Termination criteria make sure that not every moment is checked against all others. A landmark in the RoboCup environment consists of three coloured regions of the same size, whereas the two outer regions are even of the same colour. The outer colours are defined by the side and are of the same colour as the goal. Hence, the list of coloured moments is searched for corresponding moments which match an alignment and colour combination with a specific variance. A weight is assigned to found landmarks, depending on the number of appropriate coloured moments, whereupon a landmark is considered complete when the weight is three. Complete landmarks are no longer used for further combination-searching, avoiding the recognition of multiple landmarks at the same location. The dimension of the landmark is approximated using the height of the middle moment as this turned out to lead to a stable recognition.

4.3 Ball detection

For the detection of the ball, again the extracted coloured moments from the colour segmentation algorithm are used. The ball is an object with defined colour and size, and the list of coloured moments is searched for corresponding ones. Moments which are over the bottom line of landmarks recognized to be complete are ignored as this would be a ball outside the field. Suitable regions that are spatially near are merged together. Depending on the illumination of the field a dark shadow is under the ball. This shadow could be detected as an obstacle directly in front of the ball. Hence this shadow is masked to avoid this behaviour. The masking can be seen in Fig. 18 directly below the recognized ball. Under certain circumstances it is possible that multiple coloured moments match criteria of the ball. In this case of multiple possible ball locations, the largest one with the lowermost position in the camera image is taken as the ball position. Unfortunately this is not always correct, so that the actually used ball position is calculated by a particle filter which is fed with all possible ball positions.

4.5 Free Space detection

Another essential algorithm for autonomous robots that are part of a dynamically changing environment is a detection of free space. This information then is used to avoid obstacles and for general path-planning. In general the free space detection is very similar to the field detection but the two algorithms differ in detail. Like in the field detection algorithm the image area again is divided into 36 x 44 fragments. They are checked for "dark" pixels and if the amount of dark pixels exceeds a certain threshold, the corresponding fragment is marked being occupied. Afterwards the matrix is passed through column by column from bottom to top and at the image position of the first occupied matrix-fragment an obstacle

marker is placed. These obstacle markers are visualized as red bars as depicted in Fig. 17. Special attention has to be paid to the fact that our robots are equipped with an active vision system, where all cameras have two degrees of freedom to be moved. Hence, it can happen that a camera sees parts of the robot itself, which are black coated. It must be prevented that these parts are recognized as obstacles. As the actual camera position and the camera's aperture angle is known, a calculation of a suitable *robot-mask* depending on the camera position is possible. This mask is also shown in Fig. 17, but as no part of the robot is visible, all robot-mask markers are at the bottom of the image.

Fig. 17. Landmark recognition, free space detection with obstacle markings and line segment detection

Fig. 18. Ball detection with shadow-masking

4.6 Line recognition

As the lines on the field are defined by the rules of RoboCup, they suit as distinctive attributes for localization. The lines in all divide the field into sections, but one single line is not characteristical for a certain section, because there are the same markings for both sides of the field. As the camera perceives only a section of the field, there are multiple combinations of lines possible, each belonging to another section of the field. Hence, line recognition is inapplicable as the only sensor input for localization; it has to be merged with other sensor data contributing to more overall accuracy in localization.

There are many approaches for line recognition, most of which are very expensive regarding the need for computational power. The gradient based Sobel filter had been implemented on the old Trimedia but was mutually exclusive with the colour segmentation. Hence, another way of recognizing lines had to be invented, which would have less need for additional computational power. With the change to the current PC-based vision system, a lack of computational power is no more existent, but this new method saves computation time which can be used e.g. for the behaviour system. The field-detection algorithm described above seemed to be suitable to calculate a line-recognition-like operation along the way. The image is passed through bottom-up column by column. In the case of a field-fragment which is followed by a mostly white fragment which is followed again by a field-fragment, the inner fragment is assumed to be a so-called line-fragment. The fragments found by the algorithm are marked in the output image as shown in Fig. 17. This algorithm is not line

recognition in terms of vectorizing recognized lines, but rather calculates supporting points of possible lines, which then can be matched by the particle filter.

4.7 Structural simplification of images using a discrete cosine transform

One of the main challenges of computer vision is the extraction of distinct image features. A rule of thumb is: "The simpler the image, the simpler the extraction". One method to simplify images is to reduce the number of used colours before starting the colour segmentation. In the following an according method is described, which utilizes the effect of a discrete cosine transform. The discrete cosine transform is broadly used in image compression algorithms, known from the most popular image compression algorithm "JPEG". The mathematical definition of the DCT-II is as follows:

$$X(n) = \frac{2}{N} C_n \sum_{k=0}^{N-1} x(k) \cdot \cos\left[\frac{\pi(2k+1)n}{2N}\right], \; C_n = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } n = 0 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

n=0,1,…,N-1 and *X(n)* is called the *n*-th spectral component. Applied to a block of pixels, it transforms the pixels into the frequency domain, where high frequencies are sharp edges in

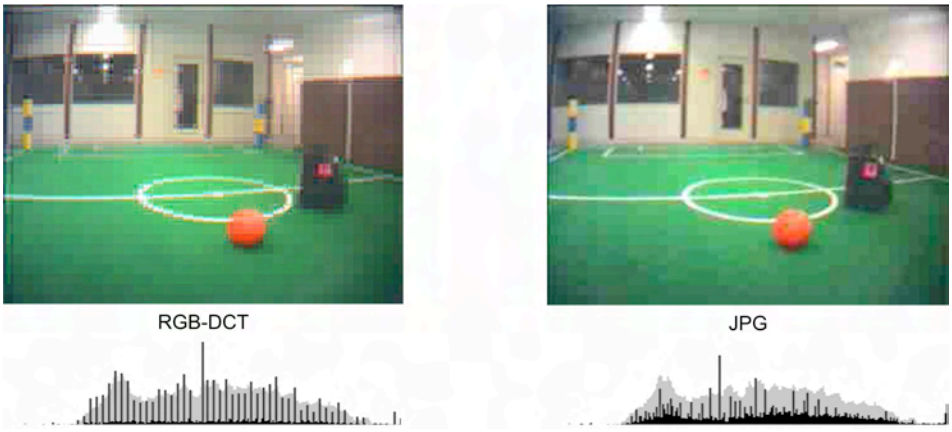


Fig. 19. Resulting colour distributions of DCT applied to the RGB colour space in comparison to ordinary JPEG compression

the image. An image compression is almost always done by cutting off higher frequencies. Talking of frequencies means that the image is transformed into the frequency domain. Then, changes are made to the frequencies, e.g. the higher ones are cut off, and at last the frequency data is transformed back to the time domain. Here, a JPEG-like "compression" is done, but not like in JPEG, the colour space is RGB. Actually the only "compression" consists in reducing the bits used in the frequency domain. Later on this method is referenced as "RGB-DCT". It can be used to reduce the structural complexity of images, e.g. by reducing the amount of used colours, while paying attention to the colour balance. This reduction has to be done in a special way, so that the overall distribution of colours remains unchanged. In Fig. 19 the resulting distributions of the application of the RGB-DCT and the

standard JPEG compression are compared. The resulting distributions are each drawn in black over the distribution of the original image drawn in grey. The RGB-DCT's distribution follows the original one's very accurately unlike the resulting distribution of the JPEG compression. On screen better than printed, the colour distortion caused by the JPEG compression is apparent.
This effect can be utilized as a pre-processing step before the above described colour segmentation algorithm. The RGB-DCT applied to an image leads to smoother images: firstly the image is transformed into the frequency domain. Then, a significant amount of higher frequencies is cut off. At last, the image's data is restored by back-transforming. As now higher frequencies (sharp edges) are reduced, the image is smoother than before. A fact for colour segmentation in general is: the smoother the image, the bigger the moments; as edges can prevent the merging of two regions with similar colours. Of course, there has to be found a balance between over-sharp and over-smooth images. If you use the RGB-DCT prior to the colour segmentation, it can significantly reduce the parameterization effort due to a reduction of the image-complexity. Badly parameterized colour segmentation can lead to an image scattered with many very small moments, and that easily happens. The prior application of the RGB-DCT leads to a significantly better colour segmentation result. The total amount of moments decreases and the average size of a single moment increases (Fig. 20). In this particular example, recognition of the ball is impossible with the left-hand side colour segmentation but is possible with the right-hand side's one. Here, the same colour-segmentation parameters were used, while in the right-hand's image the RGB-DCT was applied before the segmentation – with only 6 bits of relevance in the frequency domain.

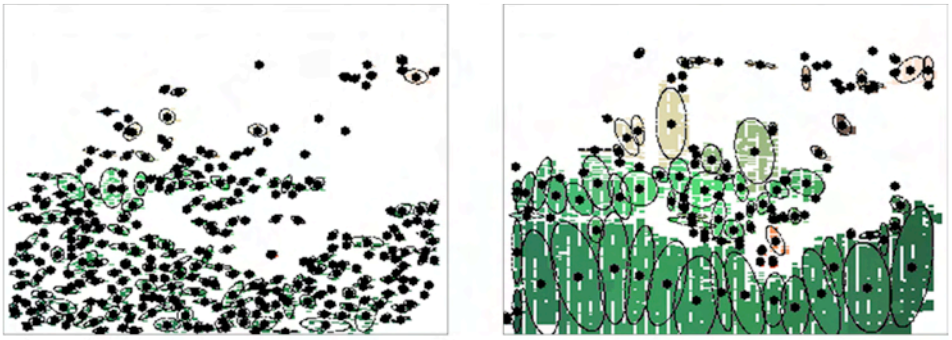


Fig. 20. Resulting colour segmentation moments with and without prior application of RGB-DCT, while using the same colour segmentation parameters

5. Behaviour-based system

Abstracting the hardware perception and action execution, the behaviour-based software is structured as can be seen in Fig. 21. There we distinguish between the *world model, strategy, automaton, and behaviour module*. All modules have to be autonomous in a way that keeps the robot full functioning even in case it has lost wireless network connection to its teammates

which is not uncommon under typical tournament conditions. In this case, the robot sticks to its last committed strategy.

Once the perception is pre-processed, the extracted information is stored in the robot's individual *world model module*. The world model module captures information specific to the robot itself regarding

- the robot's unique identifier
- its pose on the field (x, y, orientation),
- the position's confidence,
- the 2D ball-position as seen by the robot,
- the ball-position's confidence,
- the current role of the robot, and
- possible roles.

In addition it holds team information, like e.g. the global map, the teammates' roles and the game situation. The information regarding the robot's possible roles is a bit mask specifying which role the robot is able to fulfil. A goalkeeper, e.g., is not allowed to switch its role during the whole game. This can be specified by setting possible roles to only the goalkeeper role. All remaining modules (strategy, automaton, and behaviour), are allowed to access the world model module's information.

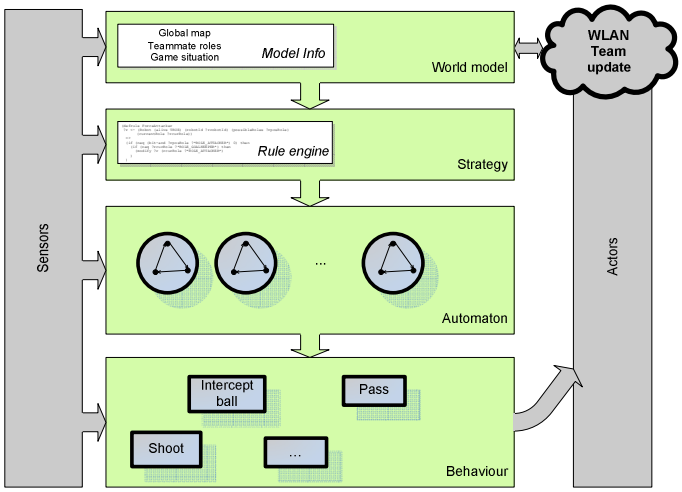


Fig. 21. Behaviour-based control flow in the Paderkicker robot

The world model information is sent to the TeamServer at 4Hz. The TeamServer will be described in the next section. It aggregates the information, decides upon a team strategy, and sends the whole information to each teammate again at 4Hz, so that the individual robot's world model information can also be treated as a life beat. Once, a robot has lost connection and is thus not transmitting its world view to the TeamServer, it is considered as lost and the strategies are reassigned within the team.

Depending on the teammates' world models the *strategy module* then decides on the current strategy for every robot. Strategies include e.g. *Defend* and *Attack*, but also standard situations like *Kick off* or *Penalty*. This is done by the strategy module's rule engine.

After the strategy module has committed to a role, the strategy is realized by an according finite state machine of the *automaton module*. Its states include e.g., ball facing, or staying between goal and opponent.
Every state in the automaton module is mapped to a low-level reactive behaviour like "move to ball" or "kick ball" in the *behaviour module*. This module finally executes the actual actions by sending the calculated action vector to the corresponding hardware boards. It is behaviour based in terms of Arkin's Motor Schemes (Arkin, 1998). Our behaviour system allows for a distinction between cooperative and competitive behaviours and behaviour control through time excited evaluation functions. It consists of a set of low-level behaviours represented by vector fields that have to be combined in order to result in a vector that can be sent to the actuators.

5.1 Team strategy

The strategy of the Paderkicker team is realized by role arbitration based on the robots' propagated world models. There are different ways to actually implement a team strategy module. One way is to use a dedicated server process that distributes roles and commands to the teammates after each of them has provided the TeamServer with its subjective world view. Another way is to design the system responsible for strategy arbitration in a distributed way, so that every robot has the same rule base. Each teammate would then have special rules for the team strategy by which it could independently come to a decision in a given situation. We started with the second solution, implementing our team play as a distributed expert system. However, when the need arose to form a mixed team at the RoboCup world championship in Bremen 2006 we chose the first solution with a dedicated server being able to connect teams with completely different hard- and software. This led to the development of the *Paderkicker TeamServer* running a separate expert system for the cooperation of robots from different teams. The interaction with a TeamServer is particularly important for mixed teams, which will become more and more common in the future. The TeamServer acts on a coach level as in real soccer games. The different robots register at the TeamServer and propagate their individual world model as described before – with the only difference that the whole team thus has only one expert system running responsible for strategy arbitration. Although the teammates now rely on a central server this does not have any impact on the individual robot's robustness: Even if each robot had its own role arbitration module the individual expert system would not trigger any new role without having a network connection in the first described solution.
The core of the TeamServer is the rule-based expert system *Jess* (Friedman-Hill, 2005), which takes as input the teammates' world models. In addition, it is directly connected to the referee box, thereby getting additional game status information, like *game start* and *stop* and special game situations like e.g. *corner kick*. Besides functioning as the team communication router the TeamServer has the task to arbitrate roles for the teammates. This is very intuitive by using the declarative programming methodology of the Jess expert system. Thereby, the expert knowledge is human readable and can be specified as "what is to be solved" and not procedural as "how something should happen". The system relies on the expert system's ability even under incomplete world information. It holds facts in its working memory, which are actually variables of the actual world models sent by the teammates, e.g. "ball is in the perception range of the robot". The rule-base holds the domain specific expert knowledge coded as rules. A typical rule consists of preconditions and actions that are to be exe-

cuted when the preconditions hold. The rule engine's pattern matcher then matches the rule premises against the facts in working memory and creates an agenda for execution of the activated rules.
The rule-based system is fed with the subjective world models of all teammates. In our example if e.g. the ball is close to a robot a rule will fire that changes the role of the robot to attacker under certain conditions and reassigns the previous attacker a different role. As part of that process the rule that forces the robot closest to the ball can be seen in Fig. 22. It uses the facts *alive*, *robotId*, *currentRole*, and *possibleRoles*. If the premise (the part in front of the "=>") matches, the action specified subsequently is executed. This means that in order

```
(defrule ForceAttacker
 ?r <- (Robot (alive TRUE) (robotId ?rrobotId) (possibleRoles ?rposRole)
       (currentRole ?rcurRole))
  =>
  (if (neq (bit-and ?rposRole ?*ROLE_ATTACKER*) 0) then
    (if (neq ?rcurRole ?*ROLE_GOALKEEPER*) then
      (modify ?r (rcurRole ?*ROLE_ATTACKER*)
    )
  )
```

Fig. 22. Example for a role-arbitrating rule in the TeamServer: The teammate with the smallest distance to the ball is enforced to be attacker, but only if the robot is allowed to be Attacker

for the rule to be executed there must be a robot in the working memory, which is alive and has sent its possible roles, its ID and its current role. Side conditions can be easily specified in such a system: In the example the goalkeeper is omitted from this specific rule.
At the moment the Paderkicker team supports the five following roles:

- Attacker
- Goalkeeper
- First and second line defender
- Supporter

As the robots may fade away because of network connection or hardware problems while the game is running, some roles will not be possessed by any teammate. Therefore, the TeamServer has to prioritize the individual roles by their importance for the team. The priorities are dependent on the ball-position on the field. In case the ball is in the team's half of the field, defending the goal is the most important aim, which leads to the following priorities:

1. Attacker
2. Goalkeeper
3. First line defender
4. Second line defender
5. Supporter

In the other case, a more offending style is advisable:

1. Attacker
2. Goalkeeper
3. First line defender
4. Supporter
5. Second line defender

This means, that if one teammate drops out of the team the teammate with the least important role is assigned the lost teammate's role.

5.2 Automatons realizing the strategy

Once the role is assigned to a robot, it executes the finite state automaton for that particular role in the current game situation. It is only dependent on the robot's own perception, i.e. if the TeamServer performs a new phase of role arbitration due to game situation changes or

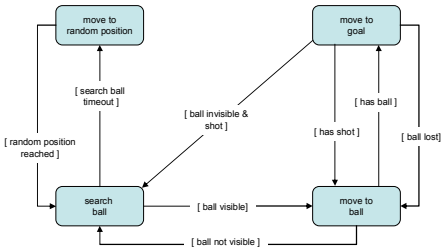


Fig. 24. Attacker's strategy as a finite state automaton

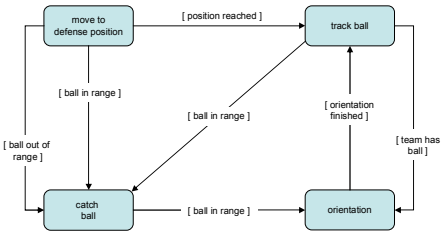


Fig. 23. Line defender's strategy as a finite state automaton

because a robot has been lost, the role change of a teammate results in a switch to the new finite state automaton corresponding to the new role and game situation.

The attacker's duty is to find the ball (with the help of all other teammates' world models including the ball-position), intercept it, drive to the opponent's goal and shoot (Fig. 24). The line defenders are assigned two lines of defence orthogonal to the goal and at different distances (Fig. 23). The line defenders have to work together in order to intercept the ball before it goes into the own goal or leaves the field (Fig. 25). If only one defender is available in the team it tries to stay between the ball and the goal. The supporter has the task to stay near the attacker to be the first choice to become the new attacker, if the old one has lost the ball.

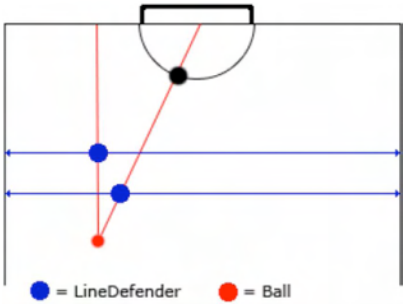


Fig. 25. Positioning of the two line defenders

5.3 Low-level behaviours

Each state in the automaton is realized by a reactive behaviour assembly which is mixed out of one or more low-level behaviours like, for instance, ball dribbling, tracking an object with

the cameras or moving towards a specified object. After all behaviour assemblies have been processed with the actual perception input, a result vector $R = (r_s)$ is generated out of the assemblies' individual outputs, specifying a value to set for every servo s of the Paderkicker, which has the following capabilities:

- 2D vector and orientation for the omniwheel motor
- Position and speed of the three ball handling rolls
- Direction for the three pan-tilt cameras to look at
- Whether the shooting device should be activated or not.

For the mixing process of the behaviour assembly, two different modes exist:

- *Competitive mixing* chooses the result of the strongest low-level behaviour as the output for the whole behaviour assembly, also known as the *Winner-Takes-All* strategy.
- *Cooperative mixing* combines all low-level behaviour's by a weighted sum.

E.g. two-dimensional movements are typically generated by using cooperative behaviour mixing, while competitive mixing is more appropriate for calculating the direction of the pan-tilt camera.

The mixing operator takes a fixed-sized vector $\vec{C}_b$ for every low-level behaviour b as input which specifies the desired parameters for each available actor possibility:

$$\vec{C}_b = \begin{pmatrix} (c_{1,b}, v_{1,b}, m_{1,b}) \\ \vdots \\ (c_{n,b}, v_{n,b}, m_{n,b}) \end{pmatrix} \tag{4}$$

The elements contain the actual behaviour value c, a vote v denoting the desired strength the value should be given in the final vector field, and the mode m telling the mixer whether this should be mixed competitively or cooperatively. In the same vector $\vec{C}_b$ modes can be different, whereas the modes for the same servo must be the same for all behaviours. If a behaviour does not want to set a certain servo, it can set the corresponding vote to 0. In addition to $\vec{C}_b$ the mixing operator needs so-called *gain* values g_b for each behaviour by which the automaton's state can configure the diverse low-level behaviours in the behaviour assembly. Together with the servo's value c, the mixing operator calculates the weight for it. Putting all together the result r_s for servo s in one cycle is done as follows:

1. Determine the weight $w_{s,b}=g_b \cdot v_{s,b}$
2. Calculate $r_s = \begin{cases} c_{s,b} \text{ , if } \mathrm{m_{s,b}} \text{ is competitive} \\ \sum_{b=1}^{B} \mathrm{k_{s,b}} \cdot c_{s,b}, \text{otherwise} \end{cases}$, $\mathrm{k_{s,b}} = \mathrm{w_{s,b}} / \sum_{b=1}^{B} \mathrm{w_{s,b}}$ is the normalized weight of behaviour b for a servo s.

6. Adaptation and learning

Especially in the RoboCup domain the need to increase the intrinsic robustness of the robots is obvious. Even more, as the environment gets more complex and uncertain the need for

the robots to consider every bit of information regarding the environment and themselves grows, in order to stay functioning in case of breaks, environmental changes, or other unforeseen events. By robustness we stick to the definition of the IEEE Standard Glossary of Software Engineering Terminology (IEEE, 1990):

> *The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.*

When considering societies of robots, each individual robot thereof has several different information sources to increase its own robustness (Fig. 26): It can foremost analyse its past behaviour and derive knowledge about e.g. which behaviour caused damage to the robot. If the robot has recognized a discrepancy between its expected behaviour and the actual result in its environment, it is even better, if it actively carries out experiments with the goal to find new behaviours that are suited better. Thereby, the robot in fact has to recognize dynamic entities like other robots as such so that it does not derive wrong hypotheses. In a society the robots even can propagate the knowledge within the robot group. And, as a last resort in case the robots cannot communicate, they can imitate each other so that the same behaviours do not have to be found out cumbersomely by each robot individually.

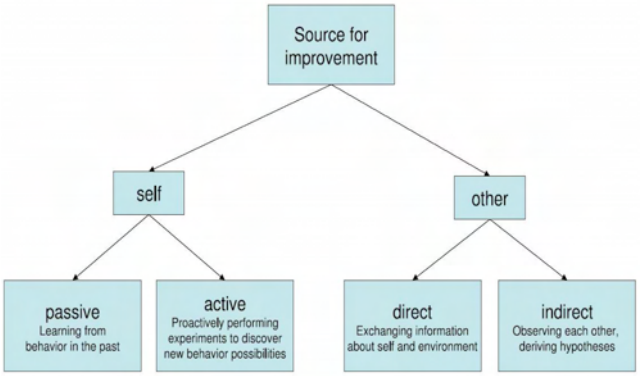


Fig. 26. A robot has several sources to improve its robustness available

In the RoboCup domain, however, the short duration of the game (2 x 15min) allows only for learning efforts that can quickly be conducted. In addition, the most environmental circumstances are guaranteed to not change during the game. Taken this into account, the changes the robot has to react on robustly are

- changes at the robot due to wear out and
- sudden breaks because of a collision with another robot.

In this section we will give an overview of our current research progress which addresses robustly learning low-level behaviours called "skills". The robust skill learning module (Richert & Kleinjohann, 2007) described in this section is meant to be placed between the behaviour module and the actual hardware components. It maps the vector resulting from the behaviour module to the actual servo possibilities. Thereby, the behaviour assemblies do not have to be adjusted, when something has changed at the hardware layer. Normally, the skill layer is turned off and the manually programmed behaviour assemblies are directly used to set the servos. The skill layer is turned on if the system detects discrepancies be-

tween the expectation and the reality regarding the environment's reaction to the system's action. The previously presented behaviour-based architecture (Fig. 21) thus has to be slightly modified, which results in the architecture in Fig. 27.

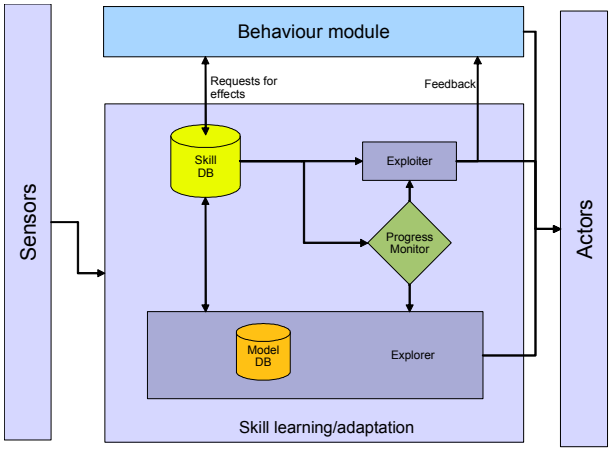


Fig. 27. Behaviour-based architecture with robust skill learning capabilities

The heart of the skill learning module is the skill database, which stores skills as tuple *(pc, a, e)* being the precondition the action and the expected effect in the environment. Normally, skills are requested by the upper layer in terms of effects that the behaviour module wants to happen, e.g. the decrease of the distance to a goal. If according skills are found in the database they are checked whether a skill's precondition holds true for the current situation. In the positive case the skill is directly executed and the corresponding effect is checked in the next cycle. In the other case, a change has been detected and the precondition of the skill is updated in a way that it is not executed in a similar situation the next time. If no skill can be found in the database that could achieve the requested effect the system starts tinkering with its actors until it finds an action that affects the requested effect. With the recorded perception data it has collected in this phase the following steps are processed:

1. Segmentation of the perception stream
2. For every segment
    a. Determine the model function most suitable and test it with the according model invariant
    b. Approximate the segment with the chosen model function
    c. Generate the skill capturing the segment's information

At first the perception stream is segmented into consecutive chunks of perception data according to the model function, which will be used to approximate the expected effects. For each segment a different function will be approximated resulting potentially in a new skill. An individual skill will be created to account for that. The choice of the model functions is subject to future research. As a first straightforward solution the system tries all model functions available and takes the one that is able to approximate the perception data with the least error. In the current implementation only polynomials are used. Experiments simulating both, gradual degradation and abrupt breaks, showed that though the modest learning speed, the skill learning module managed to robustly adapt to environmental changes.

There a robot had to repeatedly approach a goal while artificial breaks were introduced into its hardware.

In the first experiment the robot had to start with an empty skill base. After the 15th run a break was simulated by switching the first and second element in the actor. The robot thus had to cope with a skill base of which the most skills have become obsolete and new skills had to be learned. The result of how the robot copes with such a sudden break can be seen in Fig. 28. The steep increase of the running time is needed to delete skills that are not usable any more. At the 30th run it again has arrived at the old performance of run 15. As can be seen in the number of totally stored skills, not all skills had to be relearned.

The second experiment forced the robot to cope with gradual degradation of the omniwheel motor by decreasing its power in one dimension at every run. This was done according to the formula $(dx, dy) \leftarrow (dx, dy * \theta * (1 - run/30))$, with $\theta \in [0.1, 0.9]$ and run being the number of times the robot already drove to the goal. Thereby, the speed to drive left or right degraded to the fraction θ of the original capability at the last run. The result of how the robot copes with a continuous degradation over its whole lifetime for different degradation rates is shown in Fig. 29. It can be seen that the skill handles gracefully the degradation according to the grade of robot's damage: The bigger the damage the lower the performance. However, even at only 20% of its original power the robot still manages to cope with the impairment.

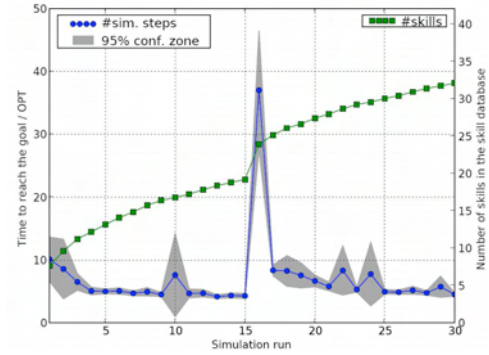


Fig. 28. Performance of robot coping with a sudden break after the 15th run: number of steps needed to reach the goal condition and the learned skills at each run. The grey background is the 95% confidence interval

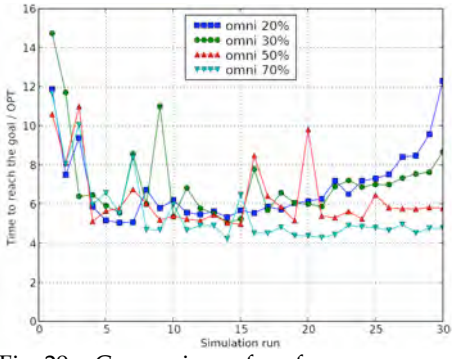


Fig. 29. Comparison of performance development for the robot coping with gradual degradation of one servo down to 20%, 30%, 50% and 70% of its original power (averaged over 400 trials)

The results are encouraging by showing how robust behaviour can be achieved at the skill level of mobile autonomous robots. They show that the robot is able to cope with sudden breaks in the hardware as well as gradual decrease of an individual component's performance. It is able to learn low-level skills guided by behaviour dependent predictions it recorded while the skill was learned. Even with simple model functions the system is able to learn skills without having a priori information about its servos. As an effect of the skill learning module not paying attention to the servo semantics and the way it dynamically creates, chooses, or deletes skills, it has all necessary properties of being robust in unforeseen environments. This also holds if the environment or the robot itself changes due to

wear out or harm caused by other robots. As long as there is a way to proceed and fulfil the goal the skill learning module will find skills to reach it.

7. Conclusion and Outlook

While realizing our soccer robot platform we achieved the two conflicting goals in the soccer domain needed to reach true autonomy: quick response rates for high reactivity and more complex but less frequent deliberation processes. This has led to robots that show autonomous behaviour while keeping attention to changes of the game situation demanding coordinated reaction of the whole team in a timely manner.
Components that are subject to quick changes in the environment as e.g. sensor and actor capabilities of the robots are arranged in a decentralized manner wherever appropriate and are located on specialized hardware. Those processes that do not need that fast update cycles like team communication, planning or processing at the higher levels are located at the Mini-ITX, allowing for faster development cycles but slower execution time. This architecture has evolved naturally out of the needs to combine a fault tolerant embedded system with fast development cycles for behaviour exploration.
In the near future we plan to endow the robots with the capabilities to self-calibrate the individual behaviours, meaning that the robots themselves adapt the behaviour parameters based on their perception of the environment. Furthermore, the TeamServer functionality will have to be decentralized to comply with the upcoming RoboCup soccer rules for the Middle Size League. A first step could be that one robot at a time hosts the TeamServer. If it looses connection to the other teammates another robot then could take over. Another big issue is the calibration automation for the various vision aspects. The usual practice to align the RoboCup rules year by year a little bit more to the FIFA rules has also a deep impact on the near-term Paderkicker development. The abandonment of the corner posts together with the colours of the goals, which are currently used by the Paderkickers to localize themselves, necessitates a partially rework of the vision and localization algorithms.

References

Arkin, R. C. (1998). Behaviour-Based Robotics, MIT Press

Beier, D., Billert, R., Brüderlin, B., Kleinjohann, B. & Stichling, D. (2003). Marker-less vision based tracking for mobile augmented reality. In *Proceedings of the Second International Symposium on Mixed and Augmented Reality (ISMAR 2003)*

Esau, N., Kleinjohann, B., Kleinjohann, L. & Stichling, D. (2003a). MEXI - machine with emotionally extended intelligence: A software architecture for behaviour based handling of emotions and drives. In *Proceedings of the 3rd International Conference on Hybrid and Intelligent Systems (HIS'03)*

Esau, N., Kleinjohann, B., Kleinjohann, L. & Stichling, D. (2003b). Visitrack - video based incremental tracking in real-time. In *6th IEEE International Symposium on Object-oriented Real-time Computing (ISORC '03)*

Friedman-Hill, E. (2003). Jess in Action: Java Rule-Based Systems (In Action series). Manning Publications, December 2002

IEEE (1990). IEEE Standard Glossary of Software Engineering Terminology.

Nilsson, N. (1998). Artificial Intelligence: a New Synthesis. Morgan Kaufmann

Prokop, R. J. & Reeves, A. P. (1992). A survey of moment-based techniques for unoccluded object representation and recognition. *Computer Vision, Graphics and Image Processing. Graphical Models and Image Processing*, Vol. 54, (Sept. 1992), pp. 438-460

Reimann, C. (2005). Kick-Real - a mobile mixed reality game. In *ACE2005, ACM SIGCHI International Conference on Advances in Computer Entertainment Techno*logy

Richert, W., Kleinjohann, B., Kleinjohann, L. (2005). Evolving agent societies through imitation controlled by artificial emotions. In M. Huang, X.-P. Zhang, and M. Huang, editors, *ICIC 2005*, number 3644 in LNCS, pages 1004–1013. Springer-Verlag Berlin

Richert, W., Kleinjohann, B., Koch, M., Bruder, A., Rose, S., Adelt, P. (2006). The Paderkicker Team: Autonomy in Realtime Environments. *Proceedings of the Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006)*

Richert, W. & Kleinjohann, B. (2007). Towards Robust Layered Learning. In *IEEE International Conference on Autonomic and Autonomous Systems (ICAS'07)*

Stichling, D. & Kleinjohann, B. (2002a). CV-SDF - a model for real-time computer vision applications. In *IEEE Workshop on Application of Computer Vision*

Stichling, D. & Kleinjohann, B. (2002b). Low latency color segmentation on embedded real-time systems. In Bernd Kleinjohann, K.H. Kim, Lisa Kleinjohann, and Achim Rettberg, editors, *Design and Analysis of Distributed Embedded Systems*. Kluwer Academic Publishers

Stichling, D. & Kleinjohann, B. (2003). Edge vectorization for embedded realtime systems using the CV-SDF model. In *Proceedings of the 16th International Conference on Vision Interfaces (VI 2003)*

Stichling, D. (2004). VisiTrack - Inkrementelles Kameratracking für mobile Echtzeitsysteme. PhD thesis, Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, 2004.

15

Learning to Play Soccer with the SimpleSoccer Robot Soccer Simulator

Jeff Riley
RMIT University
Australia

1. Introduction

The RoboCup simulated soccer league (RoboCupSoccer) is an important and useful tool for multi-agent and machine learning research which provides a distributed, multi-agent environment in which agents have an incomplete and uncertain world view (Kitano et al., 1995; Kitano et al., 1997). The RoboCupSoccer state-space is extremely large, and the agent perception and action cycles in the RoboCupSoccer environment are asynchronous, sometimes resulting in long and unpredictable delays in the completion of actions in response to some stimuli. The large state-space, the inherent delays, and the uncertain and incomplete world view of the agents can increase the learning cycle of some machine learning techniques onerously.

There is a large body of work in the area of the application of machine learning techniques to the challenges of RoboCupSoccer (e.g. Luke, 1998a; Luke, 1998b; Ciesielski & Wilson, 1999; Stone & Veloso, 1999; Uchibe, 1999; Ciesielski & Lai, 2001; Ciesielski et al., 2001; Riedmiller et al., 2001; Stone & Sutton, 2001; Bajurnow & Ciesielski, 2004; Riley & Ciesielski, 2004; Lima et al., 2005; Riedmiller et al., 2005; Riley, 2005), but because the RoboCupSoccer environment is so large, complex and unpredictable, the extent to which such techniques can meet these challenges is not certain. More progress could be made more quickly if the complexity and uncertainty could be reduced: while tactics may differ due to uncertainty in the environment, high-level strategies learned in a less complex and more certain environment should transfer directly to a more complex and less certain environment.

SimpleSoccer[1] (Riley, 2003) was developed as an environment that reduces complexity and uncertainty sufficiently to increase the viability of machine learning techniques, yet retains sufficient complexity and dynamics to allow learning from SimpleSoccer to be directly transferrable to the RoboCupSoccer environment.

2. The SimpleSoccer Robot Soccer Simulator

The primary objective when creating the SimpleSoccer environment was to create an environment complex and dynamic enough that while low-level tactics may differ due to

[1] Full documentation and source code is located at http://www.rileys.id.au/SimpleSoccer.html

the removal of systematic uncertainty, high-level strategies directly applicable to the RoboCupSoccer environment could be developed, thus providing a simple yet sufficiently accurate model of the RoboCupSoccer environment that allows rapid learning. The design objective was achieved by modelling only the attributes of the RoboCupSoccer environment necessary to allow ball and player interaction with the provision of basic player actions, and by not modelling the client-server environment and systematic uncertainty inherent in RoboCupSoccer. The SimpleSoccer environment is comprised of :

- the soccer field
- fixed landmarks - the goals
- the ball
- up to two teams with a maximum of eleven players each.

The SimpleSoccer environment was inspired in part by simplicity of the Ascii Soccer environment (Balch, 1995), but is a more complex environment which more closely models the RoboCupSoccer environment.

2.1 The Field

The soccer field in SimpleSoccer is represented by a two-dimensional grid with the goal markers being the only landmarks available to players (Fig. 1) The goal area for SimpleSoccer, in keeping with the RoboCupSoccer field and goals, is a defined area at each end of the field. The boundaries in SimpleSoccer, except for the goal areas, are hard barriers which impede movement of the ball and players: the ball does not rebound from the boundaries. Both the field size (length and width expressed as a number of cells) and goal size (in cells) are configurable.

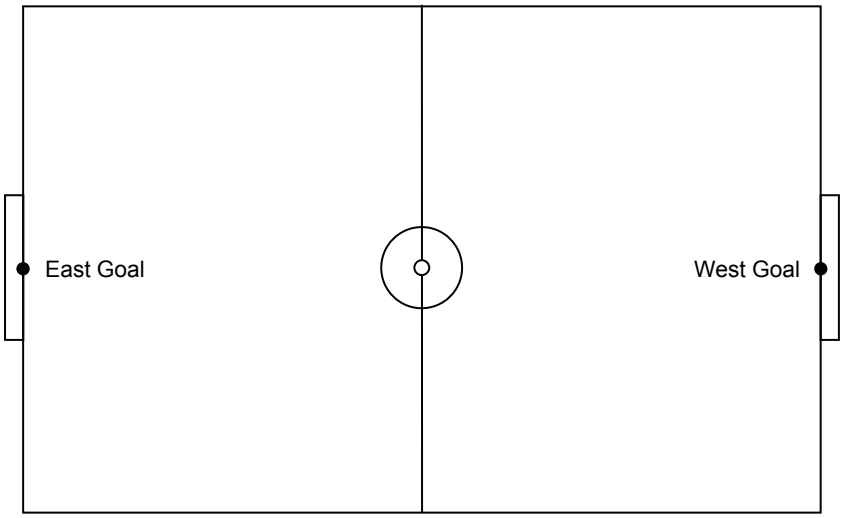


Fig. 1. Soccer field and landmarks in the SimpleSoccer environment (grid lines not shown)

2.2 The Players

Player movement and sensory capabilities in SimpleSoccer are similar to those of RoboCupSoccer. In the SimpleSoccer environment players may move in any direction, specified by a real-valued angle from 0.0 to 360.0 degrees relative to the player's current facing direction. Similarly, the ball can be kicked in any direction. Player and ball locations are specified by discrete grid co-ordinates, or cells: while movement and other actions can be in any direction, at the completion of an action, player and ball final locations are quantized to discrete cells. A player can only kick the ball if the ball is within a defined kickable distance (measured in cells) from the player.

Players in SimpleSoccer have a field of vision similar to that of RoboCupSoccer. Fig. 2 shows the range of a player's vision in the SimpleSoccer environment – players can see objects in a diamond-shaped area in the direction the player is facing. A player's viewing diamond is determined by the *view angle* and *view length*, and only objects of interest (ball, player or goal) within a player's viewing diamond can be seen by the player. The black circles shown in Fig. 2 represent objects on the field – only one is visible to the player in the diagram. At each time interval during a game all players are presented with the cell co-ordinates of, and direction (relative to the player's facing direction) and distance (number of cells) to any object of interest in the player's field of vision. Note that the information supplied to the player is limited to object location – no information regarding the movement of an object, either direction or speed, is supplied. The location, and hence direction to, an object is only known to a player if that object is visible to the player. Players may infer the location of objects based on previously known information, but this is likely to be less than reliable.

The detail of the visual feedback delivered to a player in the SimpleSoccer environment is the same irrespective of the player's vision parameters – only the size of the viewing diamond changes, the amount of detail does not. Unlike the RoboCupSoccer environment, players are not able to sense objects that are close but not visible to the player – the only sense available to players in the SimpleSoccer environment is visual.

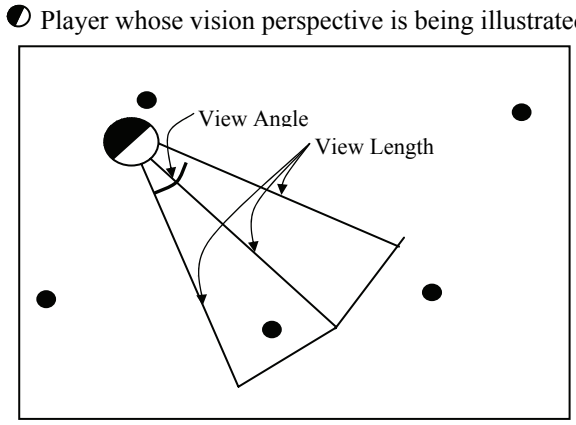


Fig. 2. The visible range of a player in the SimpleSoccer environment

2.2.1 Available Player Actions

The set of player actions provided by the SimpleSoccer simulator is a combination of some very basic, simple actions and some more complex hand-coded combinations of the basic actions. The complete set of actions available is listed in Table 1.

Action	Description
Turn	The player turns through the angle specified. *Argument:* direction.
Dash	The player dashes in the direction specified with the power specified. *Arguments:* direction, power, face.
Kick	If the ball is within a kickable distance from the player, the player kicks the ball in the direction specified with the power specified. *Arguments:* direction, power, face.
RunTowardGoal	If the direction to the player's goal is known, the player dashes once in that direction, otherwise no action is taken. *Argument:* power.
RunTowardBall	If the direction to the ball is known, the player dashes once in that direction, otherwise no action is taken. *Argument:* power.
GoToBall	If the direction to the ball is known, the player dashes towards the ball and continues to dash in that direction until the ball is within the kickable distance, otherwise no action is taken. *Argument:* power.
KickTowardGoal	If the direction to the player's goal is known, and the ball is within the kickable distance, the player kicks the ball once in the direction of its goal, otherwise no action is taken. *Argument:* power.
DribbleTowardGoal	If the direction to the player's goal is known, and the ball is within the kickable distance, the player kicks the ball once in the direction of its goal, then dashes once in the same direction. If the direction to the player's goal is not known, or the ball is not within the kickable distance, no action is taken. Argument: power.
Dribble	If the ball is within the kickable distance, the player kicks the ball once in the direction it is facing, then dashes once in the same direction. If the ball is not within the kickable distance, no action is taken. *Argument:* power.
DoNothing	The player takes no action.

Table 1. Available player actions

For each of the actions shown in Table 1:

- *direction* is specified in degrees in a clockwise direction relative to the direction the player is facing.
- *power* is specified as a percentage of maximum power and determines the number of cells the player or ball will travel as a result of the action.
- *face*, where specified, if true causes the player to turn to face in the direction specified after the completion of the action performed.

2.2.2 Player Default Action

If a player is unable to determine an action to be taken based on the information known, the player may, if so configured, perform a hand-coded default *hunt* action - on the basis that the most likely cause for a player not being able to determine an action is that the ball is not visible. The hand-coded hunt actions available as default actions are listed in Table 2.

Default Action	Description

Hunt Action 1
Goto Ball
	if the ball is not visible then
	dash in a randomly chosen direction
	else
	if ball is not in kickable distance then
	dash toward the ball
	else
	do nothing

Hunt Action 2
Locate Ball
	if the ball is not visible then
	dash in a randomly chosen direction
	else
	do nothing

Hunt Action 3 *Random Turn*	turn 90° in a randomly chosen direction

Table 2. Player default actions.

2.3 The Game

A SimpleSoccer game is played between two teams, each with a minimum of zero and a maximum of eleven players. There must be at least one player present on the field, and the team sizes may be unequal, thus allowing for single player or single team training. The *East* team starts the game on the right-hand (or east) side of the playing field (as viewed by the observer) and kicks towards the *East Goal* (Fig. 1). Similarly, the *West* team starts the game on the left-hand (or west) side of the playing field and kicks towards the *West Goal*. At the start of play the ball is placed at the centre of the field, and only the team *kicking-off* may enter the centre circle until contact is made with the ball.

There is no referee in the SimpleSoccer environment, thus there are no free kicks for offside or other rule violations. The ball is never out of bounds; the boundaries (except for the goal areas) are hard barriers which impede movement of the ball and players. There is no

concept of player momentum and stamina as implemented in the RoboCupSoccer environment.

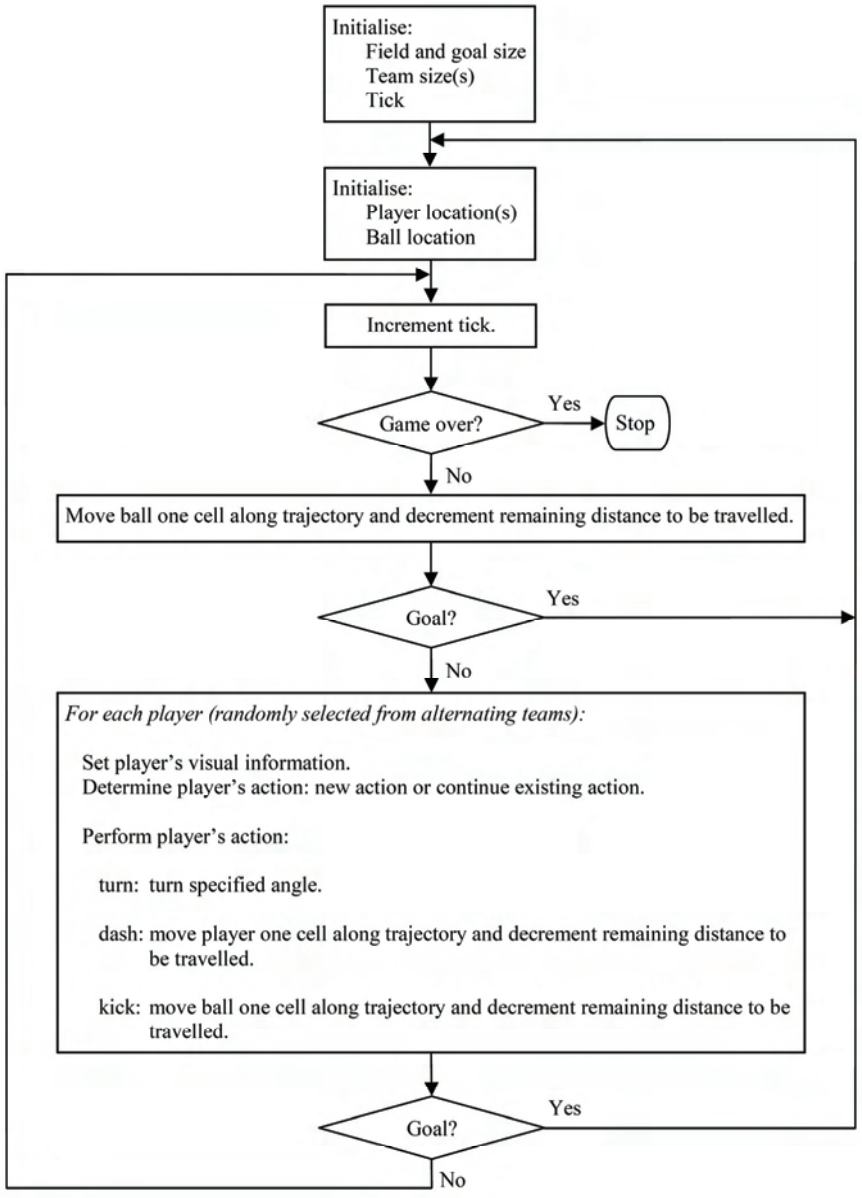


Fig. 3. SimpleSoccer program flow

When the ball is kicked, the distance it will travel is calculated (as a number of cells) and the ball will travel that distance at a constant speed and direction unless it is kicked again, or it encounters a barrier (the field boundary or a player) or reaches a goal. Similarly, when a player dashes, the distance the player will travel is calculated (as a number of cells) and the player will travel that distance at a constant speed and direction unless it initiates a new action or encounters a barrier (the field boundary, a goal or a player).
The SimpleSoccer environment provides players with a single sensor that detects visual information about the field, such as the distance and direction to objects in the player's current field of view – no other information is provided to the player. There is no coach, and there is no communication of any kind between players. In contrast to the RoboCupSoccer environment, no random "noise" is introduced to the visual sensor information provided to the player – thus the information provided is complete and certain, and there is no loss of clarity of vision over distance.
A SimpleSoccer unit of time is a single tick corresponding to one iteration of the program's main loop (Fig. 3). At each tick the ball and players are moved, if necessary, a single cell (as a result of a previous action) and each player is presented with their new (visual) view of the state of the game, whereupon each player determines what action, if any, is to be taken and that action is begun (any previous action still in progress is superseded by the new action).
After each goal scored the ball is replaced at the centre of the field and the players replaced to their side of the field, and the game continues. The game is terminated when one of the following conditions is met:

- the maximum game time, measured in ticks, expires.
- the target number of goals is scored by any team.
- a period of no player action, measured in ticks, occurs.

3. Evolving Goal-Scoring Behaviour

The usefulness of the SimpleSoccer simulator as a simplified model for the robot soccer environment is demonstrated by using the environment to train a simulated robot soccer player to exhibit goal-scoring behaviour.

3.1 Overview

A messy-coded genetic algorithm (Holland, 1975; Goldberg et al., 1989) is used to evolve a population of simulated robot soccer players, with the SimpleSoccer simulator being used to evaluate the players' ability. The behaviour of the players is governed by a *fuzzy inferencing system* (Zadeh, 1965; Jang et al., 1997) with the ruleset for the fuzzy inferencing system being evolved by the genetic algorithm.
Players being evolved are endowed with a configurable subset of soccer-playing skills taken from the full set of skills shown in Table 1. In addition, if a player is unable to determine an action to be taken based on the information known to it, the player will perform one of the hand-coded default actions listed in Table 2.
Players perform one of the available actions, or the configured default action, in response to external stimulus; the specific response being determined by the fuzzy ruleset and the fuzzy inferencing system. The external stimulus used as input to the fuzzy inference system is the

visual information supplied by the soccer simulator. The output of the fuzzy inference system is an (*action, value*) pair which defines the action to be taken by the player and the degree to which the action is to be taken. For example:

(KickTowardGoal, power)
(RunTowardBall, power)
(Turn, direction)

where *power* and *direction* are crisp values representing the defuzzified fuzzy set membership of the action to be taken. An example rule developed by the genetic algorithm is:

if Ball is Left and Goal is Left then Turn SlightlyLeft

The fuzzy inferencing system and messy-coded genetic algorithm are described briefly in the following sections, and in more detail in (Riley, 2005).

3.2 Player Architecture

The traditional decomposition for an intelligent control system is to break processing into a chain of information processing modules proceeding from sensing to action (Fig. 4).

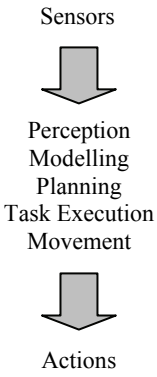


Fig. 4. Traditional control architecture

The control architecture implemented in this work is similar to Brooks' subsumption architecture (Brooks, 1985). This architecture implements a layering process where simple task achieving behaviours are added as required. Each layer is behaviour producing in its own right, although it may rely on the presence and operation of other layers. For example, in Fig. 5 the *Movement* layer does not explicitly need to avoid obstacles: the *Avoid Objects* layer, if present, will take care of that. This approach creates players with reactive architectures and with no central locus of control (Brooks, 1991).

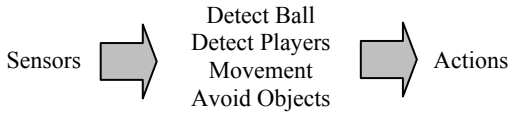


Fig. 5. Soccer player layered architecture

For the soccer player implemented for this work, the behaviour producing layers are implemented as fuzzy *if-then* rules and governed by *a fuzzy inference system* comprised of :

- the fuzzy rulebase.
- definitions of the membership functions of the fuzzy sets operated on by the rules in the rulebase.
- a reasoning mechanism to perform the inference procedure.

The fuzzy inference system is embedded in the player architecture, where it receives input from the soccer server and generates output necessary for the player to act (Fig. 6).

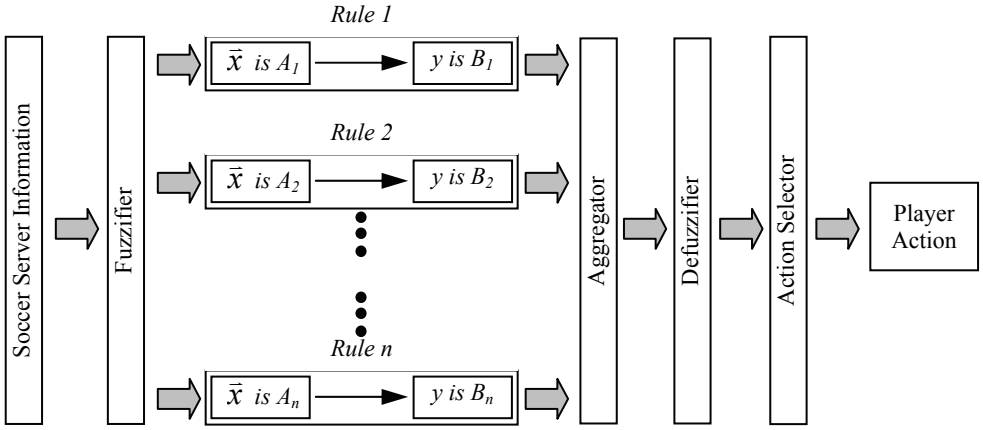


Fig. 6. Player architecture detail

3.2.1 Soccer Server Information

The application by the inferencing mechanism of the fuzzy rulebase to external stimuli provided by the soccer server results in one or more fuzzy rules being executed and some resultant action being taken by the player. The external stimuli used as input to the fuzzy inference system are a subset of the visual information supplied by the soccer server: only sufficient information to situate the player and locate the ball is used.

The SimpleSoccer server delivers only regular visual messages to the players: there are no aural or sense equivalents of the aural and sense messages delivered by the RoboCupSoccer

server in that environment. Information supplied by the SimpleSoccer server is complete, in so far as the objects actually in the player's field of vision are concerned, and certain. Players in the SimpleSoccer environment are aware at all times of their exact location on the field, but are only aware of the location of the ball and the goal if they are in the player's field of vision. The SimpleSoccer server provides the object name, distance and direction information for objects in a player's field of vision. The only state information kept by a player in the SimpleSoccer environment is the co-ordinates of its location and the direction in which it is facing.

3.2.2 Fuzzification

Input variables for the fuzzy rules are fuzzy interpretations of the visual stimuli supplied to the player by the soccer server: the information supplied by the soccer server is *fuzzified* to represent the degree of membership of one of three fuzzy sets: *direction*, *distance* and *power*; and then given as input to the fuzzy inference system. Output variables are the fuzzy actions to be taken by the player. The universe of discourse of both input and output variables are covered by fuzzy sets (*direction*, *distance* and *power*), the parameters of which are predefined and fixed. Each input is fuzzified to have a degree of membership in the fuzzy sets appropriate to the input variable.

The SimpleSoccer server provides crisp values for the information it delivers to the players. These crisp values must be transformed into linguistic terms in order to be used as input to the fuzzy inference system. This is the fuzzification step: the process of transforming crisp values into degrees of membership for linguistic terms of fuzzy sets. An example of input variable fuzzification is shown in Fig. 7. In this example the crisp input variable x has a degree of membership (μ) of both fuzzy sets A_1 (0.6) and A_2 (0.1).

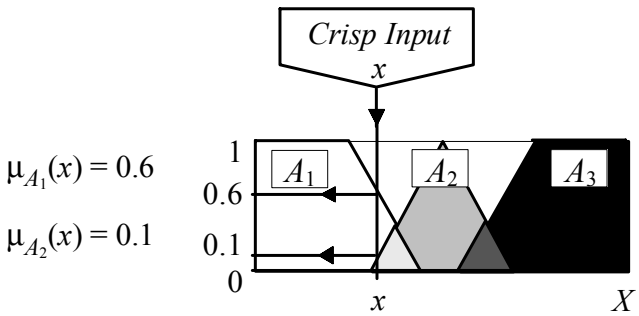


Fig. 7. Input variable fuzzification

The membership functions shown in Fig. 8 are used to associate crisp values with a degree of membership for the fuzzy sets *direction*, *distance* and *power*. The parameters for these fuzzy sets were not learned by the evolutionary process: they were fixed empirically. The

initial values were set having regard to SimpleSoccer parameters and variables, and fine-tuned after minimal experimentation in the SimpleSoccer environment.

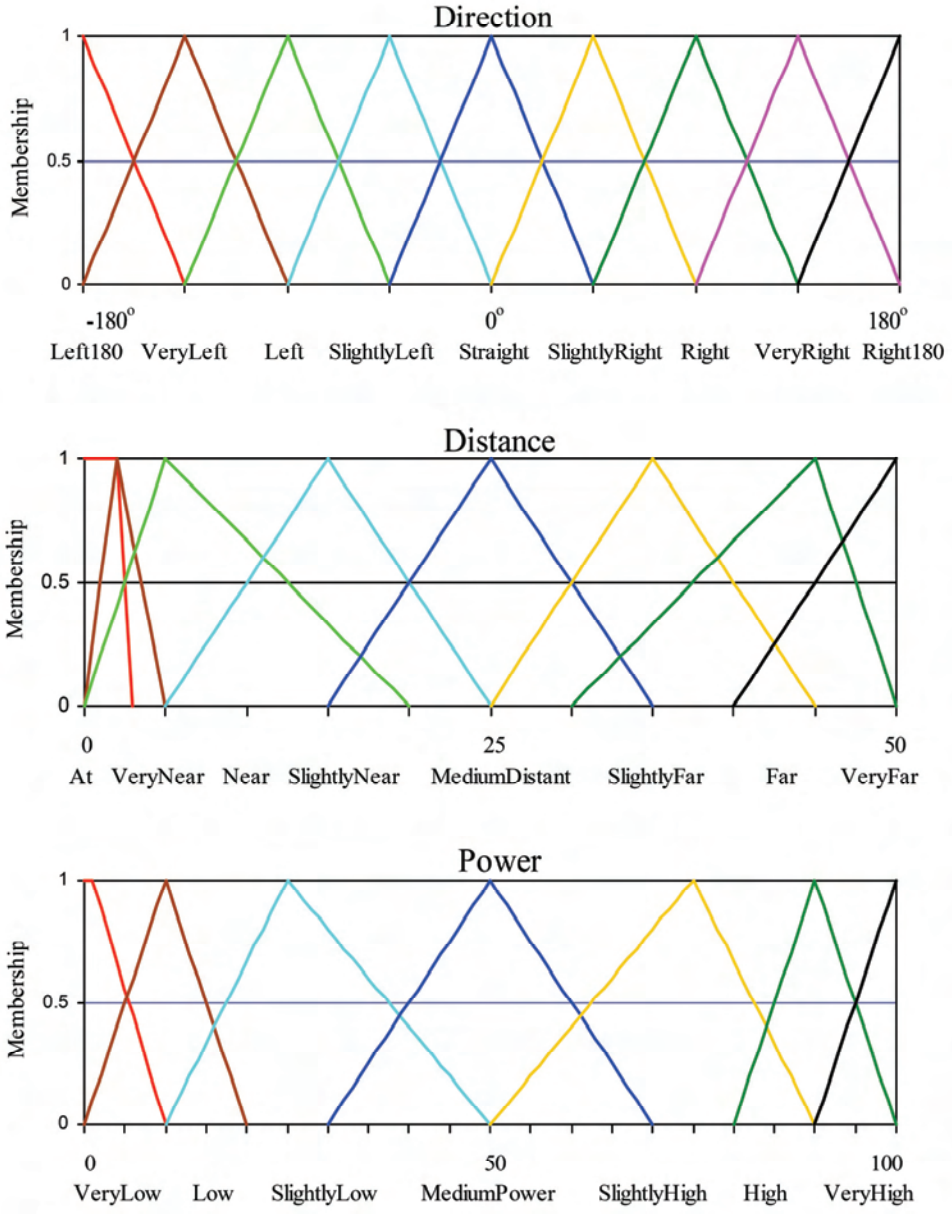


Fig. 8. Direction, distance and power fuzzy set membership

3.2.3 Implication and Aggregation

The core section of the fuzzy inference system is the part that combines the facts obtained from the fuzzification with the rule base and conducts the fuzzy reasoning process: this is where the fuzzy inferencing is performed.

After the input values are fuzzified they are applied to the antecedents of the fuzzy rules. For fuzzy rules with multiple antecedents, the fuzzy operators AND and OR are used as appropriate to obtain a single number that represents the result of the antecedent evaluation. This value is the degree to which the rule is true and is then applied to the consequent membership function. The evaluation of the antecedents is as follows:

- for the disjunction of rule antecedents, the fuzzy operator OR is defined by the fuzzy set operation union:

$$\mu_{A\cup B}(x) = \max[\mu_A(x), \mu_B(x)]$$

- for the conjunction of rule antecedents, the fuzzy operator AND is defined by the fuzzy set operation intersection:

$$\mu_{A\cap B}(x) = \min[\mu_A(x), \mu_B(x)]$$

The method implemented to correlate the result of the antecedent evaluation to the membership function of the consequent is the *correlation minimum*, or clipping method, where the consequent membership function is truncated at the level of the antecedent truth (Fig. 9).

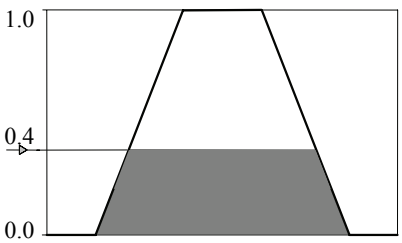


Fig. 9. Correlation minimum example

Aggregation is the process of combining the correlated fuzzy sets to produce a composite fuzzy region that represents the solution variable. The solution fuzzy region is then defuzzified if a crisp solution is required (as is the case in this work). The aggregation method used in this work is the *min/max* aggregation method. This method ORs the correlated consequent fuzzy set for each rule with the contents of the solution variable's output fuzzy region. This process takes the maximum of the consequent fuzzy set and the solution fuzzy set at each point along their mutual membership functions.

Fig. 10 is an illustration of a two-rule *Mamdani Fuzzy Inferencing System* (FIS) which implements the correlation minimum implication method and the min/max method of aggregation (Mamdami & Assilian, 1975).

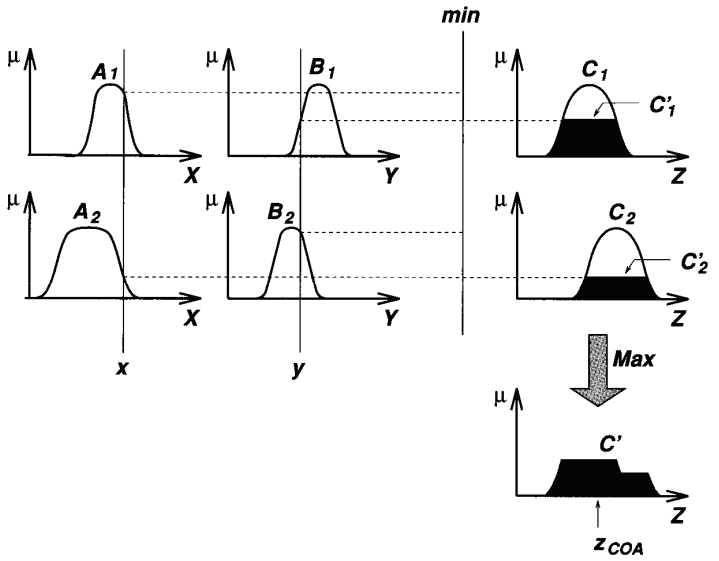


Fig. 10. 2-rule Mamdani FIS using Correlation Minimum implication and min/max aggregation. *Reproduced from (Jang et al., 1997)*

3.2.4 Defuzzification

The defuzzification method used is the *mean of maximum* method. This technique takes the output distribution and finds its mean of maxima in order to compute a single crisp number. This is calculated as follows:

$$z = \sum_{i=1}^{n} \frac{z_i}{n}$$

where z is the mean of maximum, z_i is the point at which the membership function is maximum, and n is the number of times the output distribution reaches the maximum level. An example outcome of this computation is shown in Fig. 11.

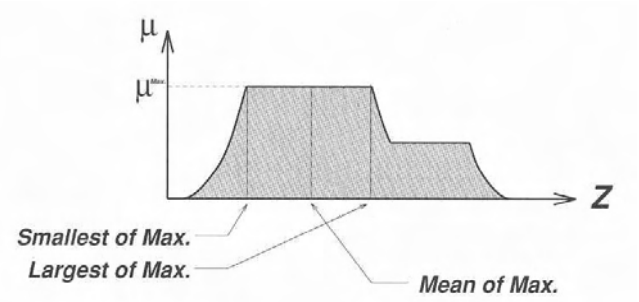


Fig. 11. Mean of Maximum defuzzification method. *Adapted from (Jang et al., 1997)*

3.2.5 Action Selection

Only one action is performed by the player in response to stimuli provided by the soccer server. Since several rules with different actions may fire, the action with the greatest level of support, as indicated by the value for truth of the antecedent, is selected.

3.3 Player Learning

This work employs an evolutionary technique in the form of a messy-coded genetic algorithm to evolve the rulebase that defines the behaviour of a robot soccer player. A genetic algorithm (GA) is an adaptive search technique which maintains a population of potential solutions that evolves over time in accordance with the rules of the genetic operators implemented by the algorithm. Each member of the population has its fitness as a solution to the problem evaluated against some known criteria, and members of the population are then selected for reproduction based upon that fitness, with a new generation of potential solutions being generated from the offspring of (typically) the most fit individuals. The process of evaluation, selection, reproduction, recombination and mutation is iterated until an acceptable solution is shown (Fig. 12).

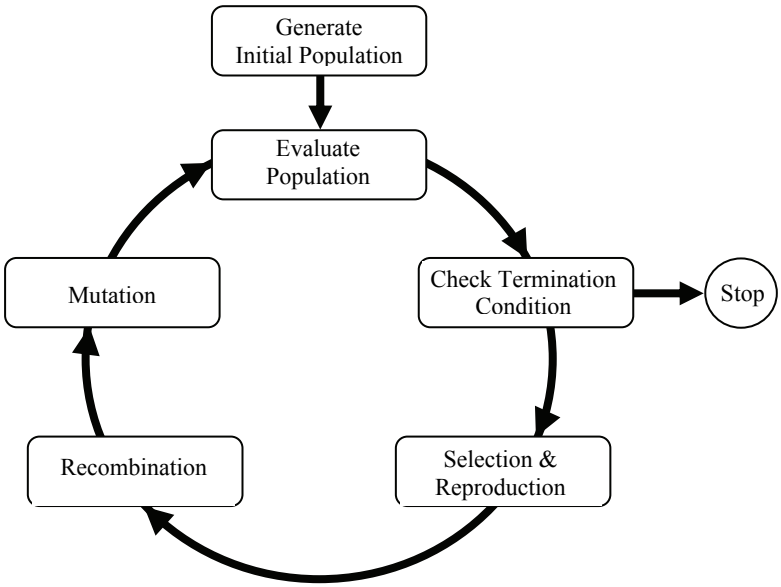


Fig. 12. The GA evolutionary cycle

The evaluation of the worth of an individual as a solution is achieved by the use of a fitness function. The objective of the fitness function is to numerically encode the performance of the individual with reference to the problem for which it is a potential solution. This is an extremely important part of the process, for without a fitness function which accurately evaluates the performance of potential solutions, the search will fail.

The flexibility provided by the messy-coded genetic algorithm is exploited in the definition and format of the genes on the chromosome, thus reducing the complexity of the rule encoding from the traditional genetic algorithm. Learning is achieved through testing and evaluation of the fuzzy rulebase generated by the genetic algorithm.
The fitness function used to determine the fitness of an individual rulebase takes into account the performance of the player based upon the number of goals scored, or attempts made to move toward goal-scoring, during a game.
The genetic algorithm implemented in this work is implemented using the Pittsburgh approach, where each individual in the population is a complete ruleset (Smith, 1980).

3.3.1 Representation of the Chromosome

For these experiments, a chromosome is represented as a variable length vector of genes, and rule clauses are coded on the chromosome as genes. The encoding scheme implemented exploits the capability of messy-coded genetic algorithms to encode information of variable structure and length. The mutation operator is analogous to the mutation operator for classic genetic algorithms, whereas the classic crossover operation is replaced by a *cut-and-splice* operation (Goldberg et al., 1989). It should be noted that while the encoding scheme implemented is a messy encoding, the algorithm implemented is the classic genetic algorithm: there are no primordial or juxtapositional phases implemented.
The basic element of the coding of the fuzzy rules is a tuple representing, in the case of a rule premise, a fuzzy clause and connector; and in the case of a rule consequent just the fuzzy consequent. The rule consequent gene is flagged to distinguish it from premise genes thus allowing multiple rules, or a ruleset, to be encoded onto a single chromosome.
For single-player trials, the only objects of interest to the player are the ball and the player's goal, and what is of interest is where those objects are in relation to the player. A premise is of the form:

(Object, Qualifier, {Distance | Direction}, Connector)

and is constructed from the following range of values:

Object : { BALL, GOAL }
Qualifier : { IS, IS NOT }
Distance : { AT, VERYNEAR, NEAR, SLIGHTLYNEAR, MEDIUMDISTANT, SLIGHTLYFAR, FAR, VERYFAR }
Direction : { LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180 }
Connector : { AND, OR }

Each rule consequent specifies and qualifies the action to be taken by the player as a consequent of that rule firing thus contributing to the set of (action, value) pairs output by the fuzzy inference system. A consequent is of the form:

(Action, {Direction | Null}, {Power | Null})

and is constructed from the following range of values (depending upon the skillset with which the player is endowed):

Action : { TURN, DASH, KICK, RUNTOWARDGOAL, RUNTOWARDBALL, GOTOBALL, KICKTOWARDGOAL, DRIBBLETOWARDGOAL, DRIBBLE, DONOTHING }

Direction : { LEFT180, VERYLEFT, LEFT, SLIGHTLYLEFT, STRAIGHT, SLIGHTLYRIGHT, RIGHT, VERYRIGHT, RIGHT180, TOWARDBALL, TOWARDBOAL}

Power : { VERYLOW, LOW, SLIGHTLYLOW, MEDIUMPOWER, SLIGHTLYHIGH, HIGH, VERYHIGH }

Fuzzy rules developed by the genetic algorithm are of the form:

if Ball is Near and Goal is Near then DribbleTowardGoal Low
if Ball is Far or Ball is SlightlyLeft then Run TowardBall High

In the example chromosome fragment shown in Fig. 13 the shaded clause has been specially coded to signify that it is a consequent gene, and the fragment decodes to the following rule:

if Ball is Left and Ball is At or Goal is not Far then Dribble Low

In this case the clause connector *OR* in the clause immediately prior to the consequent clause is not required, so ignored.

(Ball, is Left, And)	(Ball, is At, Or)	(Goal, is not Far, Or)	(Dribble, Null, Low)

Fig. 13. Messy-coded genetic algorithm example chromosome fragment

Chromosomes are not fixed length: the length of each chromosome in the population varies with the length of individual rules and the number of rules on the chromosome. The number of clauses in a rule and the number of rules in a ruleset is only limited by the maximum size of a chromosome, which for this work was 64 genes. The minimum size of a rule is two clauses (one premise and one consequent), and the minimum number of rules in a ruleset is one. Since the cut-and-splice and mutation operations implemented guarantee no out-of-bounds data in the resultant chromosomes, a rule is only considered invalid if it contains no premises. Any invalid rules are ignored when the ruleset is applied. A complete ruleset is considered invalid only if it contains no valid rules.
An example complete chromosome and corresponding rules are shown in Fig. 14 (with appropriate abbreviations). Some advantages of using a messy encoding in this case are:

- a ruleset is not limited to a fixed size
- a ruleset can be over specified (clauses may be duplicated)
- a ruleset can be under specified (not all genes are required to be represented)
- clauses may be arranged in any way

(B,N,O)	(B,nF,A)	(G,N,A)	(RB,-,L)	(B,A,A)	(G,vN,O)	(KG,-,M)	(B,L,A)	(T,L,-)

Premise	Consequent

Rule 1: *if Ball is Near or Ball is not Far and Goal is Near then RunTowardBall Low*
Rule 2: *if Ball is At and Goal is VeryNear then KickTowardGoal MediumPower*
Rule 3: *if Ball is Left then Turn Left*

Fig. 14. Example chromosome and corresponding rules

In contrast to classic genetic algorithms which use a fixed size chromosome and require "*don't care*" values in order to generalise, no explicit "*don't care*" values are, or need be, implemented for any attributes in this method. Since messy-coded genetic algorithms encode information of variable structure and length, not all attributes, particularly premise variables, need be present in any rule or indeed in the entire ruleset. A feature of the messy-coded genetic algorithm is that the format implies "*don't care*" values for all attributes since any premise may be omitted from any or all rules, so generalisation is an implicit feature of this method.

3.3.2 Selection and Reproduction

Selection and reproduction are important processes for evolutionary algorithms. Individuals from the population are selected according to some criteria to be reproduced for the next generation. GA reproduction is essentially a cloning operation in which the individuals selected for reproduction are copied, and it is during the recombination process that the copies are mated to form new individuals. For genetic algorithms, selection and reproduction alone cannot introduce new individuals into the population: that is achieved throug the genetically-inspired *recombination* operators of crossover (*cut-and-splice* in the case of messy-coded GAs) and mutation. The purpose of selection and reproduction is to favour fitter individuals on the basis that the fitter an individual the more likely it will produce even more fit offspring.

A fitness-proportionate method of selection (Holland, 1975; Goldberg, 1989) known as "roulette wheel" selection was implemented for this work. With this method the number of times an individual is expected to be selected to reproduce is the ratio of the individual's fitness to the average fitness of the population. The implementation can be likened to a biased roulette wheel, where each individual in the current population has a slot on the roulette wheel proportional to that individual's fitness. The roulette wheel is spun once for each parent required, with the winning individuals being paired for reproduction.

3.3.3 Cut-and-Splice for Variable Length Chromosomes

Since the messy-coding implemented allows chromosomes of different lengths the crossover operation of the classic genetic algorithm needs to be modified. For messy-coded genetic algorithms the crossover operation is considered in its two distinct steps: the *cut* operation and the *splice* operation. The cut operator cuts each chromosome at a randomly chosen

position, and since the chromosomes may be of different lengths, the resultant fragments may also be of different lengths. The splice operator concatenates the fragments produced by the cut operator, resulting in two new chromosomes of possibly different lengths from the original chromosomes. The cut-and-splice operation implemented in this work guarantees the operations will not result in out-of-bounds data in the resultant chromosomes. Fig. 15 is an example of the cut-and-splice operation for messy-coded chromosomes.

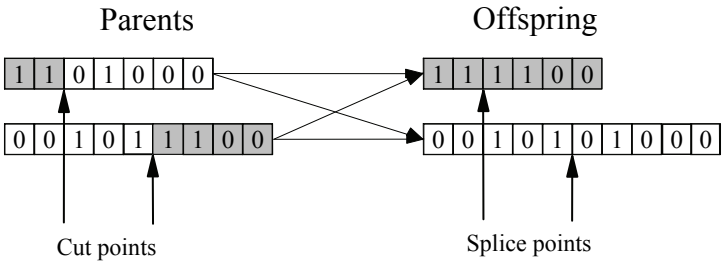


Fig. 15. Example cut-and-splice operation

3.3.4 Mutation

Mutation, which helps to maintain diversity in the population, is the arbitrary modification of individuals. The mutation scheme implemented in this work is a variation of random *single-bit* mutation, but in this case it is random *single-allele* mutation since the genes encoded in this work are integer values rather than single bits. This is a method in which a single allele is chosen randomly for modification to a random value. The mutation operator implemented guarantees mutations will not result in out-of-bounds data in the resultant chromosome.

3.4 Experimental Evaluation

A series of 20 trials was performed in order to test the viability of the fuzzy inferencing system for the control of the player, and the genetic algorithm to evolve the fuzzy ruleset. The following sections describe the trials performed, the parameter settings for each of the trials and other fundamental properties necessary for conducting the trials.

3.4.1 Fitness Evaluation

The objective of the fitness function for the genetic algorithm is to reward the fitter individuals with a higher probability of producing offspring, with the expectation that combining the fittest individuals of one generation will produce even fitter individuals in later generations. The fitness function used in these trials rewarded individuals for, in order of importance:

- the number of goals scored in a game

- minimising the distance of the ball from the goal

This combination was chosen to reward players primarily for goals scored, while players that do not score goals are rewarded on the basis of how close they are able to move the ball to the goal, on the assumption that a player which kicks the ball close to the goal is more likely to produce offspring capable of scoring goals. This decomposes the problem of evolving goal-scoring behaviour into the two less difficult problems:

- evolve ball-kicking behaviour that minimises the distance between the ball and goal, and
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

The actual fitness function implemented was:

$$f = \begin{cases} \begin{cases} 1.0 & ,kicks = 0 \\ 0.5 + \dfrac{dist}{2.0 \times fieldLen} & ,kicks > 0 \end{cases} & ,goals = 0 \\ \dfrac{1.0}{2.0 \times goals} & ,goals > 0 \end{cases} \quad (1)$$

where

goals = the number of goals scored by the player during the trial
kicks = the number of times the player kicked the ball during the trial
dist = the minimum distance of the ball to the goal during the trial
fieldLen = the length of the field

Note that this fitness function indicates better fitness as a lower number, in effect representing the optimisation of fitness as a minimisation problem.

3.4.2 GA Control Parameters

The genetic algorithm parameters used in all 20 trials are shown in Table 3.

Parameter	Value
Maximum Chromosome Length	64 genes
Population Size	200
Maximum Generations	25
Selection Method	Roulette Wheel
Crossover Method	Single point cut-and-splice
Crossover Probability	0.8
Mutation Rate	10%
Mutation Probability	0.35

Table 3. Genetic algorithm control parameters

3.4.3 Simulator Control Parameters

The SimpleSoccer simulator parameters used in all 20 trials are shown in Table 4.

Parameter	Value
Field Length	61 cells
Field Width	31 cells
Goal Width	7 cells
Kickable Distance	1.0 cells
View Angle	90 degrees
View Length	5 cells
Maximum DASH distance	7.5 cells
Maximum KICK distance	15 cells
Player Skillset	All skills listed in Table 1
Default action	Hunt action 3: *Random turn*

Table 4. SimpleSoccer control parameters

3.4.3 Trial Results

For the results reported, each trial consisted of one complete execution of the genetic algorithm during which multiple simulated games of soccer were played, with the only player on the field being the player under evaluation.

For each game, the player was placed at a randomly selected position on its half of the field and oriented so that it was facing the end of the field to which it was kicking, and the ball was placed at a randomly selected position along the centre line of the field. A game was terminated when one of the following conditions was met:

- the maximum game time of 1000 ticks expired.
- the target of 10 goals was scored, reflecting a fitness value of 0.05. This figure was chosen to allow the player a realistic amount of time to develop useful strategies yet terminate the search upon finding an acceptably good individual.
- a 100 tick period of no player action occured.

A randomly generated population of players was generated and evolved over time by the genetic algorithm, with the evaluation of each member of the population being performed in the SimpleSoccer environment. The evolutionary search was stopped:

- after a specified maximum number of generations, or

- when the specified target fitness was reached by any player.

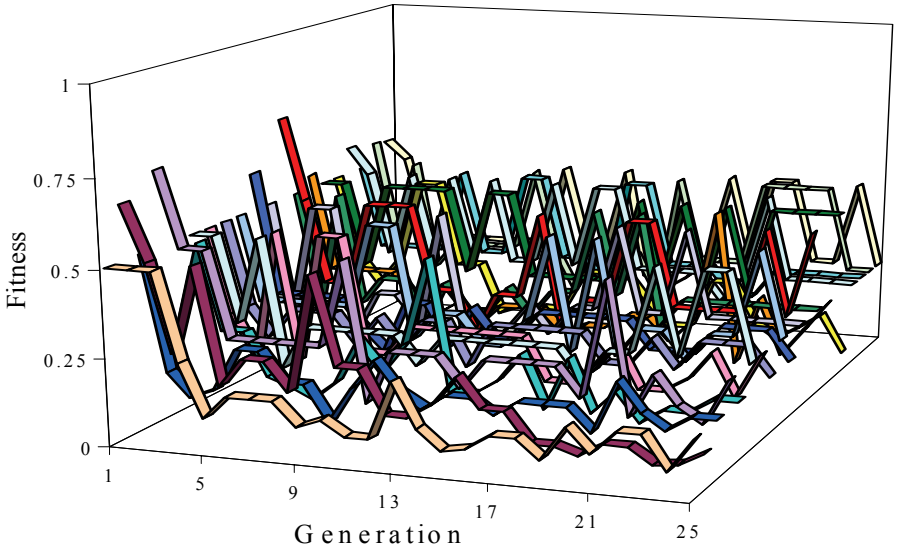


Fig. 16. SimpleSoccer: Best individual fitness

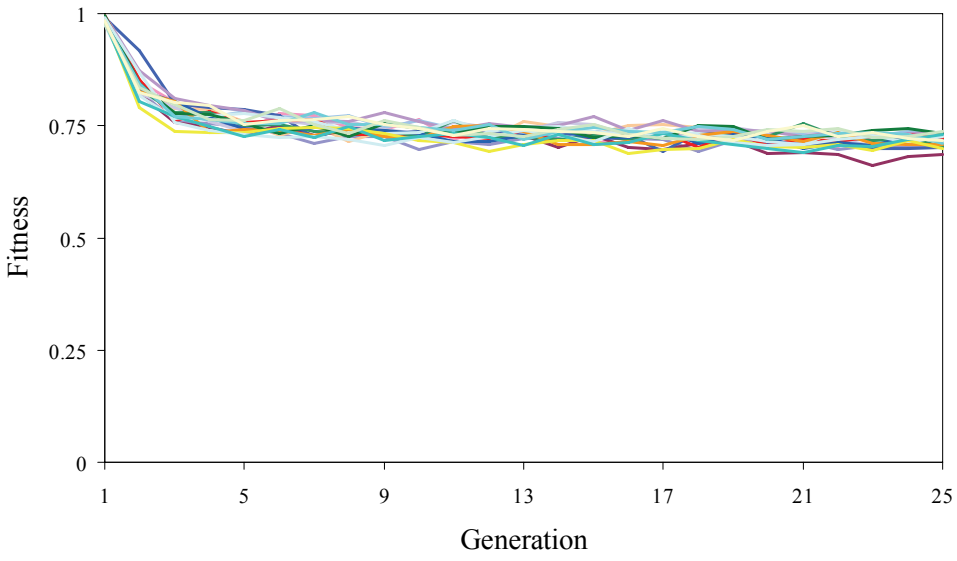


Fig. 17. SimpleSoccer: Population average fitness

Fig. 16 shows the best individual fitness from the population after each generation for each of the 20 trials. This graph shows that individuals able to score goals were found after very few generations, with some individuals being capable of scoring multiple goals in the allotted time.

Fig. 17 shows the average fitness of the population after each generation for each of the 20 trials. This graph shows that the average performance of the population improves steadily and plateaus, but while individual players do score goals, the population does not approach an average fitness of 0.5, or goal-scoring behaviour.

These results show that the method presented is capable of training a simulated robot soccer player to develop goal-scoring behaviour. The method uses a genetic algorithm to evolve the fuzzy rulesets that drive the soccer player's behaviour, with the evolutionary process being allowed to run for a maximum of only 25 generations which, while sufficient to demonstrate the effectiveness of the method, is probably not sufficient to evolve players with robust, consistent goal-scoring behaviour.

3.4.4 SimpleSoccer as a Model for RoboCupSoccer

To gauge the effectiveness of the SimpleSoccer environment as a model for RoboCupSoccer a further series of 20 trials was performed in the RoboCupSoccer environment. Similar simulator and GA control parameters were used. Game times for the RoboCupSoccer environment were limited to 120 seconds (real time) rather than a number of program ticks. The results of these trials are shown below.

Fig. 18 shows the best individual fitness from the population after each generation for each of the 20 trials. It is evident from a comparison of Fig. 16 and Fig. 18 that while good individuals are found quickly in both environments, the algorithm seems to produce more consistent behaviour in the RoboCupSoccer environment. These data show that once a good individual is found in the RoboCupSoccer environment, good individuals are then more consistently found in future generations than in the SimpleSoccer environment.

Fig. 19 shows the average fitness of the population after each generation for each of the 20 trials. This graph shows that the performance of the population does improve steadily and, in some of the trials, plateaus towards a fitness of 0.5, or goal-scoring behaviour. Fig. 19 also shows that the average fitness curves for the RoboCupSoccer trials are less tightly clustered than those of the SimpleSoccer trials (see Fig. 17), probably reflecting the more stochastic nature of the environment.

While the difference in the results of the experiments in the RoboCupSoccer and SimpleSoccer environments indicate that SimpleSoccer is not an exact model of RoboCupSoccer, as indeed it is not intended to be, there is sufficient similarity in the results to indicate that the SimpleSoccer environment is a good simplified model of the RoboCupSoccer environment.

Much of the motivation for creating the SimpleSoccer environment was the prohibitive time to train players in the real-time RoboCupSoccer environment and the need to reduce that training time to improve the efficiency and effectiveness of machine learning methods for training simulated robot soccer players. Table 5 shows the average number of seconds of real time for a single fitness evaluation in each of the environments used to evolve players for robot soccer, and from the data shown in Table 5 it is evident that the goal of creating a more efficient environment for machine learning techniques has been achieved.

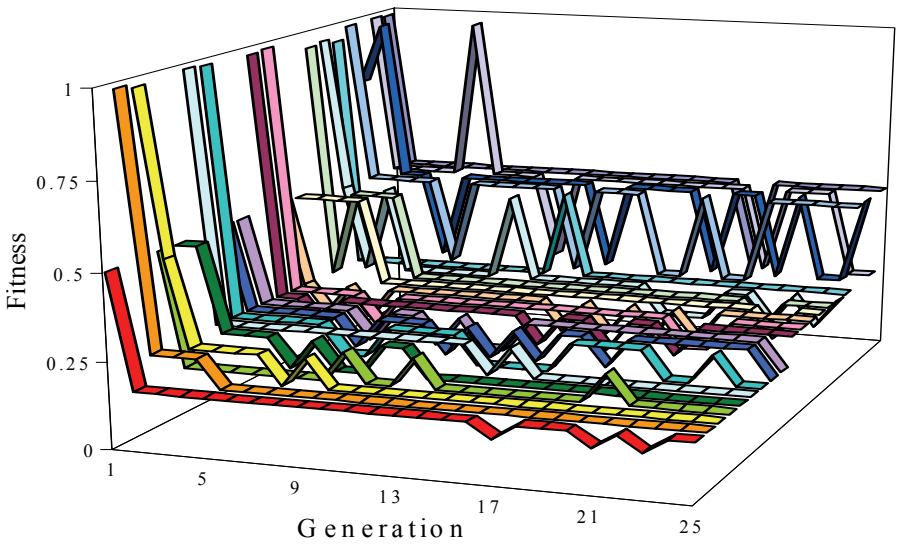


Fig. 18. RoboCuSoccer: Best individual fitness

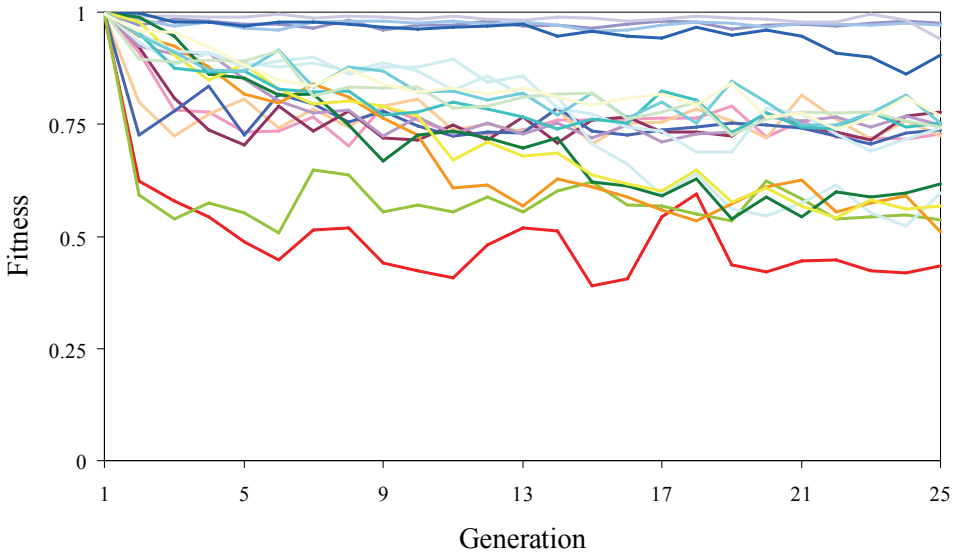


Fig. 19. RoboCupSoccer: Population average fitness

The RoboCupSoccer simulator used throughout this work was version 7.10, compiled and executed on an HP 9000/777 workstation running version 11.0 of the HP-UX operating system. The SimpleSoccer simulator was originally developed on an HP9000/777 workstation running HP-UX version 11.0, and was later ported to an Intel Pentium-based PC running Windows XP. Evaluation times are quoted for each of those systems. No trials using the RoboCupSoccer simulator were performed on the PC. Note that although the SimpleSoccer evaluation time is two orders of magnitude smaller on the PC, RoboCupSoccer evaluation times would not enjoy the same improvement if executed on the PC – the RoboCupSoccer evaluation times are constrained by the real-time nature of the simulator, and the training game times were 60 seconds. Any benefit from running the RoboCupSoccer simulations on faster hardware would be evident in the few seconds of overhead time only, and would not significantly reduce the evaluation time.

Simulator	Platform	Seconds/Evaluation
RoboCupSoccer	HP 9000/777 workstation. 120MHz PA-7200 CPU, 256MB RAM, HP-UX 11.0 Operating System	70.65
SimpleSoccer	HP 9000/777 workstation. 120MHz PA-7200 CPU, 256MB RAM, HP-UX 11.0 Operating System	10.20
SimpleSoccer	Compaq PC. 1.6GHz Pentium M CPU, 512MB RAM, Windows XP Operating System	0.112

Table 5. Evaluation times

4. Summary and Discussion

The goal of this work was to create an environment with similar complexity and dynamics to the RoboCupSoccer environment, but with reduced uncertainty, both in player perception and in the player's interaction with the environment. The motivation was to create an environment in which the training times of machine learning techniques would be reduced sufficiently so as to improve the viability of such techniques, and to show that players could be trained in this environment to display reasonable goal-scoring behaviour. The SimpleSoccer environment was developed for this purpose, and through some sample experiments it was shown that the SimpleSoccer environment does aid in the reduction of training times for some machine learning techniques.

The implementation of a messy-coded genetic algorithm which successfully evolves the ruleset for a fuzzy logic-based simulated robot soccer player was described. Several trials were performed to test the capacity of the method to produce goal-scoring behaviour. The results of the trials performed indicate that the player defined by the evolved fuzzy rules of the controller is capable of displaying consistent goal-scoring behaviour.

Furthermore, tests in which the initial population for RoboCupSoccer was seeded with players evolved in the SimpleSoccer environment suggest that there is significant benefit in using the SimpleSoccer environment as an heuristic to generate high quality initial solutions for the RoboCupSoccer environment (Riley, 2003; Riley, 2005). The evolution of players displaying reasonable goal-scoring behaviour is achievable in the SimpleSoccer

environment in a fraction of the time it would take in the RoboCupSoccer environment, and only a few generations are required in RoboCupSoccer to refine the behaviours evolved in the SimpleSoccer environment. High-level strategies learned in the more certain SimpleSoccer environment are directly transferrable to the RoboCup environment, and when used as the starting point for further learning can help to reduce the training time in the RoboCup environment.

5. References

Bajurnow, A. & Ciesielski, V. (2004). Layered Learning for Evolving Goal Scoring Behaviour in Soccer Players, *Proceedings of the 2004 Congress on Evolutionary Computation,* Vol. 2, pp. 1828-1835, Portland OR, June 2004, G. Greenwood (Ed.), IEEE, Piscataway NJ.

Balch, T. (1995). The Ascii Robot Soccer Home Page. *http://www.cs.cmu.edu/~trb/soccer/*, 1995.

Brooks, R. (1985). Robust Layered Control System for a Mobile Robot, *A.I. Memo 864*, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge MA.

Brooks, R. (1991). Intelligence Without Representation. *Artificial Intelligence, vol.* 47, 1991, pp. 139-159.

Ciesielski, V. & Lai, S.Y. (2001). Developing a Dribble-and-Score Behaviour for Robot Soccer using Neuro Evolution, *Proceedings of the Fifth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, pp. 70-78, Dunedin, New Zealand, November 2001, Wigham, P.; Richards, K.; McKay, B.; Gen, M.; Tujimura, Y. & Namatame, A. (Eds.).

Ciesielski, V., Mawhinney, D. & Wilson, P. (2001). Genetic Programming for Robot Soccer, *Proceedings of the RoboCup 2001 Symposium, Lecture Notes in Artificial Intelligence*, pp. 319-324, Seattle WA, August 2001, Birk, A.; Coradeschi, S. & Tadokoro, S. (Eds.), Springer NY.

Ciesielski, V. & Wilson, P. (1999). Developing a Team of Soccer Playing Robots by Genetic Programming, *Proceedings of the Third Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, pp. 101-108, Canberra, Australia, November 1999.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, ISBN 0-201-15767-5.

Goldberg, D., Korb, B. & Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems, vol. 3*, 1989, pp. 493-530.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor: MI.

Jang, J.-S.; Sun, C.-T. & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*, Prentice Hall, ISBN 0-13-261066-3, Upper Saddle River NJ.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa, E. (1995). RoboCup: The Robot World Cup Initiative, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, *Workshop on Entertainment and AI/ALife*, Montréal, Québec, Canada, August 1995, Morgan Kaufmann, San Francisco CA.

Kitano, H.; Tambe, M.; Stone, P.; Veloso, M.; Coradeschi, S.; Osawa, E.; Matsubara, H.; Noda, I. & Asada, M. (1997). The RoboCup Synthetic Agent Challenge 97, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pp. 24-29, Nagoya, Japan, August 1997, Morgan Kaufmann, San Francisco CA.

Lima, P.; Custódio, L.; Akin, L.; Jacoff, A.; Kraezschmar, G.; Ng, B.K.; Obst, O.; Röfer, T.; Takahashi, Y. & Zhou, C. (2005). RoboCup 2004 Competitions and Symposium: A Small Kick for Robots, a Giant Score for Science. *AI Magazine* 26(2), 2005, pp. 36-61.

Luke, S. (1998a). Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Madison WI, July 1998, Koza, J.R.; Banzhaf, W.; Chellapilla, K.; Deb, K.; Dorigo, M.; Fogel, D.B.; Garzon, M.H.; Goldberg, D.E.; Iba, H. & Riolo, R.L. (Eds.), Morgan Kaufmann, San Francisco CA.

Luke, S. (1998b). Evolving SoccerBots: A Retrospective, Proceedings of the Twelfth Annual Conference of the Japanese Society for Artificial Intelligence, Tokyo, Japan, June 1998.

Mamdani, E. & Assilian, S. (1975). An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller. *International Journal of Man-Machine Studies*, vol. 7(1), 1975, pp. 1-13.

Riedmiller, M.; Merke, A.; Meier, D.; Hoffman, A.; Sinner, A.; Thate, O. & Ehrmann, R. (2001). Karlsruhe Brainstormers – a Reinforcement Learning Approach to Robotic Soccer. In: *RoboCup-2000: Robot Soccer World Cup IV*, Stone, P.; Balch, T. & Kraetszchmar, G. (Eds.), Springer Verlag, Berlin.

Riedmiller, M.; Gabel, T.; Knabe, J. & Strasdat, H. (2005). Brainstormers 2D - Team Description 2005. In: *RoboCup 2005: Robot Soccer World Cup IX.*, Bredenfeld, A.; Jacoff, A.; Noda, I. & Takahashi, Y. (Eds.), Springer Verlag, Berlin.

Riley, J. (2003). The SimpleSoccer Machine Learning Environment, *Proceedings of the First Asia-Pacific Workshop on Genetic Programming*, pp. 24-30, Canberra, Australia, December 2003, Cho, S-B.; Nguen, H.X. & Shan, Y. (Eds.).

Riley, J. & Ciesielski, V. (2004). Evolution of fuzzy rule based controllers for dynamic environments, In: *Recent Advances in Simulated Evolution and Learning, volume 2 of Advances in Natural Computation*, Tan, K.C.; Lim, M.H; Yao, X. & Wang, L. (Eds.), chapter 23, pp. 426-445. World Scientific, ISBN 981-238-952-0, Singapore.

Riley, J. (2005). Evolving Fuzzy Rules for Goal-Scoring Behaviour in a Robot Soccer Environment, *PhD Thesis*, RMIT University: Melbourne, Australia.

Smith, S. (1980). A Learning System Based on Genetic Adaptive Algorithms, *PhD Thesis,* Department of Computer Science, University of Pittsburgh: Pittsburgh PA.

Stone, P. & Sutton, R. (2001). Scaling Reinforcement Learning Toward RoboCup Soccer, *Proceedings of the Eighteenth International Conference on Machine Learning*, Williamstown MA, July 2001, Brodley, C.E. & Danyluk, A.P. (Eds.), Morgan Kaufmann, San Francisco CA.

Stone, P. & Veloso, M. (1999). Team-partitioned, Opaque-transition Reinforcement Learning, *Proceedings of the Third International Conference on Autonomous Agents*, pp. 206-212, ISBN 1-58113-066-X, Seattle WA, May 1999, Etzioni, O.; Müller, J. & Bradshaw, J. (Eds.), ACM Press, NY.

Uchibe, E. (1999). Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots, *PhD thesis*, Osaka University: Osaka, Japan.

Zadeh, L. (1965). Fuzzy Sets. *Journal of Information and Control*, vol. 8, 1965, pp. 338-353.

16

Analysing the Difficulty of Learning Goal-Scoring Behaviour for Robot Soccer

Jeff Riley and Vic Ciesielski
RMIT University
Australia

1. Introduction

This work describes a method of analysing fitness landscapes and uses the method to analyse the difficulty of learning goal-scoring behaviour for robot soccer – a problem that is considered to be very difficult for evolutionary algorithms. Learning goal-scoring behaviour can be made easier or harder by varying the amount of expert knowledge provided to the evolutionary process. Expert knowledge can be varied by changing the innate player behaviours available, providing an *a priori* problem decomposition (that is, breaking the problem into a series of smaller, easier problems) or by providing a composite fitness function that effectively guides the search.

The concept of fitness landscapes, and the idea that the process of evolution could be studied by visualizing the distribution of fitness values across the population as a landscape, has been long-established in the field of evolutionary biology, having been first proposed by Sewell Wright (Wright, 1932). Later the landscape analogy was revived with the development of formal methods to handle optimization problems in complex physical systems (Frauenfelder et al., 1997). A major area of concern with fitness landscapes is that there is no generally accepted definition of what constitutes a fitness landscape. There is not much agreement in the field as to what a fitness landscape is and how it should be arranged - whether a neighbourhood relation is required to describe it, and much less agreement as to what the neighbourhood relation should be. This work addresses these shortcomings by describing a simple, "black-box" neigbourhood relation that defines the fitness landscape generated by an evolutionary search. The efficacy of the method is shown by applying an evolutionary technique to a difficult search problem (learning goal-scoring behaviour), and using *autocorrelation* and *information content* landscape measures to analyse features of the resultant fitness landscape to explain how the difficulty of the problem is changed by injecting human expertise. The analysis reveals that when only basic skills are available to the player the fitness landscape is very flat and contains only a few thin peaks. As more human expertise is injected, more gradient information becomes apparent on the landscape and genetic search is more successful.

Evolving soccer-playing skills for robot soccer players is a well-known difficult problem for evolutionary algorithms. A wide variety of approaches and technologies have been used in attempts to construct good robot soccer players. These include hand-coding, genetic

algorithms, genetic programming, reinforcement learning, neural networks, behaviour-based and deliberative agents, and various combinations of those. In the early years of the RoboCupSoccer competition (Kitano et al., 1995) a few researchers attempted to fine tune, or learn incrementally, some low-level skills using AI machine learning techniques (Stone, 1998; Stone & Sutton, 2001), but nearly all entrants used hand-coded skills and strategies (Luke, 1998). Even today hand-coded players, or players with hand-coded skills, generally continue to outplay players whose skills have been entirely learned or developed automatically (Lima et al., 2005). For example, the 2005 RoboCupSoccer 2D simulation league winner used hand-coded strategies which employed a mixture of hand-coded skills and skills developed using machine learning techniques (Riedmiller et al., 2005). There has been only limited success when applying standard machine learning techniques to this problem. Much of the work to date has been characterised by researchers beginning their work with high expectations, then ratcheting down their expectations as the work progresses, and finally adjusting their goals (and the soccer playing behaviours and skills of the players being developed) to align with the progress being made.

The size of the robot soccer search space and the paucity of solutions that lead to good soccer-playing behaviour tend to suggest that, in the extreme case, the problem resembles a *needle-in-a-haystack* problem (Jones & Forrest, 1995a; Langdon & Poli, 1998a; Right et al., 2002), indicating a possible cause for the difficulty of the problem for evolutionary algorithms. Previous work in the area of evolutionary learning for robot soccer has focussed on the learning, and what parameters and controls need to be manipulated in order to produce acceptable soccer-playing behaviour, but there has been no systematic investigation of the difficulty of the problem. Understanding why problems are difficult for evolutionary algorithms is a critical step not only in solving the particular problem under investigation, but also in advancing the field and improving the potential usefulness of evolutionary algorithms. The goal of this work is to describe a method for analysis of the fitness landscape generated by the problem of learning goal-scoring behaviour for robot soccer, and to use the analysis to better understand the difficulty of the problem and how progress might be made.

2. Evolving Goal-Scoring Behaviour for Robot Soccer

For this work a messy-coded genetic algorithm (Holland, 1975; Goldberg et al., 1989) is used to evolve a single robot soccer player in the SimpleSoccer simulated soccer environment (Riley, 2003). The behaviour of the player is governed by a fuzzy inferencing system (Zadeh, 1965; Jang et al., 1997) with the ruleset for the fuzzy inferencing system being evolved by the genetic algorithm.

The player being evolved is endowed with a configurable subset of soccer-playing skills taken from a rich set of basic and hand-coded skills and default actions. The player executes one of the skills or performs the default action available to it in response to some external stimulus; the specific response being determined by the fuzzy ruleset and the fuzzy inferencing system. The external stimulus used as input to the fuzzy inference system is the visual information supplied by the soccer simulator.

A full description of the method used to evolve the soccer player is given in the previous chapter.

3. Fitness Landscapes

Wright's original diagram, reproduced in Fig. 1, is a two-dimensional depiction of the relative "adaptiveness" of all possible combinations of allelic states for genes on a particular chromosome. The diagram is effectively a contour map with the contour lines representing levels of adaptiveness. This original diagram included no labels for the axes, and no indication was given as to how the various gene combinations should be arranged on the landscape - no notion of "neighbourhood" was defined or discussed by Wright. In landscape terms, the neighbourhood relation defines which points, or individuals, are arranged as immediate neighbours on the landscape, and so is extremely important in defining the landscape.

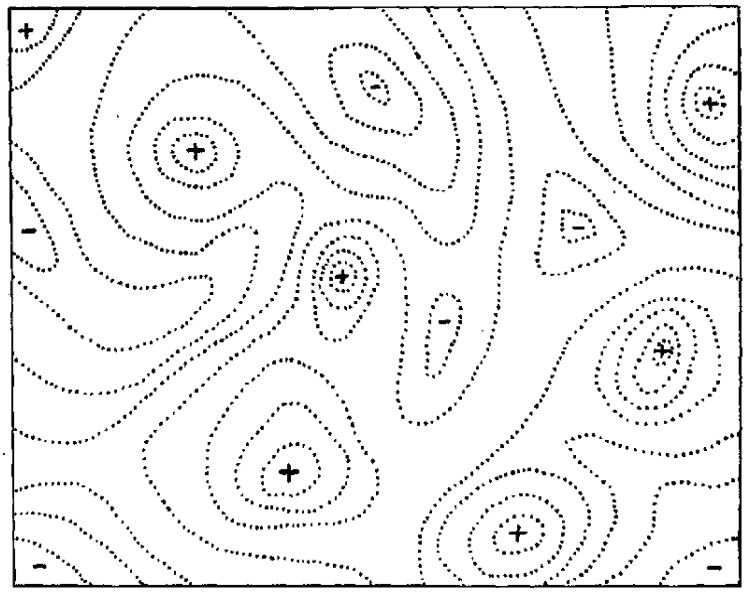


Fig. 1. Diagrammatic representation of the field of gene combinations in two dimensions instead of many thousands. Dotted lines represent contours with respect to adaptiveness. *Reproduced from (Wright, 1932)*

Much of the work involving fitness landscapes avoids a rigorous definition of the landscape under analysis (Jones, 1995a), and where it is mentioned or implied at all the landscape is usually assumed to be the single-bit mutation landscape: the landscape generated by arranging all single-bit mutations of a chromosome represented as a string of binary digits such that chromosomes that differ by only a single bit are neighbours, then plotting the fitness of each chromosome on a separate axis. On such landscapes, genetic operators such as crossover are assumed to take hypersteps over the fitness landscape described by mutation. There have been attempts to overcome this deficiency, and following is a description of some of major recent work in the field.

The NK fitness landscape model (Kauffman, 1989; Kauffman & Levin, 1987) was proposed as a means to study how epistasis - the dependency of fitness upon the interaction of the allelic state of multiple genes (Lush, 1935) - affects the ruggedness of the fitness landscape. An NK fitness landscape is defined by a fitness function which can be tuned in order to alter the ruggedness of the resultant fitness landscape. The fitness function is defined by two parameters: the number of genes (N), and the number of epistatic links, or interactions, between genes (K). In the most common implementations each gene has two possible alleles, thus the genotype can be represented by a bit string of length N. Each gene in a chromosome contributes to the fitness of the chromosome based on K+1 values: its own and those of the K genes to which it is linked. The three-dimensional fitness landscape is constructed by arranging chromosomes in two dimensions on the landscape such that bit strings that differ in the value of only one bit are neighbours, then using the fitness of the chromosomes as the third dimension. The NK landscape is widely used by the evolutionary computation community to generate epistatic landscapes as test functions for search and optimisation techniques (De Jong et al., 1997; Heckendorn et al., 1999; Skellett et al., 2005; Smith & Smith, 2001).

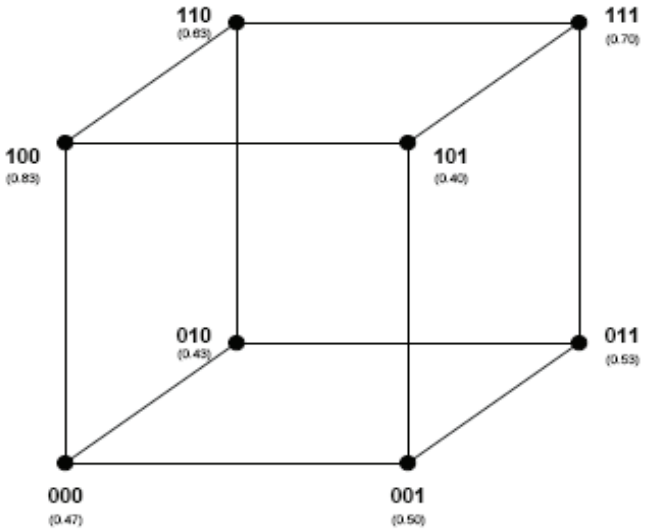


Fig. 2. An example of a fitness landscape for bit strings of length three, where the neighbourhood relation is defined by point mutation. Fitness values (randomly assigned) are shown in parentheses. *Reproduced from (Hordijk, 1996)*

Weinberger (1990a; 1991a) proposed a fitness landscape model in which the landscape is represented as a graph on which the vertices correspond to individuals and have associated fitness values, and traversing the edge of the graph corresponds to the action of a genetic operator (mutation, crossover etc.) and so taking a step on the landscape. Jones (1995a; 1995b), Culberson (1994), and later Hordijk (1996) describe similar fitness landscape models in which the landscape is represented as a graph (e.g. Fig. 2). Reidys and Stadler (2002) analyse a similar fitness landscape topography, focussing on the geometry of the moves

from one vertex to another and provide a mathematical treatment of the edge traversals and the resultant complex topographies.
Culberson's model described a landscape defined by a crossover operator rather than mutation, and in which the graph vertices represented a population of points. Jones presents a similar model, and expands it further to include the concept that each genetic operator defines its own separate landscape (Jones 1995a). In Jones' "one operator, one landscape" model, an evolutionary algorithm which implements the genetic operators selection, mutation and crossover makes transitions on three separate landscapes (Fig. 3). According to Jones' model the evolutionary algorithm takes steps on the mutation landscape, after which individuals are paired to form vertices on the crossover landscape and further steps are then taken on that landscape, and then the population is gathered into a vertex on the selection landscape for a further step there. The neighbourhood relation in Jones' model is therefore simply defined by the genetic operator for each landscape.

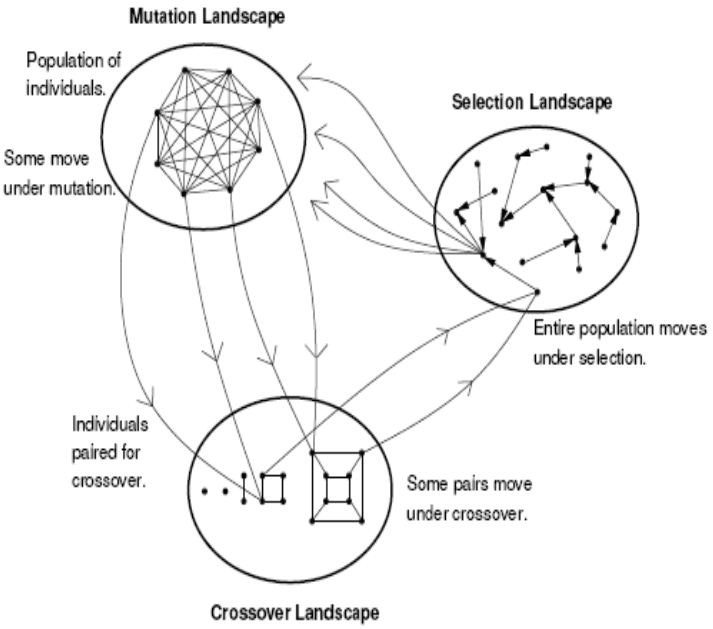


Fig. 3. A simplified view of an evolutionary algorithm operating on three landscapes.
Reproduced from (Jones 1995a)

More recently Moraglio and Poli (2004) presented a new topological framework for evolutionary algorithms and as part of that framework redefine the mutation and crossover operators to be more tightly linked to the fitness landscape. In the model proposed by Moraglio and Poli the genetic operators are defined by the fitness landscape upon which they operate – the genetic operators are a natural consequence of the neighbourhood relation and distance metric of the fitness landscape. For example, under what Moraglio and

Poli call topological uniform crossover, offspring are defined as those individuals that lie between the parents on the fitness landscape. Similarly, under topological ε-mutation, offspring are defined as those individuals that lie ε steps away from the parent on the fitness landscape.
In essence, the fitness landscape is a metaphor – a metaphor in which the landscape is often thought of as a 3-dimensional terrain, where the peaks (hills or mountains) represent areas of high fitness and the valleys between the peaks low fitness. A fitness landscape which depicts the fitness values of a population that varies greatly in fitness will likely display many local high peaks surrounded by deep valleys. Such a landscape is said to be rugged. Similarly, if many individuals in the population have a similar fitness, the landscape will vary little and is said to be flat. Search on some landscapes is notionally easier than search on others – search on a predominantly flat landscape is likely to be difficult, as is search on a rugged landscape with peaks and valleys randomly distributed.
In evolutionary biology the proliferation of an individual's genetic material is considered the ultimate objective of life, and the success or failure of a particular genotype, or its phenotypic expression, is most often measured by the number of progeny it produces – so for evolutionary biology, the fitness of an individual is a function of the number of progeny produced by that individual. For evolutionary optimisation the objective is usually less nebulous, and the success or failure of a particular genotype, or phenotype, is measured by a well-defined fitness function, and typically the number of progeny produced by an individual is a function of the fitness of that individual. Fitness landscapes are used in evolutionary biology and evolutionary optimisation to show the correlation between genotypes, or phenotypes, as a measure of success or failure and search difficulty (Jones & Forrest, 1995b; Kauffman, 1989; Kauffman & Levin, 1987; Langdon & Poli, 1998b; Vassilev, 1997).

4. Which Fitness Landscape?

As discussed in the previous section there are several possible definitions of, and representations for, fitness landscapes, and choosing the definition and representation which best describes the combination of the problem being studied and the algorithm being used to study it is extremely important.
A fitness landscape is most often defined by three basic attributes:

- a search space
- a relation that defines which points are neighbours in the search space
- a fitness function that assigns a fitness value to each point in the search space

A significant problem with this definition of fitness landscapes is the specification of the neighbourhood relation. The neighbourhood relation and its specification is extremely important because any discussion of landscapes invariably involves the terms "peaks" and "valleys", and no peak or valley can exist without the notion of neighbourhood – a peak is only a peak because it is higher than its neighbours.
Often the neighbourhood relation is defined in simple terms, such as the hamming distance for bit strings, or by defining all single bit mutations of a bit string as neighbours. This potentially ignores an important ingredient in the evolutionary processes: evolutionary algorithms are usually governed by some combination of several operators. A definition of

the neighbourhood relation in terms of the actual mix of genetic operators (for example, selection, mutation and crossover) being used by the algorithm would seem to be more relevant and useful to an analysis of the performance of the algorithm. Since the search is for individuals of good fitness, the individuals which comprise the search space and their associated fitness values should ideally be arranged as a fitness landscape which has individuals of better fitness clustered together so that the landscape contains peaks and valleys representing the fitness extremes. The issue is how to represent the fitness landscape to achieve that, and a critical question is: what attributes of an individual should determine the locality of that individual on the fitness landscape?

If the neighbourhood relation is not defined correctly in terms of describing the actions of the algorithm being used to perform the search, then the fitness landscape is not the landscape searched by the search algorithm, and so not necessarily indicative of the difficulty of the search. On a physical landscape points are neighbours, or next to each other, because (for example) a person walking on that landscape can traverse the distance between those points in a single step. The corollary is that if a searcher walking on a landscape is not able to traverse the distance between two points in a single step, then those points are not next to each other and so, by definition, are not neighbours on the landscape defined by that searcher and the associated granularity of search (a single step). Simply plotting attributes of the members of the search space against fitness – for example chromosome length in the case of a genetic algorithm, or program length for genetic programming – while possibly useful for visualising the fitness distribution of the search space, may not be sufficient to describe the fitness landscape traversed by the search algorithm, since there may be no reason to expect that the search algorithm could traverse the distance between neighbouring points on that landscape with a single step.

As discussed earlier, a fitness landscape is a metaphor and an aid to the visualisation of the operation of a search algorithm, but for anything other than the actual landscape traversed by the search algorithm the metaphor is flawed and the visualisation is misleading. For example, the analogy of the search algorithm to a short-sighted explorer wandering over a (possibly rough) terrain (Langdon & Poli, 2002) is only valid for the actual fitness landscape searched by the algorithm, and anything else gives a misleading view of the complexity, or ruggedness, of the landscape. For the explorer analogy to be useful, the neighbourhood relation must reflect the notion that an individual's neighbour on the fitness landscape is any individual in the search space which can be reached in a single step of the explorer. In the case of an evolutionary algorithm, the explorer is the algorithm, and the single step is the combination of whichever genetic operators are implemented by that algorithm: selection, mutation and crossover. It should also be remembered that during the search the explorer may not be able to see all its theoretical neighbours. This is because, for an evolutionary algorithm using crossover, the individuals reachable in a single step from one parent depend upon the other individuals selected for mating, and while an individual can theoretically mate with any other individual in the search space, in reality the individuals available for selection are restricted by the size of the population. So to continue the analogy of the short-sighted explorer, not only is the explorer short-sighted, but also lacks peripheral vision.

While it should be remembered that the 3-dimensional fitness landscape is just a metaphor, since in most cases the surface being traversed by the searcher will be multi-dimensional and so complex that there will be no landscape that could actually be visualised, the analogy

of the searcher on a 3-dimensional landscape is a useful aid to imagining how a search algorithm might use the available attributes of the search space.
It is important to understand that the (metaphoric) fitness landscape is defined by a number of attributes, which include the search algorithm, and not solely by the specification of the problem. It is not valid to visualise a fitness landscape without reference to a search algorithm, and then decide what algorithm will best search the landscape – the operation of the search algorithm defines the neighbourhood relation, and so the shape, of the landscape.
Previous work with fitness landscapes recognises the need to refer to the search algorithm when characterising the landscape (Hordijk, 1996; Jones & Forrest, 1995b; Reeves, 1999; Vassilev et al., 2000). Jones and others (Hordijk, 1996; Jones & Forrest, 1995a) further suggest that each operator employed by a search algorithm (e.g. selection, mutation and crossover for a genetic algorithm) should be viewed as operating on its own landscape. The notion that search operators define and act on separate landscapes may be useful for studying the effect of individual operators, but the combined effect of each move on each landscape needs to be considered. If the algorithm employs multiple operators, then the output of the algorithm is some combination of those operators, so it is not reasonable to consider a move on the "mutation landscape" without considering how that then affects a subsequent move on the "crossover landscape".
A further problem with the "one operator, one landscape" approach is determining what constitutes an operator. Jones (1995a) has, in the case of evolutionary algorithms, suggested that selection, mutation and crossover should each define and operate on separate landscapes, but none of those operators are necessarily atomic. Is an operator considered to define its own landscape simply because it has a well-known label? The breakdown of the evolutionary operation into the three operators of selection, mutation and crossover seems somewhat arbitrary.

5. Landscape Measures

Early work on the characterisation of landscapes involved analysing structural parameters such as the number and distribution of local minima (Edwards & Anderson, 1975; Sherrington & Kirkpatrick, 1975; Tanaka & Edwards, 1980). More recently, several methods for measuring and analysing landscapes for search algorithms have been proposed. The methods proposed can be categorised into two broad streams: statistical measures (Altenberg, 1995; Hordijk, 1996; Jones & Forrest, 1995b; Lipsitch, 1991; Manderick et al., 1991; Weinberger, 1990b; Weinberger, 1991b) and information measures (Borenstein & Poli, 2005a,b,c; Vassilev, 1997; Vassilev et al., 2000). Borenstein and Poli have extended the information measure to include a measure of the performance of the algorithm and so define a "performance landscape" which may prove useful but does not yet have sufficient history to gauge its efficacy. All the methods proposed have in common the notion that the points in the search space are arranged according to some neighbourhood relationship, and a measure of fitness, performance or information content associated with the points defines the ruggedness of the landscape.
The methods used to measure and analyse the structure of fitness landscapes in this work are the autocorrelation method suggested by Weingberger (1990b; 1991b), and the information content approach suggested by Vassilev et al. (Vassilev, 1997; Vassilev et al., 2000). These methods were chosen because they are different methods of measuring the

structure of landscapes and, while there seems to be no generally accepted standard approach, both methods have gained some favour and are commonly cited as reasonable landscape characterisation methods (Hordijk, 1996; Merz & Freisleben, 1999; Smith et al., 2002; Stadler, 1996). Jones' *Fitness Distance Correlation* (Jones & Forrest, 1995b) is an interesting landscape measure but requires that the solution be known in order to calculate the metric, so is not applicable to this work.

5.1 Autocorrelation and Correlation Length

Correlation measures are the most commonly used means of characterising fitness landscapes and are seen as good indicators of ruggedness. A landscape is characterised as rugged when it contains multiple local maxima (minima). Typically, the more rugged the landscape the lower the average fitness correlation between neigbouring points on the landscape. Autocorrelation refers to the correlation of a time series with its own past and future values, and is also known as serial correlation, referring to the correlation between members of a series of numbers arranged in time. Autocorrelation is a correlation coefficient, but instead of measuring the correlation between two different variables, autocorrelation measures the correlation between two values of the same variable at times X_i and X_{i+k}, where k is the number of time steps, or lag, between the two values.

Autocorrelation definition (Hordijk, 1996; Weinberger, 1990b; Weinberger, 1991b):

Given measurements $Y_1, Y_2, \ldots, Y_N$, at time $X_1, X_2, \ldots, X_N$, where N is the number of measurements, and

$$\overline{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i, N > 0$$

the time lag autocorrelation function is defined as

$$r_k = \frac{\sum_{i=1}^{N-k}(Y_i - \overline{Y})(Y_{i+k} - \overline{Y})}{\sum (Y_i - \overline{Y})^2}, N > k \quad (1)$$

Note that from Equation 1, if $|r_k| \approx 1.0$ there is much correlation between the points k steps apart in the series, whereas if $|r_k| \approx 0.0$ there is little correlation.

Weinberger proposed that a random walk be generated on the fitness landscape, where each step on the walk is taken between neighbouring points but the neighbour to which the step is taken is selected randomly, and the fitness values for each point visited during the random walk form a time series of numbers. The autocorrelation function can then be used as a measure of the ruggedness of the landscape described by the random walk.

The correlation length of a series of numbers is the largest distance, or time lag, between points for which some correlation exists. Hordijk (1996) defines the correlation length of a time series as one less than the first time lag for which the autocorrelation falls inside the region bounded by the two-standard-error bound (i.e. one less than the first time lag at which the autocorrelation becomes statistically equal to zero, making the correlation length

the largest time lag for which the correlation between two points is still statistically significant). This is the method used for calculating the correlation length in this work. The two-standard-error bound *e* for a series of *N* points is defined as

$$e = \pm \frac{2}{\sqrt{N}}, N > 0$$

so the correlation length *l is* defined in this work as the first lag for which

$$|r_k| < \left| \frac{2}{\sqrt{N}} \right|, N > 0 \quad (2)$$

5.2 Information Content

An alternative to the statistical autocorrelation measure proposed by Weinberger is Vassilev's information content method, based on both classic and algorithmic information theory (Chaitin, 1987; Shannon, 1948). Vassilev et al. propose three information measures that characterise the structure of a fitness landscape from a series of points generated by a random walk over the landscape (Vassilev, 1997; Vassilev et al., 2000) :

- Information Content – characterises the ruggedness of the landscape.
- Partial Information Content – measures the modality of the landscape.
- Information Stability – the sensitivity of the information content measures.

These measures are calculated by generating a random walk of length *n* on the fitness landscape, with the aim being to extract information by characterising the series of points as an ensemble of objects. To calculate the information content, a string $S(\varepsilon)$ is constructed representing a group of objects generated from a random walk over the fitness landscape, where

$$S(\varepsilon) = S_1, S_2, ..., S_n, S_i \in \{\bar{1}, 0, 1\}$$

and $S(\varepsilon)$ is enumerated according to the function

$$S_i = \psi_{f_t}(i, \varepsilon), t = 1..n$$

and

$$\psi_{f_t} = \begin{cases} \bar{1} & if \quad f_i - f_{i-1} < -\varepsilon \\ 0 & if \quad |f_i - f_{i-1}| \le \varepsilon \\ 1 & if \quad f_i - f_{i-1} > \varepsilon \end{cases}$$

Thus the string $S(\varepsilon)$ defines a sequence of objects where each object is represented by a substring S_iS_{i+1} being a sub-block of length two of the string $S(\varepsilon)$.
The parameter ε is a real number taken from the interval [0.0, 1.0] (in this case) which defines neutral fitness and determines the accuracy with which the string $S(\varepsilon)$ is defined. If the absolute fitness difference between neighbouring points is less than ε, the points are

considered to be of equal fitness. This means that as ε increases from 0.0 to the maximum possible fitness difference between points along the walk (1.0), the amount of fitness change (entropy), and the sensitivity of ψ_{f_i}, decrease to zero.

The information content $H(\varepsilon)$ is defined as the entropic measure of the group of sub-blocks of length two of string $S(\varepsilon)$, and is given by

$$H(\varepsilon) = -\sum_{p \neq q} P_{[pq]} \log_6 p_{[pq]} \tag{3}$$

$P_{[pq]}$ are frequencies of the possible blocks pq of elements from the set $\{\bar{1},0,1\}$ given by

$$P_{[pq]} = \frac{n_{[pq]}}{n}$$

where n[pq] is the number of occurrences of pq in $S(\varepsilon)$ and n is the length of $S(\varepsilon)$.

The partial information content $M(\varepsilon)$ is a measure of the modality (the number or frequency of local optima) of the landscape, and is calculated by filtering out elements of the string $S(\varepsilon)$ which are not essential for measuring modality to create a new string $S'(\varepsilon)$, then measuring the length μ of the new string. The string $S'(\varepsilon)$ is defined as

$$S'(\varepsilon) = S_{i_1}, S_{i_2}, ..., S_{i_k}, S_{i_j} \neq 0, S_{i_j} \neq S_{i_{j-1}}, j > 1$$

Thus the string $S'(\varepsilon)$ has the form $\bar{1}\,1\,\bar{1}\,1\,\bar{1}\ldots$, representing the slopes of the path taken by the random walk over the landscape, and so is empty if $S(\varepsilon)$ contains only 0s.

The partial information content $M(\varepsilon)$ is given by

$$M(\varepsilon) = \frac{\mu}{n} \tag{4}$$

Note that when $M(\varepsilon) = 1.0$, the path taken by the random walk over the landscape is considered to be maximally multimodal, and when $M(\varepsilon) = 0.0$, the path is flat.

The information stability ε^* is defined as the smallest value of ε for which the landscape becomes flat (i.e. for which $S'(\varepsilon)$ is empty). Since ε governs the sensitivity of the information content and partial information content measures, ε^* is a measure of the difference in fitness between neighbouring points on the random walk.

6. Fitness Landscape Definition

As discussed in sections 3 and 4, there has been much previous work done with regard to fitness landscapes, and there is a variety of views and some disagreement. The previous sections of this work have presented a discussion of the issues. For the purposes of this work it is critical that the fitness landscape being analysed be formally defined, and that the definition be related to the search algorithm used to search the landscape.

Genetic algorithms are typically thought of as performing a search on a landscape described by single-bit mutation, where mutation performs a *local* search and the crossover operation is depicted as a *hyperstep* on the mutation landscape, allowing mutation to perform a local

search on a different, usually distant part of the landscape. For this work however, the fitness landscape is considered to be defined by the overall operation of the genetic algorithm.

The autocorrelation and information content measures used in this work to characterize the fitness landscape provide an analysis of a series of numbers: in the case of the autocorrelation measures, the analysis indicates how well correlated numbers in the series are. Of significance is not so much how the numbers in the series are collected or generated, but that the series be representative of the entity – in this case the fitness landscape – being measured. The random walk proposed by Weinberger is a good means to generate a representative series of points for the single-bit mutation landscape (provided that the walk is sufficiently long or that the landscape is statistically isotropic), but is not directly applicable to the landscape defined by the overall operation of the genetic algorithm.

Consider an observer watching a genetic algorithm searcher perform a *random walk* on a fitness landscape and assume that although the observer is able to discern the granularity of the search (the genetic algorithm's single steps), the means by which the GA determines where each step takes it is hidden from the observer.

The observer sees the searcher walking randomly over the landscape and considers points on the landscape one step apart to be neighbours. The definition of the neigbourhood relation is of no consequence to, and is not required by, the observer since the searcher is defining neigbouring points by performing the walk. If the random walk performed by the genetic algorithm searcher was sufficiently long, and the "altitude" (fitness) at each step recorded for the observer, the entire fitness landscape would be determined by observation. The landscape so determined would be the precise fitness landscape defined by the search algorithm.

A random walk of s steps is conducted as follows:

- An individual i_0 is randomly selected from the search space
- For each step s, $s = 1 .. maxsteps$
- i_{s-1} undergoes mutation with probability $P_{mutation}$
- Another individual i, $i \neq i_{s-1}$, is randomly selected from the search space
- Crossover is performed between i and i_{s-1} with probability $P_{crossover}$, resulting in two new individuals i'_1 and i'_2, both of which are neighbours of (a single step from) i_{s-1}
- Set $i_s = i'_1$ and step to i_s

This "black box" view of the genetic algorithm operation and consequential determination of the neigbourhood relation and fitness landscape doesn't change the actual operation of the genetic algorithm or the conceptual notion of the algorithm conducting a parallel search of different areas of the fitness landscape. What this view does is change the notion of the step size of the genetic algorithm from the result of a single genetic operator to the amalgamation of the genetic operators used by the algorithm, so the perception of the topography of the landscape is changed accordingly. This view of the fitness landscape satisfies the requirement that the landscape neigbourhood relation be defined by the search algorithm and is the definition used for the robot soccer problem addressed by this work.

7. Search Space and Fitness Landscape Analysis

Since the skills with which a player is endowed will undoubtedly affect the ability of the player to learn goal scoring behaviour, a key question is *"How does changing the skills available to the soccer player affect, or change, the fitness landscape?"* It would seem to be somewhat intuitive that changing the skills available to the players in some way alters the fitness landscape over which the search is conducted. In fact the fitness landscape is altered by changing the skills available to players, but it changes because the underlying search space is different for each set of skills: the chromosomes are interpreted differently for each skillset, and in fact may take on different values for each skillset depending upon the range of skills available in the skillset, so the individuals (players) that comprise the search space are different for each skillset. Since changing the skills available to the players defines a new search space it also defines a new fitness landscape.

The same is not true of changing the function used for the fitness evaluation. Different fitness functions alter the fitness landscape, not the underlying search space, because the individuals that comprise the search space remain the same. Furthermore, since only the means by which the individuals are evaluated is changed, the change to the fitness landscape is not in the neighbourhood relation but only in the fitness values of the individuals. For this reason, changing the fitness function has the potential to significantly change the fitness landscape, and so affect the effectiveness of the search process. Some fitness functions could make "mountains out of molehills" – teasing gradient information out of otherwise flat landscapes. Sometimes this is valid, but sometimes the definition of the fitness function effectively solves the problem for the search algorithm.

The difficulty of learning goal-scoring behaviour is evidenced by the size of the search spaces defined by the different skillsets of the players. These search spaces range in size from 1.55×10^{158} different chromosomes for the base skillset of {Turn, Kick, Dash} to 7.4×10^{161} for the complete skillset. The calculation of search space size is described in detail in (Riley, 2005).

7.1 Experiments Performed

A number of experiments were performed in order to examine how the fitness landscape, and the performance of the resulting search over it, is affected by varying the player skillset (refer to the full list of available skills shown in Table 1 of the previous chapter), the default action (previous chapter, Table 2) and the fitness function (discussed below). In addition to the evolutionary search, five random walks (as described in section 6) were conducted for each experiment, each walk starting at a randomly selected point on the fitness landscape and continuing for a duration of 100,000 steps. The statistics gathered during the walks are also analysed.

7.2 Fitness Functions Evaluated

Two fitness functions were compared: a composite fitness function and a simple, goals-only fitness function. For both fitness functions implemented the fitness values range from 0.0 to 1.0, with 1.0 being the worst fitness possible, and optimal fitness values approaching 0.0.

The simple, goals-only fitness function rewards a player for goals scored only – the greater the number of goals scored the greater the reward. The goals-only fitness function was implemented as:

$$f = \begin{cases} 1.0 & , goals = 0 \\ \dfrac{1.0}{2.0 \times goals} & , goals > 0 \end{cases} \quad (5)$$

where *goals* is the number of goals scored by the player.

The composite fitness function rewards, in order of importance:

- the number of goals scored in a trial
- minimising the distance of the ball from the goal

This combination was chosen to reward players primarily for goals scored, while players that do not score goals are rewarded on the basis of how close they are able to move the ball to the goal on the assumption that a player which kicks the ball close to the goal is more likely to produce offspring capable of scoring goals. This decomposes the original problem of evolving goal-scoring behaviour into the two less difficult problems:

- evolve ball-kicking behaviour that minimises the distance between the ball and goal, and
- evolve goal-scoring behaviour from the now increased base level of skill and knowledge

The composite fitness function was implemented as:

$$f = \begin{cases} \begin{cases} 1.0 & , kicks = 0 \\ 0.5 + \dfrac{dist}{2.0 \times fieldLen} & , kicks > 0 \end{cases} & , goals = 0 \\ \dfrac{1.0}{2.0 \times goals} & , goals > 0 \end{cases} \quad (6)$$

where *goals* is the number of goals scored by the player
kicks is the number of times the player kicked the ball
dist is the minimum distance of the ball to the goal
fieldLen is the length of the field

Note that both the simple and composite fitness functions represent the problem as a minimisation problem, so a lower fitness value is a better result than a higher fitness value.

7.3 Genetic Algorithm Parameters

The genetic algorithm parameters used in the experiments were chosen empirically after some experimentation. The population size and number of generations were chosen to provide reasonable population diversity and search space coverage. The GA parameters are shown in Table 1. Each player was allowed a fixed time in which to score as many goals as possible.

Skill	Description
Maximum Chromosome Length	64 genes
Population Size	500
Maximum Generations	500
Selection Method	Elitist: *a percentage of the best players is retained, with the remainder selected using the Roulette Wheel method.*
Elitist Retention Percentage	2.5
Crossover Method	Single Point Cut and Splice
Crossover Probability	0.95
Mutation Rate	10%
Mutation Probability	0.15

Table 1. GA parameters

7.4 Results and Analysis

For each experiment the following information is presented for analysis and comparison:

- Data from the evolutionary search:
  - Line graphs showing the population average fitness and individual best fitness for generations 1 to 500. Note that a *lower* fitness is a better fitness.
  - A bar chart showing the cumulative fitness distribution for all individuals evaluated during the 500 generations, showing the percentage of all individuals evaluated which failed to kick the ball (denoted by "nm" on the graph x-axis), moved the ball closer to the goal (graduated), kicked one goal, two goals, three goals etc.
- Data from the random walks:
  - A correlogram showing the autocorrelation data for time lags 1 to 100 for the five random walks and the two-standard error bounds.
  - A graph showing the information content data for $0 <= \varepsilon <= 1.0$ for the five walks.
  - A table showing:
    - the mean fitness and fitness variance of the points visited in the random walks.
    - the fitness distribution for all individuals evaluated during the random walks.
    - the average autocorrelation for the first time lag (i.e. for immediate neighbours).
    - the average correlation length.
    - the average information content $H(\varepsilon)$ and average partial information content $M(\varepsilon)$ for selected ε.
    - the average information stability ε^*.

7.4.1 Experiment 1: Composite Fitness; All Skills; Default Action = Hunt Action 1

Experiment 1 is the problem for which the search algorithm has been given the most help in the form of initial player skills, default action, and a composite fitness function to guide the search. The data shown on the graph of population average fitness (Fig. 4) tend to indicate that the population as a whole ceases to improve after 30 to 40 generations though, as evidenced by the graph of best fitness values, individuals of good fitness continue to be found beyond that point. The percentage of the population exhibiting ball-kicking or goal-scoring behaviour is reasonably high, as shown by the frequency distribution (Fig. 5).

From the data presented in Table 2 it is apparent that although the mean fitness of the 100,000 individuals evaluated during the random walk is close to 1.0 (indicating no ball movement) and the variance small, the random walk did find individuals which exhibited goal-scoring behaviour, and in fact found more than 100 individuals which scored multiple goals.

The autocorrelation data shown in Fig. 6 and the correlation length given in Table 2 indicate that the fitness landscape for this problem (as described by the random walk) offers a reasonable amount of gradient information that the search algorithm can use to guide the search. With an autocorrelation of ~0.32 for points on the random walk a single step apart and a fairly steep descent for points further apart, the correlation between next and near neighbours on this fitness landscape is not so high that a search algorithm is led unerringly to a solution, but with a good correlation and a long correlation length the problem, in this form, should be readily solved by a search algorithm able to take advantage of the landscape features.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.98283							
Fitness Variance	0.00669							
Individual Fitness	**No Movement**	**Ball Movement**	**Goals**					
			1	2	3	4	5	>5
Number of Individuals	95974	3462	434	99	26	4	1	0
Autocorrelation *r(1)*	0.31716							
Correlation Length	41							
ε	**Information Content *H(ε)***		**Partial Information Content *M(ε)***					
0.0	0.12447		0.03266					
0.2	0.09770		0.03020					
0.4	0.06179		0.01269					
0.6	0.00754		0.00098					
0.8	0.00229		0.00025					
0.885	0.00000		0.00000					

Table 2. Experiment 1: Random walk statistics

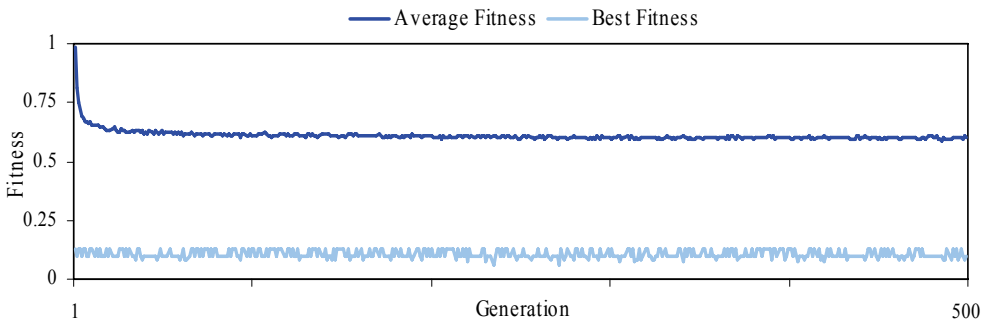


Fig. 4. Experiment 1: Average and best fitness

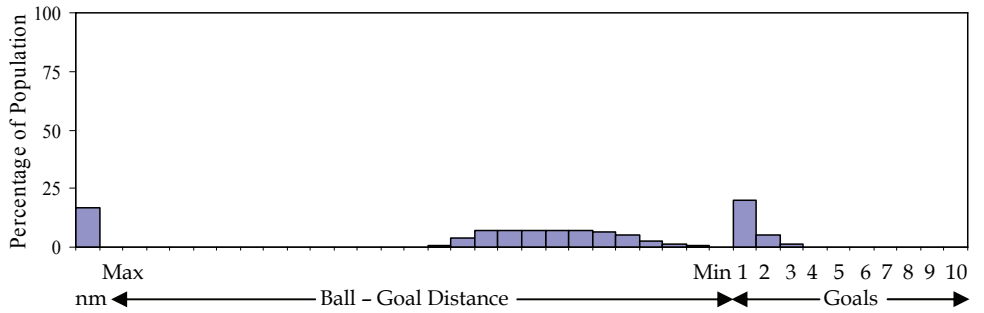


Fig. 5. Experiment 1: Ball movement and goals scored

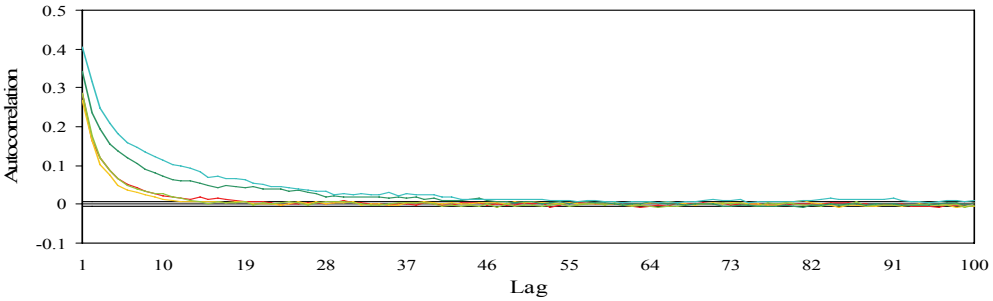


Fig. 6. Experiment 1: Autocorrelation

The information content graph shown in Fig. 7 supports the autocorrelation data for this experiment. Information stability is quite high at 0.885, indicating a high difference in fitness among neighbouring points, so pointing to some good gradient information being present in the landscape. *H(0.0)* is not particularly large, indicating that the diversity of shapes on the landscape is not high. Similarly *M(0.0)* is relatively small, indicating that the degree of modality of the landscape is low.

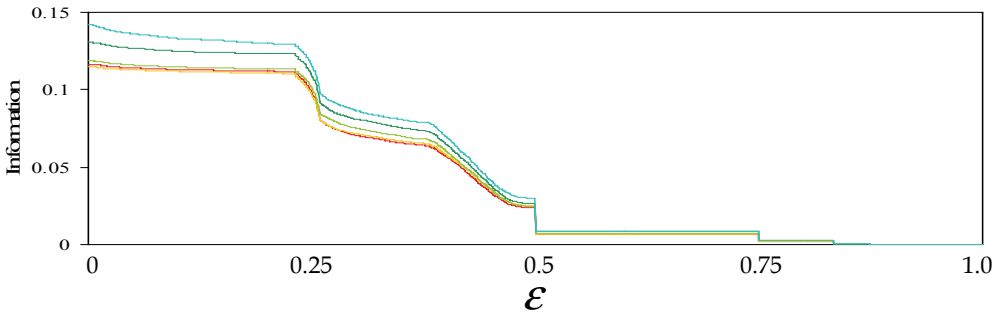


Fig. 7. Experiment 1: Information content

7.4.2 Experiment 2: Composite Fitness; Base Skills; Default Action = Hunt Action 1

The difference between this experiment and experiment 1 is that instead of the player being endowed with all available skills, the player in this experiment has only the base skills of turn, kick and dash. The player has hunt action 1 configured as the default action. The fitness function is the composite fitness function.

The data shown on the graph of population average fitness (Fig. 8) indicate that improvement of the population stops at about generation 150, and although the graph of best fitness values indicates that individuals exhibiting goal-scoring behaviour continue to be found, terminating the search after generation 150 would not have adversely affected the result. Fig. 9 shows that the percentage of the population exhibiting goal-scoring behaviour is extremely small, with a very large proportion of the population not kicking the ball at all.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99947							
Fitness Variance	0.00014							
Individual Fitness	**No Movement**	**Ball Movement**	**Goals**					
			1	2	3	4	5	>5
Number of Individuals	99776	224	0	0	0	0	0	0
Autocorrelation *r(1)*	0.10724							
Correlation Length	9							
ε	**Information Content *H(ε)***		**Partial Information Content *M(ε)***					
0.0	0.01268		0.00180					
0.1	0.01258		0.00178					
0.2	0.01256		0.00178					
0.3	0.00202		0.00022					
0.371	0.00000		0.00000					

Table 3. Experiment 2: Random walk statistics

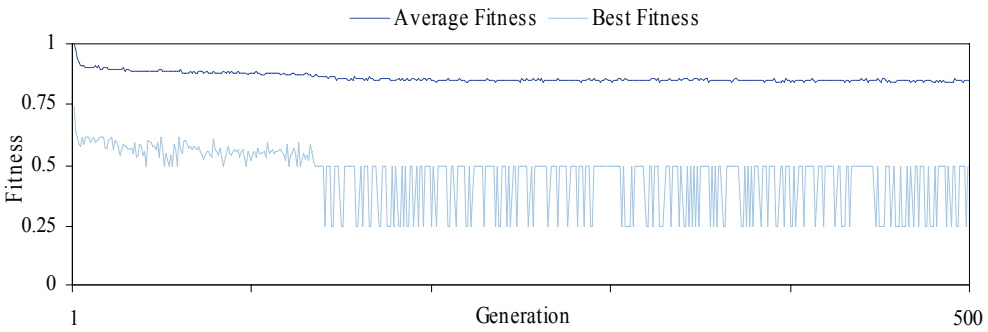


Fig. 8. Experiment 2: Average and best fitness

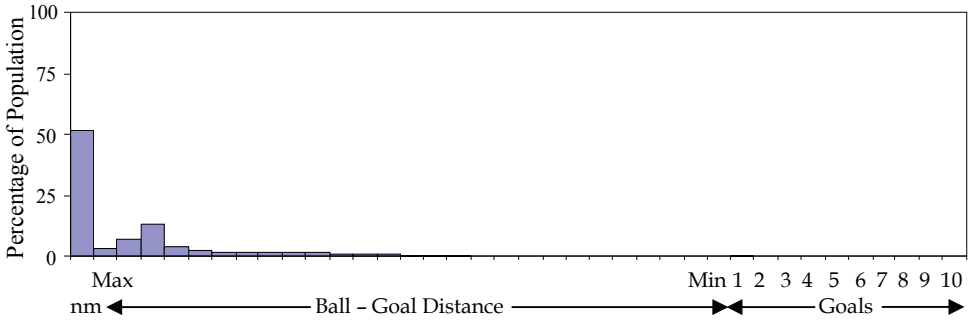


Fig. 9. Experiment 2: Ball movement and goals scored

These results show clearly the effect of removing from the players a range of mid-level hand-coded skills, and raise the question of what effect removing those skills has on the structure of the fitness landscape and how that affects the search.

The autocorrelation graph (Fig. 10) and correlation length (Table 3) indicate that the fitness landscape for this problem offers only a limited amount of useful gradient information the search algorithm can use to guide the search. With an autocorrelation of ~0.1 for points on the random walk a single step apart and falling to zero for points just a few steps further apart, the correlation between next and near neighbours on this fitness landscape indicates that the structure of the fitness landscape is close to random and not as conducive to search as was the fitness landscape of experiment 1, thus increasing the difficulty of the problem.

Similar to experiment 1, the data presented in Table 3 show the mean fitness of the 100,000 individuals evaluated during the random walk is close to 1.0 (indicating no ball movement) and the variance very small, but in this case no individuals exhibiting goal-scoring behaviour were found during the random walk.

The information content graph for this experiment (Fig. 11) supports the autocorrelation data. Information stability is relatively low at 0.371, indicating a low difference in fitness among neighbouring points. With the autocorrelation data indicating a near random landscape, and information stability indicating a low fitness variation among neighbouring

points, there is almost no useful gradient information in the landscape to guide the search. *H(0.0)* is very small, indicating that the diversity of shapes on the landscape is very low. Similarly *M(0.0)* is extremely small, indicating that the landscape lacks any real degree of modality. Both values further indicate the lack of useful landscape data to guide the search. The data presented all indicate that the removal of a set of mid-level, hand-coded skills has changed the relative difficulty of the problem, and that this is a result of the structure and features of the fitness landscape being altered by the problem representation – what was a landscape reasonably rich in features that helped guide the search has become a relatively barren landscape lacking in information useful for search.

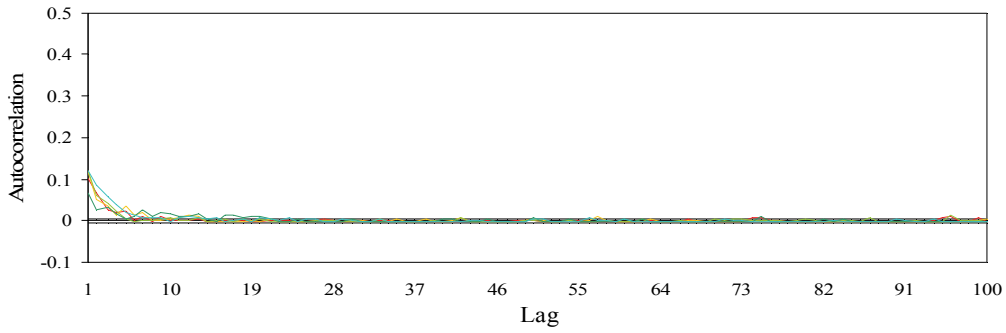


Fig. 10. Experiment 2: Autocorrelation

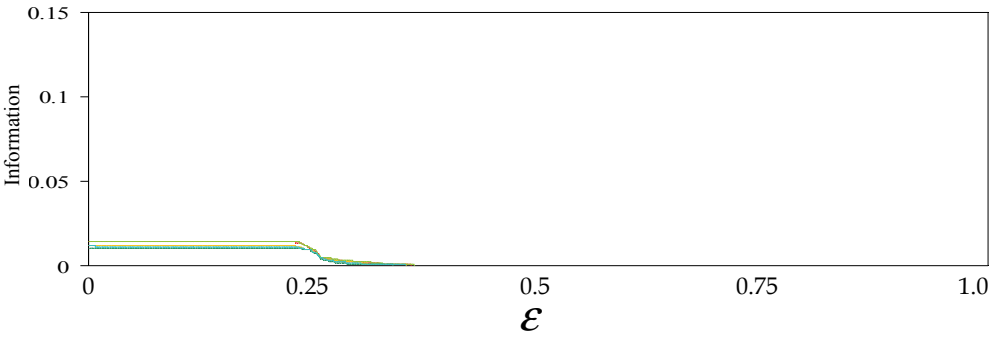


Fig. 11. Experiment 2: Information content

7.4.3 Experiment 3: Composite Fitness; All Skills; Default Action = Hunt Action 3

For experiment 3 the player is again given access to all available skills. The difference between experiment 3 and experiment 1 is that the default hunt action for experiment 3 is limited to a 90° turn in a randomly chosen direction.

The graphs of population average and best fitness (Fig. 12) indicate that while the evolutionary search was able to find individuals which scored multiple goals, the overall population hardly improved for the duration of the search. The percentage of the

population exhibiting ball-kicking or goal-scoring behaviour is low, as shown by the frequency distribution (Fig. 13). From the data presented in Table 4 it is evident that the random walk found very few individuals which exhibited goal- scoring behaviour – significantly fewer than in experiment 1.

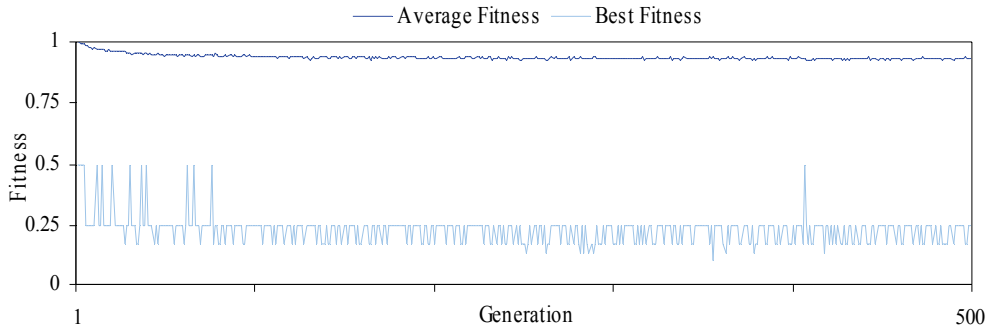


Fig. 12. Experiment 3: Average and best fitness

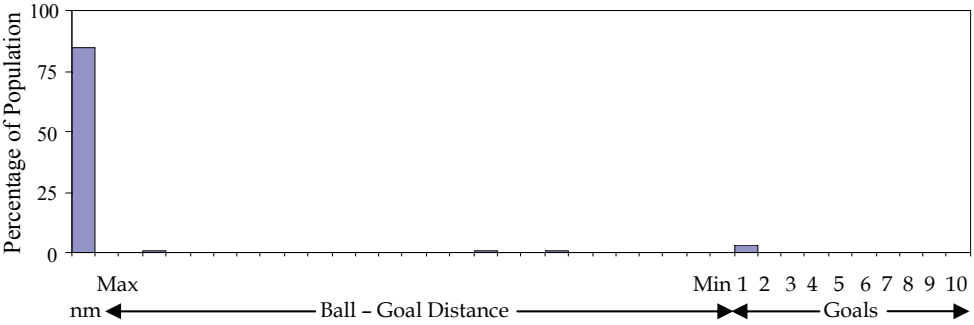


Fig. 13. Experiment 3: Ball movement and goals scored

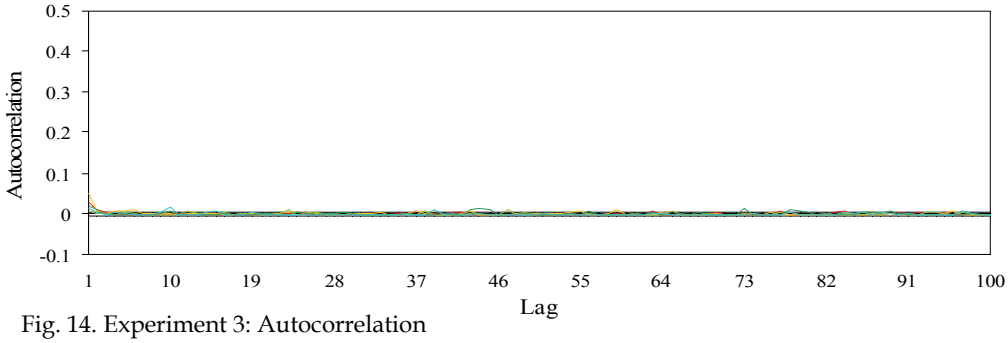


Fig. 14. Experiment 3: Autocorrelation

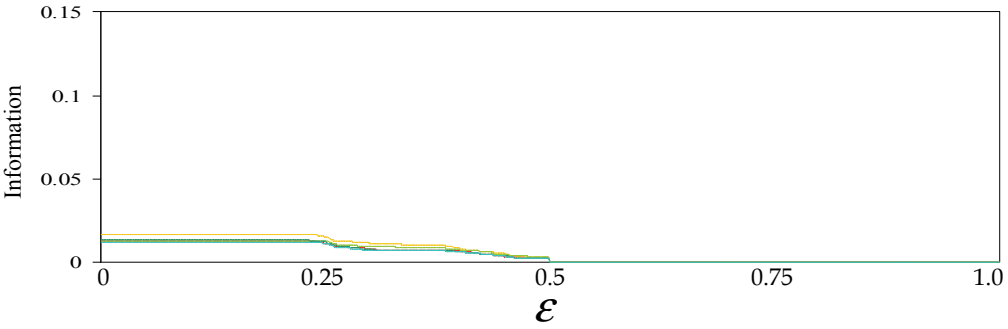


Fig. 15. Experiment 3: Information content

Random Walk Statistics (Average over 5 walks)									
Mean Fitness	0.99927								
Fitness Variance	0.00029								
Individual Fitness	**No Movement**	**Ball Movement**	**Goals**						
			1	2	3	4	5	>5	
Number of Individuals	99801	168	31	1	0	0	0	0	
Autocorrelation *r(1)*	0.02546								
Correlation Length	3								
ε	**Information Content *H(ε)***		**Partial Information Content *M(ε)***						
0.0	0.01344		0.00193						
0.1	0.01344		0.00193						
0.2	0.01344		0.00193						
0.3	0.00899		0.00120						
0.4	0.00703		0.00090						
0.5	0.00000		0.00000						

Table 4. Experiment 3: Random walk statistics

The autocorrelation and information content data shown in Fig. 14, Fig. 15 and Table 4 paint a similar picture – from the data it is clear that the fitness landscape lacks much of the gradient information evident in experiment 1, and is even closer to random than in experiment 2. Information stability at 0.5 indicates a reasonable difference in fitness among neighbouring points, but with an autocorrelation of 0.025 for points on the random walk a single step apart and a correlation length of only 3, this fitness landscape offers the searcher virtually no assistance. *H(0.0)* and *M(0.0)* are both extremely small, indicating that the diversity of shapes and the degree of modality on the landscape are very low, further indicating the lack of any useful landscape data to guide the search. This is entirely due to the removal of any sort of intelligence from the default hunt action.

7.4.4 Experiment 4: Goals-only Fitness; All Skills; Default Action = Hunt Action 1

In experiment 4 the player is given all available skills and the default action is hunt action 1. The fitness function used for this experiment is the goals-only fitness function, so the player is rewarded only for scoring goals.

The results for experiment 4 are very similar to those for experiment 1, with the major difference being the autocorrelation and information content data (Fig. 18, Fig. 19 and Table 5). The population average fitness is higher for experiment 4, but this is expected for a goals-only fitness function - since players are not rewarded for moving the ball close to the goal players not actually scoring goals are assigned the worst possible fitness of 1.0. The best fitness and fitness frequency graphs (Fig. 16 and Fig. 17) are almost identical (for goal-scoring behaviour in the case of the fitness frequency graph), indicating that the evolutionary search was almost not affected by the change in fitness evaluation.

The autocorrelation and information content graphs (Fig. 18 and Fig. 19) indicate that the fitness landscape has somewhat less gradient information useful for search, but still sufficient to facilitate a successful search. In fact, the data from Table 5 show that the random walk for experiment 4 found more players which scored goals than did the random walk for experiment 1. This is further indication that when the players are given the full complement of hand-coded skills the number of solutions in the search space increases, and so the difficulty of the problem and the importance of the fitness evaluation are both reduced significantly.

From Table 5 it can be seen that as for experiment 1, information stability is quite high at 0.882, indicating a high difference in fitness among neighbouring points and the existence of good gradient information on the landscape. $H(0.0)$ is much lower than for experiment 1, indicating that the diversity of shapes on the landscape is much lower.

Random Walk Statistics (Average over 5 walks)								
Mean Fitness	0.99672							
Fitness Variance	0.00189							
Individual Fitness	**No Movement**	**Ball Movement**	**Goals**					
			1	2	3	4	5	>5
Number of Individuals	99417	n/a	456	104	19	3	1	0
Autocorrelation *r(1)*	0.09497							
Correlation Length	20							
ε	**Information Content $H(\varepsilon)$**		**Partial Information Content $M(\varepsilon)$**					
0.0	0.03028		0.00520					
0.2	0.03028		0.00520					
0.4	0.03020		0.00519					
0.6	0.00868		0.00116					
0.8	0.00229		0.00025					
0.882	0.00000		0.00000					

Table 5. Experiment 4: Random walk statistics

Similarly *M(0.0)* is very small, and much smaller than for experiment 1, indicating that the degree of modality of the landscape is much lower. These are expected results for the goals-only fitness evaluation compared to the composite fitness function.

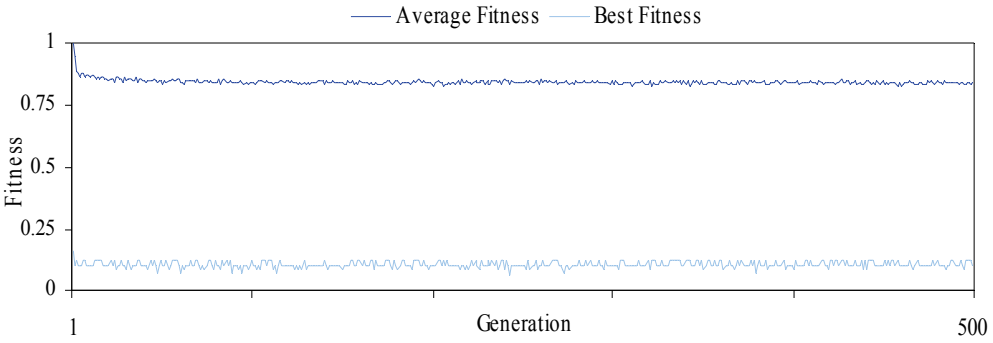


Fig. 16. Experiment 4: Average and best fitness

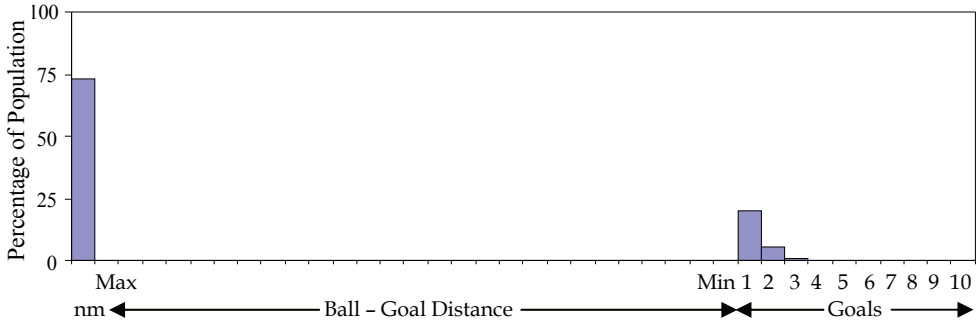


Fig. 17. Experiment 4: Ball movement and goals scored

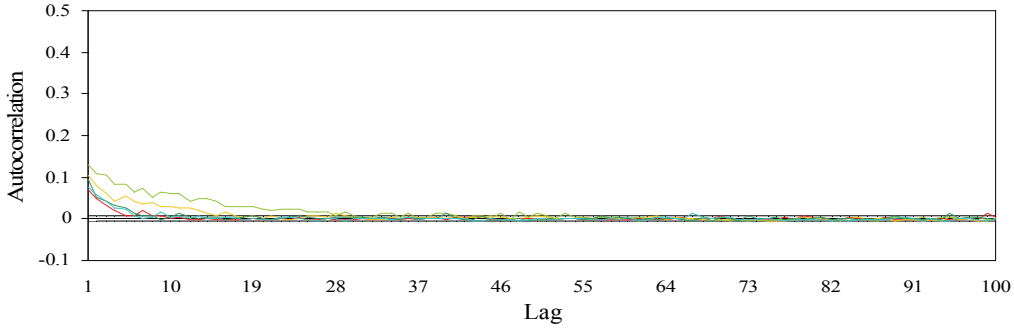


Fig. 18. Experiment 4: Autocorrelation

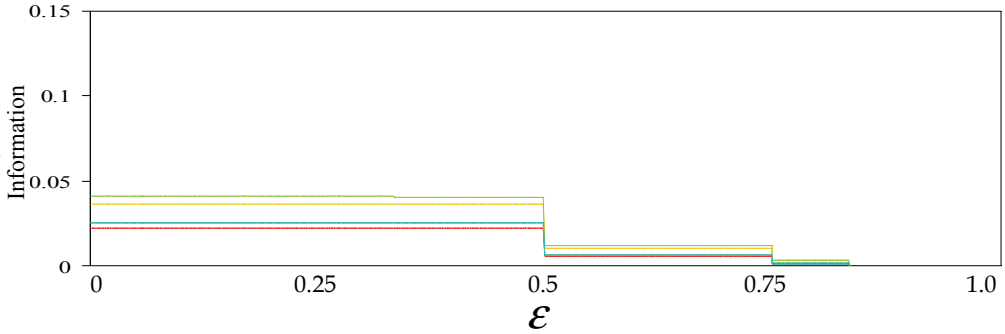


Fig. 19. Experiment 4: Information content

8. Conclusions and Further Work

This work has presented a method for the analysis of the fitness landscape described by the problem of learning goal-scoring behaviour using a genetic algorithm. The method described uses landscape measures to examine features of the fitness landscape such as the degree of correlation between, or randomness of, points on the landscape, the modality of the landscape, and the ruggedness of the landscape. These landscape features are used to better understand the reasons for the difficulty of the problem, especially for evolutionary algorithms. The *autocorrelation* and *information content* measures are indicative of the ease with which the searcher will be able to identify the most productive areas of the landscape and in which direction (on the landscape) the search should proceed.

Also presented in this work is an analysis of the fitness landscape for the robot soccer problem using the landscape measures described, showing how the measures can be used to assess how varying different attributes (e.g. initial player skills, player default action and the fitness function) changes problem difficulty. The analysis indicates that as more human expertise and expert knowledge is injected into an evolutionary search algorithm via hand-coded innate skills, smart default actions and a composite fitness function to guide the search, the problem of learning goal-scoring behaviour for robot soccer becomes more solvable. In the case of the experiments performed for this work, the genetic algorithm is able to evolve individuals which display goal-scoring behaviour to the extent that they are able to consistently score multiple goals in the tests conducted. This is an expected result, and the focus of this work is on using the fitness landscape analyses to explain the results and the difficulty of the problem.

The fitness landscape analysis further indicates that as human expertise and expert knowledge is removed from the algorithm by restricting the hand-coded innate skills and smart default actions available to the players, or by using a simple goals-only fitness function, at some threshold the problem becomes intractable for the evolutionary algorithm. This suggests that while there may be gradient information in the fitness landscape, as the human expertise is reduced the density of solutions in the search space becomes very low, the "mountain ranges" in the landscape begin to become isolated from each other, and the landscape begins to appear as a flat plain, sparsely populated by individual peaks – so the problem begins to resemble a *needle-in-a-haystack* problem. In this case the genetic algorithm is not able to locate the sparsely distributed gradient information any way other than by

randomly sampling the search space and it performs little or no better than random search - confirmed by a small number of tests performed comparing genetic and random search. This is an indication that the injection of human expertise and expert knowledge acts like a magnifying glass to the searcher - as more expertise and knowledge is injected the fitness landscape features conducive to search are magnified, and as the expertise and knowledge is removed those landscape features become less discernible. As the granularity of the injected knowledge is decreased (e.g. a less rich set of skills) the modality of the landscape decreases and the gradients between peaks become smoother.

This is one of the underlying causes of the difficulty of the robot soccer problem for evolutionary algorithms, and the analysis presented in this work suggests that with a difficult problem such as robot soccer an evolutionary algorithm will only find a reasonable solution if one of:

- a rich skill set (placing the initial population closer to the desired solution)
- a composite fitness function (providing a solution recipe)

is present - if both of those components are absent the problem becomes very difficult for evolutionary algorithms.

Varying player skills changes the search space of the problem – a different set of players is being searched for each skillset. It is therefore not surprising that changing the innate skills of the players has a significant effect on the outcome of the search – searching the set of players which have essentially been pre-coded to exhibit good soccer-playing skills is almost guaranteed to find more players exhibiting better goal-scoring behaviour than searching the set of players which have no pre-coded soccer-playing skills. This is akin to searching for good goal-scoring behaviour in an assembled group of premier league soccer players, and then searching an assembled group of randomly selected people off the street – every now and then a person from the street will score a goal, but the premier league players will do it more often and more consistently.

The default player action was shown to be a very significant determinant of performance for the robot soccer problem. This was also seen in the results presented by Luke (1998). For this work the default action was almost always invoked when the ball could not be seen. The importance of the default action tends to suggest that while some basic skills such as kicking, dribbling, perhaps even passing, can be somewhat readily learned, learning subtle tactics and strategies is difficult.

Similarly, it was shown through experiments that compared the effectiveness of a simple goals-only fitness function against that of a composite fitness function that a composite fitness function can better guide the search. A well-designed composite fitness function can effectively act as a recipe for solving a problem. A great deal of human expertise and expert knowledge can be injected into a composite fitness function, and in effect the problem can virtually be solved by the fitness function before the algorithm begins the search.

Further work to ascertain the best balance between the two components identified as being necessary for successful evolutionary search (a rich skill set or a composite fitness function) would be useful, as would work to determine if there is a limit in the level of initial skills at which a difficult problem becomes intractable.

Within the evolutionary computation community there are several definitions and variations of the fitness landscape traversed by an evolutionary algorithm. This work proposed a definition of the fitness landscape described by the combination of the genetic

algorithm and the problem being investigated, but further work needs to be done to develop a consistent definition of the fitness landscape, or landscapes, described by the operation of an evolutionary algorithm. With a consistent landscape definition, more work can be done to develop measures that will aid researchers in tuning algorithms and search methods based on landscape analysis.

The fitness landscape definition used for this work is based on the assumption that the combination of the genetic operators implemented defines the neighbourhood relation of the landscape, and that the fitness function defines the height of the landscape at each point. Given this definition the shape, ruggedness, and indeed the searchability of the landscape, will be affected by changes to any of the genetic operators and the fitness function. This suggests that it would be advantageous to experiment with various combinations of those parameters in order to determine if a search might be successful or what combination would improve the chance of a successful search, prior to launching into an exhaustive search. A useful avenue of further work is to develop a framework for this prior analysis so that a description of the means by which the analysis should be conducted can be determined (for example, the number and size of random walks conducted over the landscape in order to calculate the landscape measures), as well as quantifying the desired values for the landscape measures.

9. References

Altenberg, L. (1995). The Schema Theorem and Price's Theorem. In: *Foundations of Genetic Algorithms 3*, D. Whitley & M. Vose, (Ed.), pp. 23-49, Darrell Whitley and Michael Vose, San Francisco CA.

Borenstein, Y. & Poli, R. (2005a). Information Landscapes, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 1551-1522, Washington DC, June 2005, The ACM Press, NY.

Borenstein, Y. & Poli, R. (2005b). Information Landscapes and Problem Hardness, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 1425-1431, Washington DC, June 2005, The ACM Press, NY.

Borenstein, Y. & Poli, R. (2005c). Information Landscapes and the Analysis of Search, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pp. 1287-1294, Washington DC, June 2005, The ACM Press, NY.

Chaitin, G.J. (1987). *Information, Randomness & Incompleteness: Papers on Algorithmic Information Theory*, World Scientific, ISBN 9971-50-479-0, Singapore.

Culberson, J.C. (1994). Mutation-Crossover Isomorphisms and the Construction of Discriminating Functions. *Evolutionary Computation*, Vol. 2, No. 3, 1994, pp. 279-311.

De Jong, K.A.; Potter, M.A. & Spears, W.M. (1997). Using Problem Generators to Explore the Effects of Epistasis, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 338-345, East Lansing MI, July 1997, T. Bäck, (Ed.), Morgan Kaufmann, San Francisco CA.

Edwards, S.F. & Anderson. P.W. (1975). Theory of Spin Glasses. *Journal of Physics F: Metal Physics*, Issue 5, May 1975, pp. 965-974.

Frauenfelder, H.; Bishop, A. R.; Garcia, A. & Perelson, A., eds. (1997). Landscape Concepts in Physics and Biology. *Special Issue of Physica D*, Vol. 107, 1997, pp. 2-4, Elsevier Science, Amsterdam Holland.

Goldberg, D.; Korb, B. & Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, Vol. 3, No. 5, 1989, pp. 493-530.

Heckendorn, R.B.; Rana, S. & Whitley, D.L. (1999). Test Function Generators as Embedded Landscapes, In: *Foundations of Genetic Algorithms 5*, W. Banzhaf & C. Reeves, (Ed.), pp. 183-198, Morgan Kaufmann, ISBN 1558-60-559-2, San Francisco CA.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor: MI.

Hordijk, W. (1996). A Measure of Landscapes, *Evolutionary Computation*, Vol. 4, No. 4, 1996, pp. 335-360.

Jang, J.-S.; Sun, C.-T. & Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*, Prentice Hall, ISBN 0-13-261066-3, Upper Saddle River NJ.

Jones, T. (1995a). One Operator, One Landscape, *Technical Report 95-02-025*, Santa Fe Institute, Santa Fe NM.

Jones, T.C. (1995b). Evolutionary Algorithms, Fitness Landscapes and Search, PhD Thesis, 1995, University of New Mexico, Albuquerque NM.

Jones, T. & Forrest, S. (1995a). Genetic Algorithms and Heuristic Search, *Technical Report 95-02-021*, Santa Fe Institute, Santa Fe NM.

Jones, T. & Forrest, S. (1995b) Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 184-192, ISBN 1-55860-370-0, Pittsburgh PA, July 1995, L. J. Eshelman, (Ed.), Morgan Kaufmann, San Francisco CA.

Kauffman, S.A. (1989). Adaptation on Rugged Fitness Landscapes, In: *Lectures in the Sciences of Complexity*, D. Stein, (Ed.), pp. 527-618, Addison-Wesley, ISBN 1-201-51015-4, Redwood City CA.

Kauffman, S.A. & Levin, S. (1987). Towards a General Theory of Adaptive Walks on Rugged Landscapes, *Journal of Theoretical Biology*, Vol. 128, No. 1, September 1987, pp. 11-45.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I. & Osawa, E. (1995). Robocup: The Robot World Cup Initiative, *Working Notes of the 1995 IJCAI Workshop on Entertainment and AI/Alife*, pp. 19-24, Montreal Canada, August 1995.

Langdon, W.B. & Poli, R. (1998a). Why "Building Blocks" Don't Work on Parity Problems, *Technical Report CSRP-98-17*, School of Computer Science, University of Birmingham.

Langdon, W.B. & Poli, R. (1998b). Why Ants are Hard, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 193-201, Madison WI, July 1998, Koza, J.R.; Banzhaf, W.; Chellapilla, K.; Deb, K.; Dorigo, M.; Fogel, D.B.; Garzon, M.H.; Goldberg, D.E.; Iba, H. & Riolo, R.L. (Eds.), Morgan Kaufmann, San Francisco CA.

Langdon, W.B. & Poli, R. (2002). *Foundations of Genetic Programming* , Springer, ISBN 978-3540424512.

Lima, P.; Custódio, L.; Akin, L.; Jacoff, A.; Kraezschmar, G.; Ng, B.K.; Obst, O.; Röfer, T.; Takahashi, Y. & Zhou, C. (2005). RoboCup 2004 Competitions and Symposium: A Small Kick for Robots, a Giant Score for Science. *AI Magazine* 26(2), 2005, pp. 36-61.

Lipsitch, M. (1991). Adaptation on Rugged Landscapes Generated by Iterated Local Interactions of Neighboring Genes, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 128-135, San Diego CA, July 1991, Morgan Kaufmann, San Francisco CA.

Luke, S. (1998). Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Madison WI, July 1998, Koza, J.R.; Banzhaf, W.; Chellapilla, K.; Deb, K.; Dorigo, M.; Fogel, D.B.; Garzon, M.H.; Goldberg, D.E.; Iba, H. & Riolo, R.L. (Eds.), Morgan Kaufmann, San Francisco CA.

Lush, J.L. (1935). Progeny Test and Individual Performance as Indicators of an Animal's Breeding Value. *Journal of Dairy Science*, Vol. 18, 1935, pp. 1-19.

Manderick, B., de Weger, M. & Spiessens, P. (1991). The Genetic Algorithm and the Structure of the Fitness Landscape, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 143-150, San Diego CA, July 1991, Morgan Kaufmann, San Francisco CA.

Merz, P. & B. Freisleben, B. (1999). Fitness Landscapes and Memetic Algorithm Design. In: *New Ideas in Optimization*, Corne, D., Dorigo, M. & Glover, F. (Eds.), pp. 245-260, McGraw-Hill, ISBN 978-0077095062, London.

Moraglio, A. & Poli, R. (2004). Topological Interpretation of Crossover, *Proceeding of the 2004 Genetic and Evolutionary Computation Conference*, Part I, pp. 1377-1388, Seattle WA, June 2004, Deb, K., Poli, R., Banzhaf, W., Beyer, H.-G., Burke,E.K., Darwen, P.J., Dasgupta, D., Floreano, D., Foster, J.A., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Thierens, D., Tyrrell, A.M. (Eds.), Springer.

Reeves, C.R. (1999). Landscapes, Operators and Heuristic Search. *Annals of Operational Research*, Vol. 86, 1999, pp. 473-490.

Reidys, C.M. and P.F. Stadler. (2002). Combinatorial Landscapes. *Society for Industrial and Applied Methematics, SIAM Review*, Vol. 44, No. 1, 2002, pp. 3-54.

Riedmiller, M.; Gabel, T.; Knabe, J. & Strasdat, H. (2005). Brainstormers 2D - Team Description 2005. In: *RoboCup 2005: Robot Soccer World Cup IX.*, Bredenfeld, A.; Jacoff, A.; Noda, I. & Takahashi, Y. (Eds.), Springer, Berlin.

Right, A.H., Row, J.E. & Neil, J.R. (2002). Analysis of the Simple Genetic Algorithm on the Single-peak and Double-peak Landscapes, *Proceedings of the 2002 Congress on Evolutionary Computation*, pp. 214-219, Hawaii, May 2002, IEEE Computer Society, Washington DC.

Riley, J. (2003). The SimpleSoccer Machine Learning Environment, *Proceedings of the First Asia-Pacific Workshop on Genetic Programming*, pp. 24-30, Canberra, Australia, December 2003, Cho, S-B.; Nguen, H.X. & Shan, Y. (Eds.).

Riley, J. (2005). Evolving Fuzzy Rules for Goal-Scoring Behaviour in a Robot Soccer Environment, *PhD Thesis*, RMIT University: Melbourne, Australia.

Shannon, C.E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, Vol. 27, 1948, pp. 379-423; 623-656.

Sherrington, D., & S. Kirkpatrick, S. (1975). Solvable Model of a Spin Glass. *Physical Review Letters*, Vol. 35, 1975, pp. 1792-1796.

Skellett, B., Cairns, B., Geard, N., Tonkes, B. & Wiles, J. (2005). Maximally Rugged NK Landscapes Contain the Highest Peaks, *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, pp. 579-584, Washington DC, June 2005, Beyer, H.-G., O'Reilly, U.-M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P. & Zitzler, E., (Eds.), ACM Press.

Smith, R.E. & J.E. Smith, J.E. (2001). New Methods for Tunable, Random Landscapes. In: *Foundations of Genetic Algorithms 6*, Morgan Kaufmann, ISBN 978-1558607347, San Francisco CA.

Smith, T., Husbands, P., Layzell, P. & O'Shea, M. (2002). Fitness Landscapes and Evolvability. *Evolutionary Computation*, Vol. 10, No. 1, 2002, pp. 1-34.

Stadler, P. (1996). Landscapes and Their Correlation Functions. *Journal of Mathematical Chemistry*, Vol. 20, 1996, pp. 1-45.

Stone, P. (1998). Layered Learning in Multiagent Systems, *PhD Thesis*, Carnegie Mellon University, Pittsburgh PA.

Stone, P. & Sutton, R. (2001). Scaling Reinforcement Learning Toward RoboCup Soccer, *Proceedings of the Eighteenth International Conference on Machine Learning*, Williamstown MA, July 2001, Brodley, C.E. & Danyluk, A.P. (Eds.), Morgan Kaufmann, San Francisco CA.

Tanaka, F. & S. F. Edwards, S.F. (1980). Analytic Theoeries of the Ground State Properties of a Spin Glass. I. Ising Spin Glass. *Journal of Physics F: Metal. Physics*, Vol. 10, 1980 pp. 2769-2778.

Vassilev, V.K. (1997). Information Analysis of Fitness Landscapes, *Proceedings of the Fourth European Conference on Artificial Life*, pp. 116-124, Brighton, UK, July 1997, Husbands, P. & Harvey, I. (Eds.), The MIT Press.

Vassilev, V., T. Fogarty, and J. Miller. (2000). Information Characteristics and the Structure of Landscapes. *Evolutionary Computation*, Vol. 8, No. 1, 2000, pp.31-60.

Weinberger, E. (1990a). Fourier and Taylor Series on Fitness Landscapes. *Biological Cybernetics*, Vol. 65, 1990, pp. 321-330.

Weinberger, E. (1990b). Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference. *Biological Cybernetics*, Vol. 63, 1990, pp. 325-336.

Weinberger, E. (1991a). Measuring Correlations in Energy Landscapes and Why it Matters. In: *Information Dynamics*, Atmanspacher, H. & Scheingraber, H. (Eds.), Plenum Press, New York NY.

Weinberger, E. (1991b). Local Properties of Kauffman's N-K model: A Tunably Rugged Energy Landscape. *Physical Review A*, Vol. 44, No. 10, 1991, pp. 6399-6413.

Wright, S. (1932). The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution, *Proceedings of the Sixth International Congress on Genetics*, pp. 356-366, Ithaca NY, August 1932, Jones, D.F., (Ed.), Brooklyn Botanic Garden NY.

Zadeh, L. (1965). Fuzzy Sets. *Journal of Information and Control*, vol. 8, 1965, pp. 338-353.

17

Motion Detection and Object Tracking for an AIBO Robot Soccer Player[1]

Javier Ruiz-del-Solar and Paul A. Vallejos
Universidad de Chile
Chile

1. Introduction

Movement analysis is a fundamental ability for any kind of robot. It is especially important for determining and understanding the dynamics of the robot's surrounding environment. In the case of robot soccer players, movement analysis is employed for determining the trajectory of relevant objects (ball, teammates, etc.).

However, most of the existing movement analysis methods require the use of a fixed camera (no movement of the camera while analyzing the movement of objects). As an example, the popular background subtraction algorithm employs a fixed background for determining the foreground pixels by subtracting the current frame with the background model. The requirement of a fixed camera restricts the real-time analysis that a soccer player can carry out. For instance, a human soccer player very often requires the determination of the ball trajectory when he is moving himself, or when he is moving his head, for making or planning a soccer-play. If a robot soccer player should have a similar functionality, then it requires an algorithm for real-time movement analysis that can perform well when the camera is moving. The aim of this work is to propose such an algorithm for an AIBO robot. This algorithm can be adapted for almost any kind of mobile robot.

The rationale behind our algorithm is to compensate in software the camera movement using the information about the robot body and robot head movements. This information is used to correctly align the current frame and the background. In this way, a stabilized background is obtained, although the camera is always moving. Afterward, different traditional movement analysis algorithms can be applied over the stabilized background. Another feature of our algorithm is the use of a Kalman Filter for the robust tracking of the moving objects. This allows to have reliable detections and to deal with common situations such as double detections or no detection in some frames because of variable lighting conditions.

This chapter is organized as follows. In section 2 we present some related work. In section 3 is described the here proposed motion analysis algorithm for AIBO robots. Experiments

[1] This research was partially funded by Millenium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.

using real video sequences are described and analyzed in section 4. Finally, some conclusions are given in section 5.

2. Related Work

A large literature exists concerning motion analysis in video streams using fixed cameras. As an example, the PETS event is held every year, in which several state-of-the-art tracking and surveillance systems are presented and tested. See for example (Ferryman, 2002) and (Ferryman, 2005). Different approaches have been proposed for moving object segmentation; including frame difference, double frame difference, and background suppression or subtraction. In the absence of any a priori knowledge about target and environment, the most widely adopted approach is background subtraction (Cucchiara et al., 2002). Motion History is another simple and fast motion detection algorithm. According to (Piater & Crowley, 2001), the Motion History and Background Subtraction algorithms have complementary properties, and when possible it is useful their joint use.
Image alignment using gradient descending is one of the most used alignment algorithms. It can be divided into two formulations: the additive approach, which starts with an initial estimation of the parameters, then iteratively finds appropriate parameters' increments, until the estimated parameters converge (Lucas & Kanade, 1981); and the compositional approach, which estimates the parameters using an incremental warp. This last approach iteratively solves the estimation problem using an incremental warp of the images to be aligned with respect to a template. This allows pre-computing the Jacobean more efficiently (Baker & Matthews, 2001). However, the key to obtain an efficient algorithm is switching the role of the image and the template. This leads to the formulation of the inverse compositional algorithm (Baker & Matthews, 2002), where the most computationally expensive operations are pre-calculated, allowing a faster convergence. In (Ishikawa et al., 2002) it was proposed the robust inverse compositional algorithm as an extension to the inverse compositional algorithm, allowing the existence of outliers into the alignment with almost the same efficiency.
Regarding object tracking, Kalman Filtering, Extended Kalman Filtering and Particle Filtering (also known as Condensation and Monte Carlo algorithms) are some of the most common used algorithms. Due to its simplicity, the Kalman filter is still been used in most of the general-purpose applications when the linearity and Gaussianity assumptions are valid.
The here-proposed motion detection and tracking system is based on the described algorithms: background difference and motion history for motion detection, robust inverse compositional algorithm for the image and background alignment, and Kalman filtering for the tracking of moving objects.

3. Proposed Motion Detection System

3.1 System Overview

In figure 1 a block diagram of the proposed system is shown. The system is composed by four main subsystems: *Image Alignment*, *Motion Detection*, *Detection Estimation*, and *Background Update*. In the *Image Alignment* module, the last updated background image (B_{k-1}) and the last frame image (I_{k-1}) are aligned with respect to the current frame image (I_k). The camera motion angles (α_k) are employed in this alignment operation. Both aligned

images, B_k^* and I_{k-1}^*, respectively, are then compared with I_k in the *Motion Detection* module for determining the current moving pixels. As a result of these comparisons, the *Motion History* and *Background Subtraction* algorithms generate preliminary detections (a set of moving pixels), D_{H_K} and D_{B_k}, respectively. These detections are joined in the Rejection Filter module, and a final set of candidate blobs (moving objects) Det_k is obtained using adjacent moving pixels. The motion detections are analyzed in the *Detection Estimation* module using a Kalman Filter, and the final detections Det_k* are obtained. Finally, the background is updated using B_k^*, I_k and Det_k* (which defines the new foreground pixels) by the *Background Update* module.

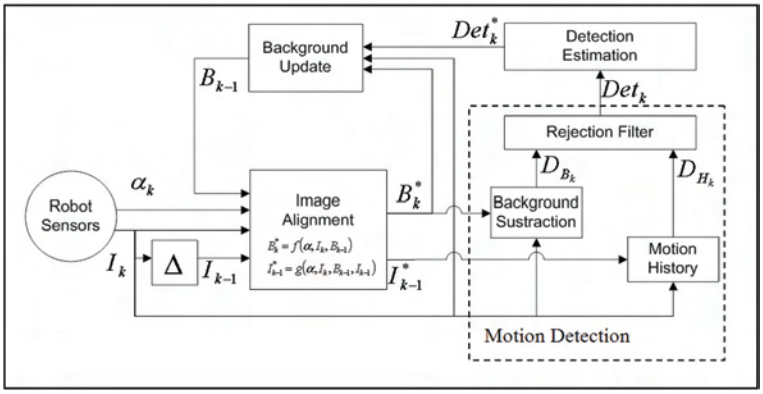


Fig. 1. Block diagram of the proposed system. Parameters are described in the main text

3.2 Image Alignment

The alignment of the last updated background image (B_{k-1}) and the last frame image (I_{k-1}) is implemented using the robust inverse compositional algorithm (Ishikawa et al., 2002). The alignment operation is implemented as a sequence of incremental warps (see section 2). The initial estimation of the warp is calculated based on the camera motion angles (stored in the α_K vector; they correspond to the tilt, pan and roll camera rotation angles). The initial estimated warp is a composition of a rotation followed by a displacement. The angle of rotation is estimated as the variation of the roll angle of the camera, while the displacement D_x/D_y in the X/Y axis corresponds to the pan/tilt angle:

$$\begin{aligned} \alpha_R &= \alpha_{pan2} \cdot \sin(BA - \alpha_{tilt2}) - \alpha_{pan1} \cdot \sin(BA - \alpha_{tilt1}) + \alpha_{roll2} - \alpha_{roll1} \\ Dx &= (\alpha_{pan2} - \alpha_{pan1}) \cdot 180 \\ Dy &= \left((BA - \alpha_{tilt2}) \cdot \cos(\alpha_{pan2}) - (BA - \alpha_{tilt1}) \cdot \cos(\alpha_{pan1})\right) \cdot 180 \end{aligned} \tag{1}$$

where α_R represent the rotation in radians, D_x and D_y represent the displacement in pixels in their respective axis, BA is the angle of the body, and the angles α_{tilt1}, α_{pan1}, α_{roll1}, α_{tilt2},

α_{pan2} , α_{roll2} are the tilt, pan and roll angles of the robot's head in the last and in the current image, respectively, measured in radians.
Then the warp is defined by a set of six parameters as:

$$\begin{aligned} Wx &= (1+P1)\cdot x + P3\cdot y + P5 \\ Wy &= P2\cdot x + (1+P4)\cdot y + P6 \end{aligned} \tag{2}$$

where (W_x, W_y) define the new pixel coordinates which initial coordinates were (x,y). A pure displacement warp has the parameters *P1* to *P4* equals to zero, and the parameters *P5* and P6 equals to the displacement in pixels in the x and y axis respectively. A pure rotation warp has the parameters *P1* to *P4* equals to the rotation matrix, and the parameters *P5* and *P6* equals to zero. Finally, a compound warp of a rotation followed by a translation have the parameters: $P1 = \cos(\alpha_R) - 1$, $P2 = \sin(-\alpha_R)$, $P3 = \sin(\alpha_R)$, $P4 = \cos(\alpha_R) - 1$, $P5 = Dx$, $P6 = Dy$.
For aligning B_{k-1}, the area of the current image (I_k), which has being estimated to overlap the background, is chosen as a template for the algorithm. This preliminary template is divided into nine blocks (sub-images). In each block, the normalized variance of its pixels (intra-block variance), and the normalized variance of the error with respect to the correspondent block in the background (inter-block variance) are calculated. A *variability factor* is computed as the quotient of the intra-block variance and the inter-block variance. The six blocks with the largest variability factors are selected as the final templates for the background alignment. Taking into account the normal camera motion, B_{k-1} and I_k should have different spatial sizes for a correct alignment. In our implementation, B_{k-1} has the same height than I_k, but the double of its width. We will denote the set of parameters defining the warping of the background P_B. The algorithm for obtaining P_B is detailed described in (Ishikawa et al., 2002).
For aligning $I_{k-1,}$ the calculated warp of B_{k-1} is employed as a first approximation. However, given that I_{k-1} and I_k have the same size, the calculated warp has to be actualized with a composition with a prior displacement to achieve the same spatial configuration of the background (I_{k-1} should be translated into background spatial coordinates). Then, the result has to be composed with a post displacement to reach the spatial configuration of the current image (the warped image should be taken back to its original coordinates). Thus, defining *P1* as the set of parameters needed to produce a displacement equal to the last position of I_{k-1} inside the background, and *P2* as the set of parameters needed to produce a displacement equal to the inverse of the estimated final position of I_k inside the background, the warp needed to align I_{k-1} is (the set of parameters defining this warping are P_l):

$$W(\mathbf{x}, \mathbf{P_1}) = W(\mathbf{x}, \mathbf{P_2}) \circ (W(\mathbf{x}, \mathbf{P_B}) \circ W(\mathbf{x}, \mathbf{P_1})) \tag{3}$$

For simplicity on the notation, x denotes both spatial image coordinates. The function $W(x,P^*)$ corresponds to a warping operation over x using the set of parameters P^*.

3.3 Motion Detection

The motion detection module is composed by three algorithms, *Motion History* and *Background Subtraction* for movement detection, and *Rejection Filter* for filtering wrong detections and forming the movement blobs.

3.3.1. Motion History

This module keeps a representation of the motion history in the sequence of frames for each time step *k*. For detecting the motion, the distance in the luminance space between the current image I_k and the last aligned image I^*_{k-1} is calculated, obtaining the difference image DM_k defined as follow:

$$DM_k(\mathbf{x}) = \begin{cases} mIncrement & \text{if } \left|I_k(\mathbf{x}) - I^*_{k-1}(\mathbf{x})\right| > T_m \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Where *mIncrement* corresponds to a factor of increment in the motion, and T_m corresponds to a motion threshold. DM_k contains the initial set of points that are candidate to belong to the MVOs (Moving Visual Object). In order to consolidate the blobs to be detected, a 3x3 morphological closing (Russ, 1995) is applied to DM_k. Isolated detected moving pixels are discarded applying a 3x3 morphological opening (Russ, 1995). The motion history image MH_k, calculated from DM_k, is then updated as:

$$MH_k = MH_{k-1} * DecayFactor + DM_k \tag{5}$$

Finally, all pixels of MH_k whose luminance is larger than a motion detection threshold (T_h) are considered as pixels in motion. These pixels generate the detection image D_{H_K}. 3x3 morphological closing and opening are applied to D_{H_K}.

3.3.2. Background Subtraction

Foreground pixels are selected at each time *k* by computing the distance between the current image I_k and the current aligned background B^*_k, obtaining D_{B_K} as:

$$D_{B_k}(\mathbf{x}) = \begin{cases} 1 \;\; if \;\; \left|I_k(\mathbf{x}) - B^*_k(\mathbf{x})\right| > T_p \\ \\ 0 \;\; otherwise \end{cases} \tag{6}$$

In order to consolidate the blobs to be detected, a 3x3 morphological closing is applied followed by a 3x3 morphological opening.

3.3.3. Rejection Filter

Blobs corresponding to neighbor (8-connectivity) moving pixels are built. Both images containing moving pixels, D_{H_K} and D_{B_K}, are used for this computation. For each blob b a movement density MD_b is defined as:

$$MD_b = \frac{\sum_{x \in b} \left| I_k(x) - I_{k-1}(x) \right|}{Area(b)} \tag{7}$$

MD_b measures the average change in the last frame for the blob b. *Ghosts*, defined as groups of pixel that are not moving, but detected as moving because they were part of a MVO in the past, should have a low MD_b, while the MVOs should have a large MD_b. Then, blobs with a small area (area ≤ T_b), with a large area (area ≥ T_s) and blobs with a small movement density ($MD_b \leq T_d$) are considered miss detections and discarded.

3.4 Detection Estimation

Target objects (the MVOs) are tracked by keeping a list with the state of each of them. The state of a given target u includes the position of the center of mass (x_u,y_u), the speed (Vx_u,Vy_u), the area (a_u), and the growing speed (Va_u). For each received movement blob, the area and the center of mass are calculated. These variables are used as sensor observations, and integrated across the different motion detections (the ones coming from D_{H_K} and D_{B_K}), and over time using a first order Kalman Filter (Gong, 2000). This process includes six stages: *prediction*, *observation-target matching*, *update*, *detection of new targets*, *targets elimination,* and *targets merge.* After those six stages, the target state list, whose values are estimated by the Kalman Filter, corresponds to the final motion detections (Det_k*).

Prediction. Using a first order cinematic model, it predicts the state vector for each target based on the last estimated state, and projects the error covariance ahead.

Observation-Target matching. In order to update the targets, it is imperative identifying which observation affects each target. For each observation-target combination, a confidence value is calculated as the probability function given by the Kalman filter for the target evaluated on the observation. For each target, all observations with a confidence value over a threshold T_{t1} are associated with the target. If an observation does not have any associated target, then it is considered as a new target candidate and passed to the *Detection of new targets* stage. For each observation associated with a target, speeds (spatial speed: Vx and Vy, and growing speed Va) are estimated. This estimation is performed using the difference between the target state before the prediction and the observation state, divided by the elapsed time since last prediction.

Update. It computes the Kalman gain, updates the state vector for each target using their associated observations, and updates the error covariance. The targets without associated observations are not updated.

Detection of new targets. All observations without an associated target are considered as new target candidates. Their spatial speed is calculated as the distance from the image border, in the opposite direction of the image center, divided by the elapsed time since the last frame, and their growing speed is set to zero.

Targets elimination. Targets without associated detections in the last two frames are considered as disappeared MVO, and eliminated from the target list.
Targets merge. For each target-target combination, two confidence values are calculated as the probability function given by the Kalman filter for one target evaluated on the other target state. If any of this confidence values is over a threshold T_j, then the two targets are considered equivalents, and the target with the largest covariance (measured as the Euclidian norm of the covariance matrix) is eliminated from the target list.

3.5 Background Update

The background model is computed as the weighted average of a sequence of previous frames and the previously computed background:

$$B_k(\mathbf{x}) = \begin{cases} B_k^*(\mathbf{x}) & if\,(\mathbf{x}) \in DET_k^* \\ \\ \alpha I_k(\mathbf{x}) + (1-\alpha) B_k^*(\mathbf{x}) & otherwise \end{cases} \tag{8}$$

4. Experiments

For the experiments, an AIBO robot using the motion software of the *UChile Kiltros* AIBO soccer team (Ruiz-del-Solar, 2007), configured for allowing just head movements was used. The algorithm runs in the robot in real-time. For analysis purposes, two video sequences were employed. In both, the robot moves its head in an ellipsoidal way, keeping the roll angle of the camera approximately aligned with the horizon. While the robot is moving its head, a ball is moving, once in the same direction of the camera movement, and once in the opposite direction. In figure 2, the different stages of the algorithm while processing the frame 28 of the first video sequence are shown.
The here-proposed motion detection algorithm can be enhanced using additional object information, such as color when detecting moving balls. Thus, in the *Rejection Filter* module, it was implemented a ball color filter applied to the blobs. This color filter uses the average U-V values (YUV color space) from each blob for filtering. If the Euclidean distance between the U-V average value of a blob and the ball U-V value (model) is larger than a threshold *Tc*, then the blob is discarded. This filter decreases significantly the number of false positives errors. It should be stressed that this filter can be applied only after the blobs have been already detected.
The first/second video sequence was 33/37 frames long. 11/14 frames contain a moving ball, but the first appearance of the ball cannot be detected because there is no way to know if the ball is moving or if it is stopped. Thus the relevant information are only 9/12 frames with moving ball, 6/9 of them were successfully detected, which correspond to a successful detections rate of 67%/75%. In table 1, consolidated statistics of the analysis of these video sequences, with and without using the color filter are shown.

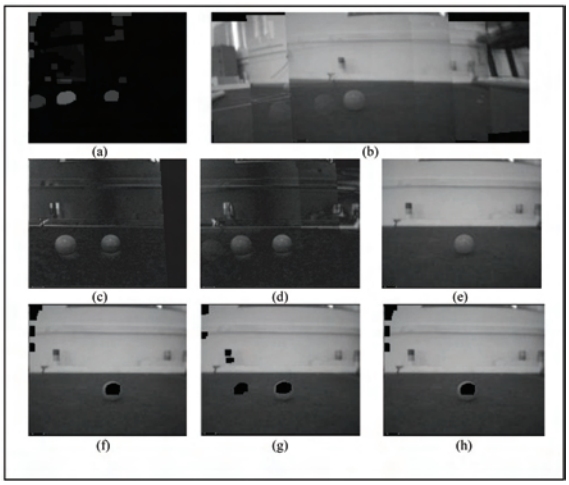


Fig. 2. Process stages at frame 28 in video sequence 1. (a) Motion history representation M_{Hk}. (b) Background model B_k^*. (c) Motion history error image (error intensity values are enlarged for better visualization). (d) Background subtraction error (error intensity values are enlarged for better visualization). (e) Current Image I_k. (f) Motion history detection D_{Hk} in black overlapped to the current image. (g) Background subtraction detection D_{Bk} in black overlapped to the current image. (h) Final detections DET_k*, generated by the Kalman Filter

Sequence number	1		2	
Number of frames	33		37	
Frames with a detectable moving ball	9		12	
Ball color filter	Off	On	Off	On
Frames with successful moving ball detection	6 (67%)	6 (67%)	9 (75%)	6 (50%)
Detections corresponding to moving balls	7	6	9	6
Detections corresponding to ghosts	9	0	7	0
Detections corresponding to other moving objects	10	0	8	0
False detections (excluding ghosts)	296	2	265	5
Total number of detections	322	8	289	11
False detections average by frame, excluding ghosts	8,97	0,06	7.16	0,14

Table 1. Analysis of detections in the video sequence 1 and 2

For a better understanding of the algorithm in figure 3, six frames of the video sequence 1are shown. There is shown the first frame (a), two frames with a ball moving in the opposite direction of the camera movement (b) and (c), and three frames with a ball moving in the same direction of the camera movement (d), (e) and (f). In those frames, there are marks with boxes for the detections, which color represents the detection kind: red for successful detections, blue for fake detections and black for ghost detections. Solid line boxes represent detections after the color filter application; in contrast, dotted line boxes represent detections eliminated by the color filter. In figure 4, six frames of the video sequence 2 are shown. There are shown three frames with a ball moving in the same direction of the camera movement (a), (b) and (c), in those frames the tracking is performed successfully; and three frames with a ball moving in the opposite direction of the camera movement (d), (e) and (f), where the movement is detected partially.

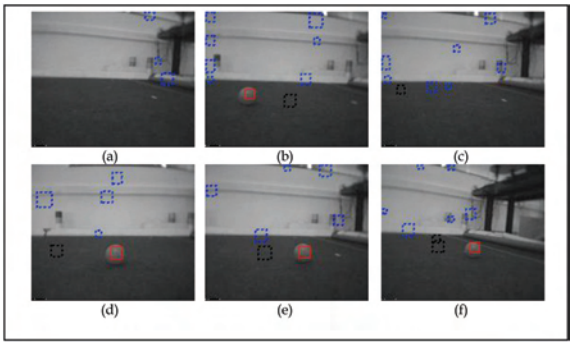


Fig. 3. Six example frames in video sequence 1. Solid line rectangles represent detections after the color filter application; in contrast, dotted line rectangles represent detections eliminated by the color filter. Red boxes denote correct detections; black boxes, ghost detections; and blue boxes, false detections. (a) Frame 1: there is no ball (all movement analysis algorithms need more than 1 frame for making comparisons). (b) Frame 16: the ball and his ghost are detected. (c) Frame 17: the ghost of a disappeared ball is detected. (d) Frame 28: tracking a ball. (e) Frame 29: tracking a ball. (f) Frame 30: tracking a ball

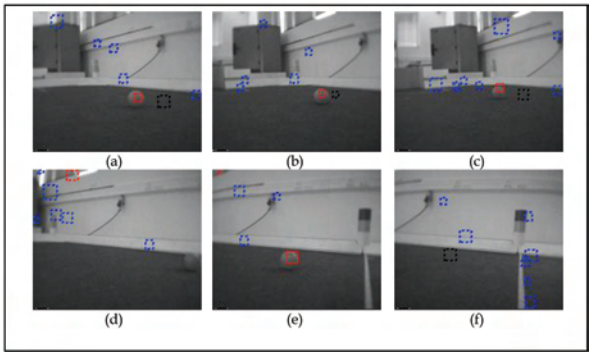


Fig. 4. Six example frames in video sequence 2. Solid line rectangles represent detections after the color filter application; in contrast, dotted line rectangles represent detections eliminated by the color filter. Red boxes denote correct detections; black boxes, ghost detections; and blue boxes, false detections. (a) Frame 13. (b) Frame 14. (c) Frame 15. (d) Frame 24. (e) Frame 25. (f) Frame 26. From frame 13 to frame 15 there is a successful ball tracking. In frames 24, 25 and 26 there is a fast appearance of a ball, in this frames the ball is detected partially

5. Conclusion

In this work we proposed an approach for motion detection and object tracking with a moving camera, with application to robot soccer. In the proposed system, the camera movements are compensated in software by aligning the current frame and the background. Results of the motion detection and tracking of objects in real-world video sequences using

the proposed approach were shown. The system operates in real-time and the relevant moving objects, the ball in this case, are detected and tracked.
In the described system, the images' alignment is achieved using the robust inverse compositional algorithm, which requires an initial estimation of the warp. This initial estimation is obtained from the measured camera motion (robot joints' information). We have developed an improved images' alignment module based on local features (SIFT features), which does not require any information of the camera motion (Vallejos, 2007). This module runs at 5 frames per second. Currently, we are working on reducing the processing time of the module, and in its integration into our motion detection and object tracking system.

7. References

Baker, S. & Matthews, I. (2001). Equivalence and Efficiency of Image Alignment Algorithms, *Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition*, pp I-1090- I-1097, ISBN 0-7695-1272-0.

Baker, S. & Matthews, I. (2002). *Lucas-Kanade 20 years on: A unifying framework: Part 1*, Technical Report CMU-RI-TR-02-16, Carnegie Mellon University, Robotics Institute.

Cucchiara, R.; Grana, C. & Prati, A. (2002). Detecting Moving Objects and their Shadows - An evaluation with the PETS2002 Dataset, *Proceedings of the 3rd IEEE Int. Workshop on PETS*, pp 18- 25, Copenhagen, June 2002.

Ferryman, J. (2002). *Proceedings of the 3rd IEEE Int. Workshop on PETS*, Copenhagen, Denmark, June 2002.

Ferryman, J. (2006). *Proceedings of the 9th IEEE Int. Workshop on PETS*, New York, USA, June 2006.

Gong, S.; McKenna, S. & Psarrou, A. (2000) *Dynamic Vision From Images to Face Recognition*, Imperial College Press, ISBN 1860941818.

Ishikawa, T.; Matthews, I. & Baker, S. (2002). *Efficient Image Alignment with Outlier Rejection*, Technical Report CMU-RI-TR-02-27, Carnegie Mellon University, Robotics Institute, October 2002.

Lucas, B. & Kanade, T. (1981). An Iterative image registration technique with an application to stereo vision, *Proceedings of the International Joint Conference on Artificial Intelligence*, pp 674-679.

Piater, J. & Crowley, J. (2001). Multi-Modal Tracking of Interacting Targets using Gaussian Approximations, *Proceedings of the 2nd IEEE Int. Workshop on PETS*, Hawaii, USA, Dec. 2001.

Ruiz-del-Solar, J.; Guerrero, P.; Arenas, M.; Loncomilla, P.; Fredes, J.; Díaz, G.; Palma-Amestoy, R.; Vallejos, P.; Valenzuela, M.; Dodds, R. & Monasterio, D. (2007). UChile Kiltros 2007 Team Description Paper, *Proceedings of the 2007 RoboCup Symposium* (CD Proceedings).

Russ, J. (1995). *The Image Processing Handbook*, Second Edition, CRC Press inc, Boca Raton, FL, USA, ISBN 0-8493-2516-1.

Vallejos, P. (2007). *Detección y Seguimiento de Personas y Objetos en Movimiento usando Cámaras Móviles*, Master degree thesis, Universidad de Chile, 2007 (in Spanish).

18

Comprehensive Omni-Directional Soccer Player Robots

Mehdi DaneshPanah[1], Amir Abdollahi[2], Hossein Ostadi[3] and Hooman Aghaebrahimi Samani[4]

[1] *Dept. of Electrical and Computer Eng., University of Connecticut, CT USA*

[2] *Dept. of Mechanical Eng., Yamaguchi University, Ube Japan*

[3] *Dept. of Mechanical Eng., Universitat Polytécnica de Catalunya, Barcelona Spain*

[4] *Dept. of Electrical and Computer Eng., National University of Singapore Singapore*

1. Introduction

RoboCup competitions were first proposed by Mackworth in 1993. The main goal of this scientific competition is to exploit, improve and integrate the methods and techniques from robotics, machine vision and artificial intelligence (AI) disciplines to create an autonomous team of soccer playing robots. At the time of preparing this chapter, RoboCup is organized in several different leagues including soccer simulator (2D and 3D), small size, middle size and legged robots, (Kitano, 1997a, Kitano et. al, 1997. Kitano, et. al, 1998). These leagues are designed to break down the problem into several venues so that the challenges can be addressed efficiently. Robots in middle-size league should operate autonomously only with local resources including local sensors, batteries and local vision. Each team can have a maximum number of four robots with a maximum footprint of 2000cm^2. They can communicate with each other through a central computer via a radio link. The rules in the competition are the same as the international soccer rules as far as they are practical for robots (Kitano, 1997b).

Recently, most conventional mobile robots have used a wheeled mechanism. Such mechanism consists of two independent driving wheels responsible for all needed robot motions (front-steering and rear-wheel driving mechanism). Motion restriction is a major problem in the use of such mechanism in mobile robots. There are also other suggested mechanisms such as universal wheel mechanisms, ball wheel, crawler and offset steered wheel mechanisms (Watanabe, 1998, West, 1992, Nakano, 1993).

Omni directional mobile robots have been popularly employed in several applications especially in soccer player robots considered in Robocup competitions. Such robots can reach to any position with no rotation through a straight line, so they can provide high mobility with no motion restriction. In these robots, providing high speed with an acceptable error is a very important factor in the competitive and dynamic environments (see Fig. 1). An omni directional robot can respond more quickly and it would be capable for more sophisticated behaviors such as ball passing or goal keeping.

Control and self- localization of omni directional mobile robots are important issues and different teams in the Robocup competitions have used different techniques to tackle it. Setting the PID controllers coefficients heuristically with no prior estimation based on just trials and errors is generally very time consuming. On the other hand, solving the set of coupled differential equations is very complicated and may not be practical for a real time control (Kalmar-Nagy, et. al, 2002). Some teams decoupled the mathematical model of the system while the others use fault tolerant control strategy for their systems (Jung, et. al, 2001). Real-time path generation based on the polynomial spline-interpolation with prediction of velocities of spline functions is also proposed and used (Paromatchik, et. al, 1994). A fuzzy model of the omni directional robot control was studied analytically in (Watanabe, 1998). In this chapter, we propose a calibration method for the robot controller in dynamic environments based on a simple motion to estimate initial values for the PID coefficients. For reliable and robust control, we propose a combined feedback from the odometry and vision mechanisms.

We show that the accuracy of the vision based self localization is not uniform everywhere in the field. Thus, we apply the sensitivity analysis method to evaluate the performance of the vision self-localization for feedback generation and we suggest techniques to improve the accuracy of the location feedback. By combining these strategies and utilizing the comprehensive omni directional robot (Samani, et. al, 2004), *Persia* Middle Size team won the 1st place in world Robocup technical challenge competitions in Portugal 2004 and 3rd place in Italy 2003.

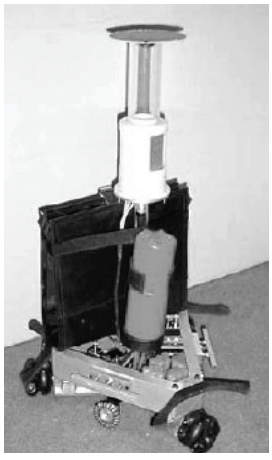


Fig. 1. A comprehensive omni directional robot having omni directional vision, motion and kicking systems

Another significant subject in the robots' soccer competition is artificial intelligence (AI), since soccer needs cooperative behavior and coordination between agents which need some form of intelligence. In this chapter, we propose a comprehensive AI architecture for this purpose in three well defined, distinct layers which provides the team with fully dynamic and flexible team work with little computational or architectural complexity cost.

This chapter is organized as follow. Sections 2, 3 and 4 describe the kinematics, dynamics and control of the robot respectively. Feedback generation and self localization using both omni-vision and odometry is presented in section 5. The omni directional kicking mechanism is described in section 6. The artificial intelligence algorithms in our robot are explained in details in section 7 and the experimental results is explained in section 8. We finally conclude this chapter in section 9.

2. Robot Kinematics

2.1 Omni Directional Wheels and Robot Chassis

Omni directional robots usually use special wheels. These wheels are known as omni directional poly roller wheel. The most common wheels consist of six spindle like rollers which can freely rotate about their longitudinal axis (Fig. 2a) (Asama, 1995, Watanabe, 1998).

(a) (b)

Fig. 2. (a) Omni directional poly-roller wheel, (b) Omni directional small-roller wheel

The surface shape and size of the poly rollers are designed such that all six rollers form a complete circle and generate a low vibration while rotating similar to a normal wheel. However, since the wheel has a low surface contact on the field compared with a normal wheel, the slippage is more severe. Due to the low vibration, this wheel is suitable for the actuating mechanism and is connected to DC motors while it is not proper for feedback generation considering its slippage. In order to avoid the slippage effects of this wheel, we design another type of omni directional wheel which consist of small cylindrical rollers mounted on the main body of the wheel in a feedback mechanism (Fig. 2b). As shown in Fig. 2b, this wheel covers a polygonal shape, so the wheel vibration is considerable. In fact, it should be mounted on the system with a flexible structure such as a flat spring (Fig. 3). Shaft encoders are mounted on these wheels.

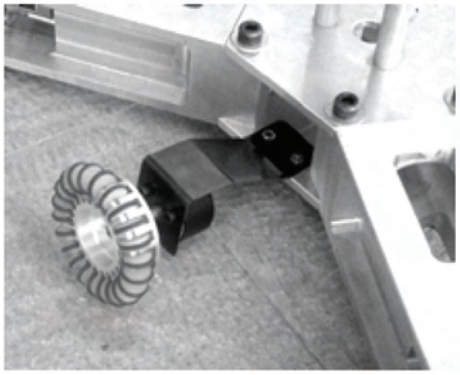


Fig. 3. Omni directional small-roller wheel connected to the body via a flat spring

A robot with three omni directional wheels can essentially follow any 2D trajectory. Our robot structure includes three big black omni-directional wheels for motion system (Fig. 5a), and three small free wheels on which shaft encoders are mounted as feedback mechanism (Fig. 4b) (Asama, et. al, 1995, Watanabe, 1998).

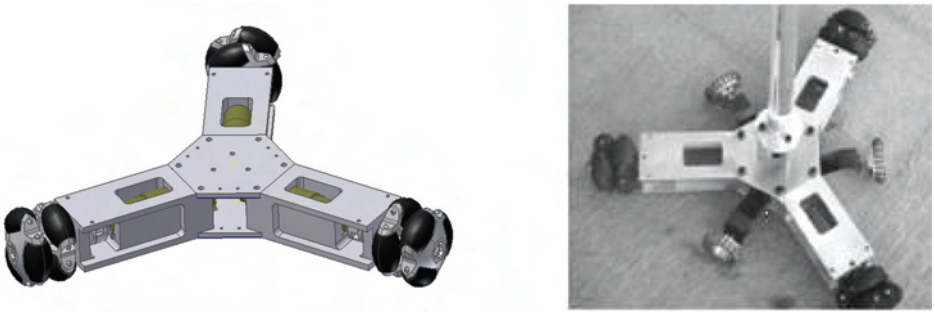


Fig. 4. (a) Three black omni directional poly-roller wheels act as actuators, (b) three free omni directional small-roller wheels used in a feedback mechanism

2.2 Kinematics Equations

Using omni directional wheels, the schematic view of robot kinematics can be shown as follows (Fig. 5) (Kalmar-Nagy, et. all, 2002).

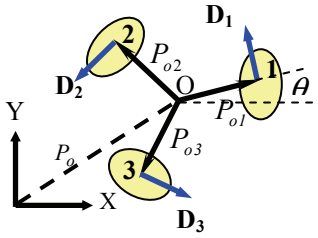


Fig. 5. Robot kinematics diagram

In the figure above, O is the robot center of mass, $\mathbf{P}_o$ is defined as the vector connecting O to the origin and $\boldsymbol{D}_i$ is the drive direction vector of each wheel. Using unitary rotation matrix, $\mathbf{R}(\theta)$ is defined as:

$$\mathbf{R}(\theta)=\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix} \tag{1}$$

The positions vectors $\mathbf{P}_{o1},\ldots,\mathbf{P}_{o3}$ with respect to the local coordinates centered at the robot center of mass are given as:

$$\mathbf{P}_{o1}=L\begin{bmatrix}1\\0\end{bmatrix} \qquad \mathbf{P}_{o2}=\mathbf{R}(\frac{2\pi}{3})\times\mathbf{P}_{o1}=\frac{L}{2}\begin{bmatrix}-1\\\sqrt{3}\end{bmatrix} \qquad \mathbf{P}_{o3}=\mathbf{R}(\frac{4\pi}{3})\times\mathbf{P}_{o1}=-\frac{L}{2}\begin{bmatrix}1\\\sqrt{3}\end{bmatrix} \tag{2}$$

where L is the distance of wheels from the robot center of mass (O). The drive directions can be obtained by:

$$\mathbf{D}_i=\frac{1}{L}\mathbf{R}(\theta)\times\mathbf{P}_{oi} \tag{3}$$

$$\mathbf{D}_i=\begin{bmatrix}0\\1\end{bmatrix} \qquad \mathbf{D}_2=\begin{bmatrix}1\\0\end{bmatrix} \qquad \mathbf{D}_3=\frac{1}{2}\begin{bmatrix}\sqrt{3}\\-1\end{bmatrix} \tag{4}$$

Using the above notations, the wheel position and velocity vectors can be expressed with the use of rotation matrix $\mathbf{R}(\theta)$ as:

$$\mathbf{R}_i=\mathbf{P}_o+\mathbf{R}(\theta)\times\mathbf{P}_{oi} \tag{5}$$

$$\mathbf{V}_i=\dot{\mathbf{P}}_o+\dot{\mathbf{R}}(\theta)\times\mathbf{P}_{oi} \tag{6}$$

The vector $\mathbf{P}_o=\begin{bmatrix}x & y\end{bmatrix}^T$ is the position of the center of mass with respect to Cartesian coordinates. The angular velocity of each wheel can be expressed as:

$$\phi_i=\frac{1}{r}\mathbf{V}_i^T\times\mathbf{R}(\theta)\times\mathbf{D}_i \tag{7}$$

where r is the radius of odometry wheels. Substituting for $\mathbf{V}_i$ from equation (6) yields:

$$\phi_i=\frac{1}{r}\left[\mathbf{P}_o^T\times\mathbf{R}(\theta)\times\mathbf{D}_i+\mathbf{P}_{oi}\times\dot{\mathbf{R}}(\theta)^T\times\mathbf{R}(\theta)\times\mathbf{D}_i\right] \tag{8}$$

Note that the second term in the right hand side is the tangential velocity of the wheel. This tangential velocity could be also written as:

$$L\dot{\theta}=\mathbf{P}_{oi}^T\times\dot{\mathbf{R}}(\theta)^T\times\mathbf{R}(\theta)\times\mathbf{D}_i \tag{9}$$

From the kinematics model of the robot, it is clear that the wheel velocity is a function of linear and angular velocities of robot center of mass, i.e.:

$$\begin{Bmatrix}\dot{\phi}_1\\\dot{\phi}_2\\\dot{\phi}_3\end{Bmatrix}=\frac{1}{r}\begin{bmatrix}-\sin\theta & \cos\theta & L\\ -\sin(\frac{\pi}{3}-\theta) & -\cos(\frac{\pi}{3}-\theta) & L\\ \sin(\frac{\pi}{3}+\theta) & -\cos(\frac{\pi}{3}+\theta) & L\end{bmatrix}\begin{Bmatrix}\dot{x}\\\dot{y}\\\dot{\theta}\end{Bmatrix} \tag{10}$$

Or in a equivalent vector form:

$$\dot{\boldsymbol{\varphi}} = \frac{1}{r}\mathbf{W} \times \dot{\mathbf{S}} \tag{11}$$

where L is the distance of wheels from the robot center of gravity (O) and r is the main wheel radius.

3. Robot Dynamics

Linear and angular momentum balance for the robot can be written as:

$$\sum_{i=1}^{3} f_i \mathbf{R}(\theta) \times \mathbf{D}_i = m\ddot{\mathbf{P}}_O \qquad L\sum_{i=1}^{3} f_i = J\ddot{\theta} \tag{12}$$

where $\ddot{\mathbf{P}}_o$ is the acceleration vector, f_i is the magnitude of the force produced by the ith motor, m is the mass of the robot and J is its moment of inertia about its center of gravity. Assuming no-slip condition, the force generated by a DC motor can be written as:

$$\mathbf{f} = \alpha\mathbf{U} + \beta\mathbf{V} \tag{13}$$

where, $\mathbf{V} = \{V_i(t), i = 1,2,3\}$ is the velocity of each wheel. The constants α and β are motor characteristic coefficients and can be determined either from experiments or from motor catalogue. Note that $\mathbf{U} = \{U_i(t), i = 1,2,3\}$ is the voltage applied by supplier to the DC motors. Substituting equation (13) into equation (12) yields:

$$\sum_{i=1}^{3} (\alpha U_i - \beta V_i)\mathbf{R}(\theta)\mathbf{D}_i = m\ddot{\mathbf{p}}_o \qquad L\sum_{i=1}^{3} (\alpha U_i - \beta V_i) = J\ddot{\theta} \tag{14}$$

This system of differential equations can be written in the matrix form as:

$$\begin{bmatrix} m\ddot{x} \\ m\ddot{y} \\ J\ddot{\theta} \end{bmatrix} = \alpha\mathbf{P}(\theta)\mathbf{U} - \frac{3\beta}{2}\begin{bmatrix} \dot{x} \\ \dot{y} \\ 2L^2\dot{\theta} \end{bmatrix} \tag{15}$$

$$\mathbf{P}(\theta) = \begin{pmatrix} -\sin\theta & -\sin(\frac{\pi}{3} - \theta) & \sin(\frac{\pi}{3} + \theta) \\ \cos\theta & -\cos(\frac{\pi}{3} - \theta) & -\cos(\frac{\pi}{3} + \theta) \\ L & L & L \end{pmatrix} \tag{16}$$

and

$$\mathbf{U} = \begin{bmatrix} U_1(t) & U_2(t) & U_3(t) \end{bmatrix}^T . \tag{17}$$

4. Robot Controller

PID controllers are used for controlling the robot position and orientation. In this chapter we propose a controller that is robust enough for controlling a soccer player robot (Jung, et. al,

2001). For obtaining the PID controller coefficients, one needs to first obtain the whole transfer functions of the system and then solve it accordingly. Since the equations, even if derived properly, are a set of coupled nonlinear differential equations, it is very difficult to solve them. Even though we derive and solve the equations, the results (PID coefficients) are not reliable since they depend on many other parameters such as ground surface friction factor, characteristics of batteries and so on. So the equations will be decoupled with these assumptions:

1. Omni directional mechanism is a mechanism which can reach to any position with no rotation ($\theta = 0$) through a straight line, this specification help the robot to reach the desired position in the least time compared with a 2 wheel mechanism. It is also true that every curve can be divided into some straight lines and at the end of each line the robot did not need to rotate to follow the next line.
2. Whenever it is necessary to rotate (for example when the robot kicker should be in a particular position) the robot rotates while it is moving in a straight line to reach the right position. This can be regarded as a pure rotation in addition to the first assumption.
3. The pure rotation in our robot is obtained by applying equal voltages to each motor.
4. In order to find PID coefficients for the robot position controller, moving through a straight line is very similar to move through an axis like X Direction (Y=0 in equation 15) The voltage obtained from position controller is then added to the voltage found by orientation controller.

Based on the above assumption, the robot position is not depend on θ, so for position control, we assume that θ=0. In the cases where rotation is required, the voltage obtained from orientation control for each motor equally added to the position controller output.

For PID tuning in position controller, a simple movement was considered, i.e., θ=0, Y=0 (or a constant value) in equation 15. Similarly, for orientation control, a pure rotation is considered, i.e., X=0 (or constant), Y =0 (or constant).

4.1 Position Controller Architecture

Figure 6 illustrates the overall block diagram of the system. As it is clear from Fig. 6, the omni directional robot control loop contains a PID and PD controller (with the transfer function $\boldsymbol{H}_{\mathrm{PID}}$), a plant transfer function ($\boldsymbol{H}_{\mathrm{P}}$ which is obtained from the system dynamics) and a self-localization transfer function (as a feedback function that only senses the robot's position).

A noise node, $\boldsymbol{N}$, is also included that has an additive effect on the system position input. The input of the system is considered to be a step function and the output is the robot position and orientation.

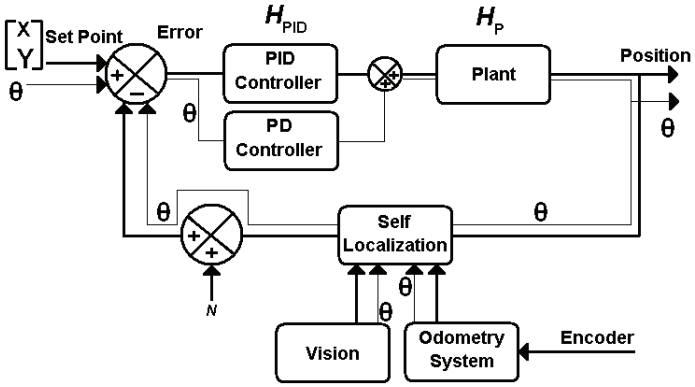


Fig. 6. Control diagram of the omni directional robot

4.1.1 H_{PID} Transfer Function

H_{PID} can be written in a general form as follows:

$$H_{PID}(s) = K_P + \frac{K_I}{s} + K_D s \tag{18}$$

Here k_P, k_I and k_D are proportional, integral and derivate gains respectively.

Experiments showed that the system overall performance is satisfactory and thus this type of controller is robust enough for controlling a soccer player robot (Kalmar-Nagy, et. al, 2002).

4.1.2 H_P Transfer Function

As it is mentioned in section 4, two simple motions were considered and solved, namely straight line motion of the robot in x direction and pure rotation about the z axis. The former means that one motor is turned off and the other two are turned on with the same angular velocity while the latter means that all three motors are turning with the same angular velocities.

We will study the orientation separately in section 4.2. The output voltage from the orientation controller (w) is then added to the voltage obtained from the position controller output (v_i). The assumption of summing up these voltages is valid while motors are operating in their linear regions. In order to apply the straight line motion, one can consider equation (15) with: $\theta = 0$ and $\dot{\varphi}_1 = \dot{y} = \dot{\theta} = \ddot{\theta} = 0$ and $\dot{\varphi}_2 = -\dot{\varphi}_3$. Equation (15) reduces then to:

$$m\ddot{x} + 3\frac{\beta \dot{x}}{2} = \sqrt{3}\alpha U_2 \tag{19}$$

Applying Laplace transfer function to equation (19) with the initial conditions: $X(0) = 0, \dot{X}(0) = 0$, one obtains:

$$H_P(s) = \frac{X(s)}{U_2(s)} = \frac{\sqrt{3}\alpha}{ms^2 + \frac{3\beta s}{2}} \tag{20}$$

It should be noted that for ideal case (in absence of noise) the complete transfer function for position control will be obtained as follows ($H_{Self\ Localization} = 1$):

$$H_{Total}(s) = \frac{H_{PID}H_P}{1 + H_{PID}H_P} = \frac{\sqrt{3}\alpha(k_D s^2 + k_P s + k_I)}{ms^3 + (\frac{3\beta}{2} + \sqrt{3}\alpha k_D)s^2 + \sqrt{3}\alpha k_p s + \sqrt{3}\alpha k_I} \tag{21}$$

Figure 7 shows the step and noise response curves with various k_p, k_I, k_D values.

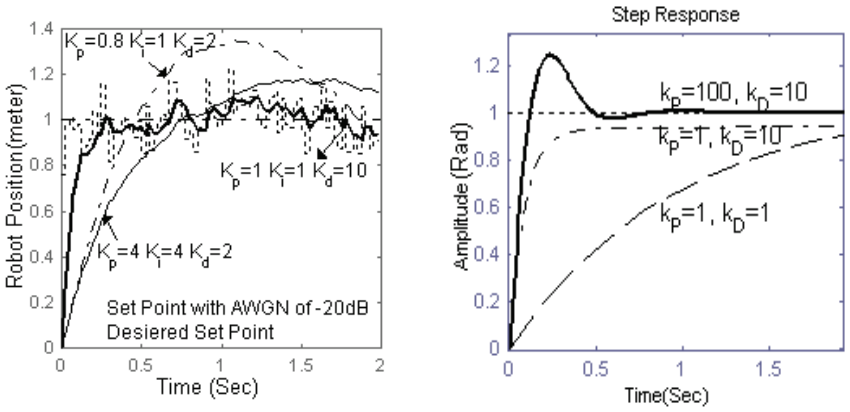


Fig. 7. (a) System step in addition noise, (b) Step response of the PD response for different values of k_p, k_I, k_D controller for orientation control

The following observations can be deducted. The dotted line in Fig. 7a shows a step function with an added noise of Gaussian distribution. In this curve the noise is applied to the system every 40 micro seconds due to the robot processing time. The mean value and standard deviation of the noise are 0. In Fig. 7a, by increasing k_P, k_I (dash-dotted line & solid line), the system response delay time will increase too. Also, there are some overshoots in these curves. However, by increasing the k_D value, this effect will reduce drastically. In order to find optimum values for the PID coefficients, different combinations of the parameters were selected and examined. Eventually, the proper PID coefficients were obtained for our system as $K_p = 1, K_I = 1, K_d = 10$. The response of the system for these values is depicted by thick solid line in Fig. 7a.

4.2 Orientation Control

It is also necessary to apply a controller for the robot rotation control. Assume that the robot only rotates about its vertical axis, i.e., Z-axis. We then derive: $\dot{\varphi}_1 = \dot{\varphi}_2 = \dot{\varphi}_3, U_1 = U_2 = U_3$.

Substituting these values into the third equation in relation (15) leads us to:

$$J\ddot{\theta} + 3\beta L^2\dot{\theta} = 3LU_1 \tag{22}$$

Applying Laplace transform to the above equation yields

$$\frac{\theta(s)}{U(s)} = \frac{3L}{Js^2 + 3\beta L^2 s} \tag{23}$$

and considering a PD controller for this case, we obtain the total transfer function for orientation control as:

$$H_{Total}(s) = \frac{3L(k_P + k_D s)}{Js^2 + (3\beta L^2 + 3Lk_D)s + 3Lk_P}. \tag{24}$$

Figure 7b shows the step response of the control system. Since the experience showed that residual error for orientation control is not of great importance in our scenario, a PD controller will result in desired system response. Therefore, there is no need to apply the Integrator controller for the orientation or robot.

The optimum parameters for this case are k_P=100, and k_D=10. The step response for these parameter values is shown by solid line in Fig. 7b. The slight overshoot is desirable since we did not consider the effects of friction that damps the response in our model.

4.3 Final Robot Controller

In order to implement the position controller, the position error vector is determined as follows:

$$\mathbf{e} = \begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{25}$$

while the vector [x y] and [x′ y′] are the initial and the desired position of robot in the field respectively. Thus, the position control output can be written as:

$$\mathbf{V} = K_P\mathbf{e} + K_I\int\frac{d\mathbf{e}}{dt} + K_D\frac{d\mathbf{e}}{dt} \tag{26}$$

where **V** expresses the output of the position controller for the driving units whose components on each driving wheel are extracted with:

$$\mathrm{V}_i = \mathbf{V}^T \cdot \mathbf{D}_i \tag{27}$$

In this equation, vector $\mathbf{D}_i$ is the drive direction of ith motor. The output of the position controller for each motor is v_i. Considering the PD controller for orientation control, assuming that the head angle of the robot is δ and the desired head angle is Δ, the error angle is then determined as:

$$e_\Delta = \Delta - \delta \tag{28}$$

The orientation controller output is thus given by:

$$w = K_P e_\Delta + K_D\frac{de_\Delta}{dt} \tag{29}$$

The voltage from the orientation controller output is then added to the voltage obtained from the position control output. The final applicable voltages on the motors are then computed as:

$$u_i = v_i + w \tag{30}$$

This voltage is applied to each motor to reach the desired point. Since the system sensitive parts such as electronic control board, computer, batteries, etc., may be damaged by rapid rotation of the robot, we need to apply an upper and a lower threshold for the orientation controller output. Practically we set the threshold to be ± 10v.

The PID and PD coefficients were obtained from the two previous cases, used as a first estimation. This is due to the robot working conditions such as friction, gear boxes clearances and tolerances, motors mechanical time constant and etc that are not considered in modelling. The proper coefficients were then tuned experimentally in each competition. The results showed that for real cases, the maximum changes in the calculated values were around 10%. Therefore, such simplification is a good approximation for control model. Therefore, such simplification is a good approximation for control model.

5. Feedback Generation and Self-Localization

The position control method described in the former sections, calls for some form of position feedback in order to work properly. The performance of this feedback lies in its reliability, accuracy and real time computability. There have been plenty of algorithms and methods proposed by different researchers in the literature (Borenstein 1997, Olson 2001, et. al, Talhuri, et. al, 1993). Among them self- localization by visual information and odometry approach are dominant due to their special characteristics which will be discussed in the following paragraphs.

In this work, a compound novel method was developed and optimized for RoboCup MSL (Middle Size League) in which both visual and odometry information are used to ameliorate a real time, accurate and reliable method. Although optimized for soccer player robots, the self-localization method proposed here has enough modularity and flexibility to be applicable in most robotics applications involving self-localization.

Each of these complementary methods (vision/odometry self-localization) operates autonomously and has its own advantage and drawbacks in providing position feedback for robot control. For example, odometry method is known to have memory-based operation, accumulative error, low jitter, simplicity of implementation, cheap hardware, etc.

On the other hand, vision-based self- localization algorithms often have memoryless implementations (although there exists memory-based ones), no error accumulation, high jitter, relatively high computation complexity and expensive hardware. That is why amalgamating these methods can bring us to a global novel algorithm with good performance in vast and diverse conditions.

In the first subsection, the vision self-localization is explored in details and its different aspects are inspected. The second subsection describes a sensitivity analysis on the proposed method that demonstrates its performance in different locations in the field. Then using the evidence of sensitivity, slight modifications are presented for further robustness of the

overall algorithm. Next, odometry self- localization is proposed in brief and at last, the fusing algorithm of both visual and odometry outputs is described together with an Additive White Gaussian Noise (AWGN) model for the jitter.

5.1 Vision-Based Self-Localization Using Omni Vision Sensor

5.1.1 Hyperboloidal Omni-vision Sensor

The hyperboloidal mirror is a specific solution among the family of polynomial mirror shapes. These mirrors do not provide a central perspective projection, except for the hyperbolic one, but still guarantee a linear mapping between the angle of elevation φ and the radial distance from the center of the image plane ρ (Fig. 8) (Gaechter, S., 2001).

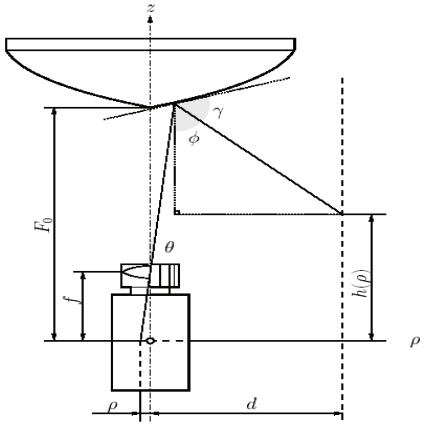


Fig. 8. Schematic diagram of the omni vision sensor

Another required specification is to guarantee a uniform resolution for the panoramic image. The resolution in the omni directional image increases with growing eccentricity, (ρ), when using a camera with an images of homogenous pixel density. (see Fig. 9)

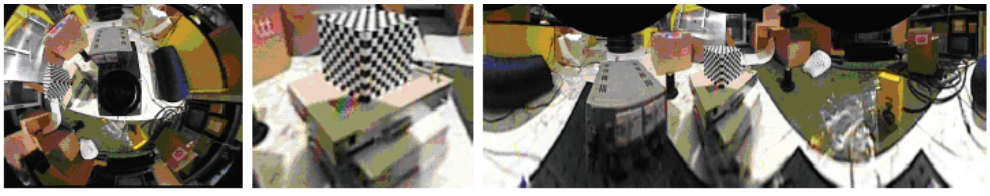


Fig. 9. (a) Input omni directional image, (b) perspective and (c) panorama image (Yachida, M., 1998)

Searching through the literature, we found that the following hyperbolic curve is commonly used for omni directional vision mirror (Ishiguro H., 1998):

$$\frac{x^2}{233.3} - \frac{y^2}{1135.7} = -1 \tag{31}$$

However, this equation is suitable for the mirror with large size and wide view. For our soccer player robot, we need an image with a diameter of 3.5m on the field. To design a compact mirror with wide view, the above curve scaled down by a factor of 2.5 to yields:

$$\frac{x^2}{233.3} - \frac{y^2}{1135.7} = -6.25 \tag{32}$$

Next, a special three stages process was considered for the mirror manufacturing:

1) Curve fabrication , 2) Polishing, 3) Coating

In the first stage the curve was fabricated on steel 2045 with CNC machining. Then, the work-piece was polished by a special process and finally Ni-P electroless plating was employed.

5.1.2 Vision-Based Self-Localization

Vision module was designed with several goals in mind; preparing spatial information of ball, opponents, team mates and self-localization raw data, exploited by self-localization module, are the key prospects. Our robot platforms were equipped with omni directional cameras (Talluri, et. al, 1993), with which the projection of the whole field area was available to the camera with a hyper-parabolic mirror described in section 5.1.1. Since the omni directional mirror introduces a map with very high non-linearity between pixel separation in the scene and the real physical distance (of such pixels) in the field itself, it is not reliable enough to develop algorithms which use distances as their input data with such mirrors.

Instead angles are preserved completely in a linear manner if the center of mirror and camera are aligned perfectly. Therefore, the algorithms with angles as their input data are more reliable and can perform more efficiently.

Our approach in vision-based self-localization is based on arcs. In basic geometry, there is a fact that having an angle of observation ω to a fixed and spatially known object in a 2D plane, can provide us with possible loci of the observation point. Actually, the points are located on the circumference of two circles (C_1, C_2). This simple idea is illustrated graphically in Fig. 10.

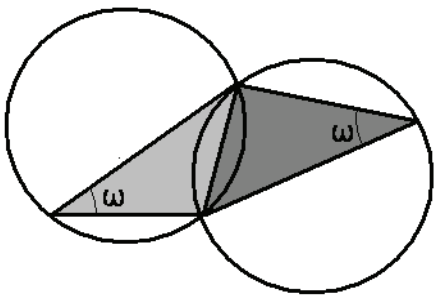


Fig. 10. Angle of observation ω and the two related arcs

The proposed algorithm here employs three different observation angles to constrain the unique position of observer (robot) in the field (assuming the ideal case with no visual noise). A good set of observation angles should have the following properties:

- Availability from different locations in the field.
- Extractable from visual data with low computational effort.
- The arcs resulting from these angles must be independent which means they should leave no location ambiguity at any point in the filed.
- The greater the angles magnitude, results in the lower sensitivity to visual noise (that will be discussed later).

Since goals are fixed landmarks and at least one of them has reasonable observation angle within the whole field area, their use for self-localization is popular in Robocup Middle Size League (Stroupe, et. al, 2002). An insightful examination through different combinations of possible observation angles for this purpose revealed that the following three angles are suitable regarding the above characteristics:

1. The observation angle from the robot itself to the nearest goal (α_{Goal}).
2. Angle between the center of the farthest goal and left side of the nearest one (β_{Goal}).
3. Angle between the center of the farthest goal and right side of the nearest one (γ_{Goal}).

These angles are depicted for an arbitrary location of the robot in the field in Fig. 11. Assume that the intersection points between Arc(j), and Arc(k) to be defined as:

$$P_i^{j,k} \qquad \begin{matrix} j,k \in \{1,1',2,2',3,3'\}; j \neq k \\ i \in \{1,2\} \end{matrix} \tag{33}$$

where the superscripts denote intersecting arcs and a subscript denotes the index of intersection. Note that the robot position is always on one point located on Arc (1).

First, a list of intersection points pairs are made using equation 33. In order to find the exact location of the robot, the Euclidian distances of different pairs of intersections are computed and the one which has zero norm is selected as the answer. In other words, there is only one point that is located on the intersection of three arcs and this point is the real position of the robot in ideal case, i.e., with no noise.

$$\underset{i,j,s,t}{Min}\left\{\left\|P_s^{1,i} - P_t^{1,j}\right\|\right\} \qquad \begin{matrix} i,j = 2,2',3,3' \qquad i \neq j \\ s,t = 1,2,3,... \end{matrix} \tag{34}$$

Considering imperfectness in visual information extraction, the intersection of Arc(1) with other two arcs may not coincide. In such a case, the set that yields the minimum distance introduces the possible position of the robot. The final position is simply computed by averaging over the neighboring intersection points that satisfy the above criteria (Fig. 11).

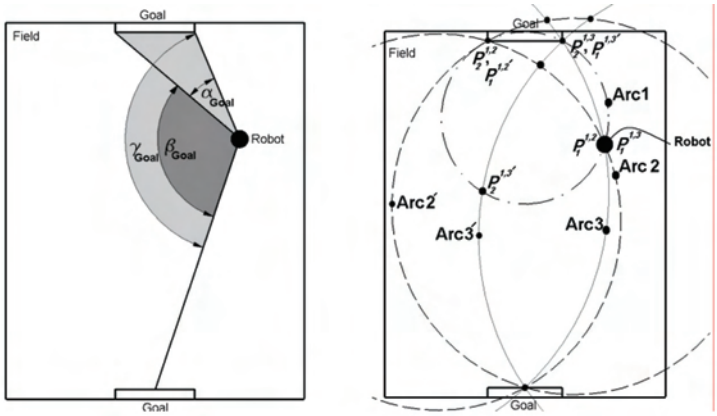


Fig. 11. (a) Angles observed by the robot, (b) The arcs and possible intersection positions

5.2 Sensitivity Analysis

The performance of vision-based self-localization method, developed in this work, relies on accurate visual information obtained from the vision module by means of image processing algorithms and techniques. Since goals are of two distinct colors in the play field (Yellow and Blue), the pixels representing them are distinguished by their position in RGB color space, and then their position and angle of observation are extracted with special region growing algorithms.

As mentioned before, although the angles are preserved linearly in the omni directional filed of view projected by the hyperbolic mirror, there is always the possibility that some error would exist in the detection procedure.

The sensitivity analysis of vision-based self-localization method reveals the regions in which the method is most sensitive to visual noise. The sensitivity of some performance characteristic y regarding parameter xi, is defined as the measure of its change Δy , resulting from a change Δx_i in the parameter xi. Suppose:

$$y = y(x_1, x_2, ..., x_n) \tag{35}$$

The variation of y is defined as:

$$dy = y\sum_{i=1}^{n}\left[\frac{x_i}{y}\frac{\partial y}{\partial x_i}\right]\frac{dx_i}{x_i} = y\sum_{i=1}^{n} S_{x_i}^{y}\frac{dx_i}{x_i} \tag{36}$$

where $S_{x_i}^{y}$ denotes the sensitivity of y with respect to parameter xi , and is computed as:

$$S_{x_i}^{y} = \frac{x_i}{y}\frac{\partial y}{\partial x_i} \tag{37}$$

Applying the above analysis on the proposed self- localization method in section 5.1 shows that in certain areas near the corner posts, the sensitivity increases and the accuracy of the method degrades drastically. Therefore, the proposed algorithm may be prone to severe errors in these regions (see Fig. 12). Since there are flags in the corner posts (that are of good

visibility and detectibly in that region by vision module), these landmarks are proper candidates for self-localization in these regions.

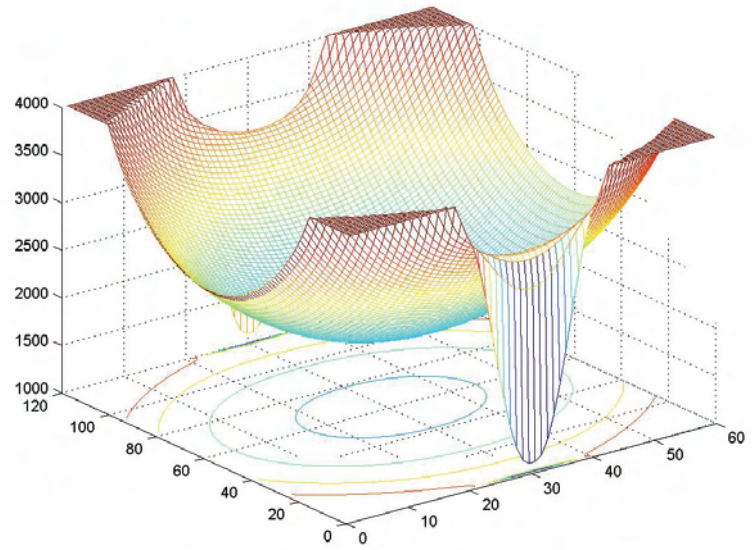


Fig. 12. Sensitivity of vision-based self-localization method at different location in the field

5.3. Localization Using Flags

For achieving better performance in the regions in which the sensitivity of the vision-based self-localization method is high, flags are used instead of goals to determine the position of robot. The procedure can be summarized as follow.

- By using visual data of goals and previous location of robot from its memory, the location of robot is roughly determined as Front-Left, Front-Right, Back-Left, Back-Right, where Front and Back show opponent and own side fields respectively.
- The nearest flag is then detected and the distance of robot to the flag base is approximated by a non-linear map constructed experimentally.
- Since the exact position of flag is known and the relative position of robot from flag is also available (R), then calculating the final robot position is a trivial task, i.e.:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} X_{FLAG} \\ Y_{FLAG} \end{bmatrix} + \begin{bmatrix} R\cos(\varphi) \\ R\sin(\varphi) \end{bmatrix} \tag{38}$$

Since the method of localization changes in these regions, and in order to avoid bouncing and confusion between these two methods, a hystersis strip (the grey area between two arcs near the flag in Fig. 13) is defined. Therefore, once the method changes to usage of flag (robot crosses the inner ring), it sticks to the new algorithm till the robots moves out of the outer ring in the hystersis strip.

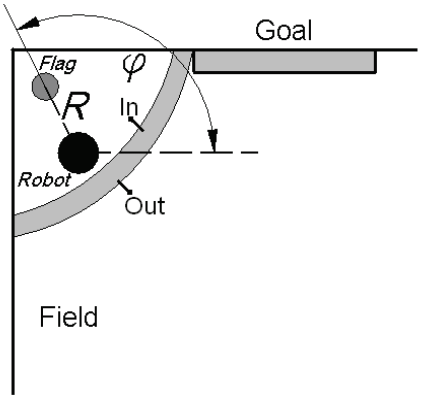


Fig. 13. The schematic view of the robot and flag near the corners (the grey strip is where the hystersis occurred)

5.4. Self-Localization Using Odometry

As it can be seen in Fig. 4b, three free rotating omni directional wheels are placed 60 degrees apart from the main driving wheels. These wheels are only passive, attached to three independent shaft encoders and have the role of odometry wheels. In reference (Samani, et. al, 2004) the direct method for position extraction using the data of shaft encoders is described. We only state the results here:

$$x = r\int \frac{1}{3\sin(\frac{\pi}{3})}\begin{bmatrix}(\cos(\frac{\pi}{3}+\theta)-\cos(\frac{\pi}{3}-\theta))\dot{\varphi}_1 + \\ (-\cos(\theta)-\cos(\frac{\pi}{3}+\theta))\dot{\varphi}_2+(\cos(\theta)+\cos(\frac{\pi}{3}-\theta))\dot{\varphi}_3\end{bmatrix}dt\ ,$$

$$y = r\int \frac{1}{3\sin(\frac{\pi}{3})}\begin{bmatrix}(\sin(\frac{\pi}{3}-\theta)-\sin(\frac{\pi}{3}+\theta))\dot{\varphi}_1 + \\ (-\sin(\theta)-\sin(\frac{\pi}{3}+\theta))\dot{\varphi}_2+(\sin(\theta)-\sin(\frac{\pi}{3}-\theta))\dot{\varphi}_3\end{bmatrix}dt\ , \quad (39)$$

$$\theta = r\int \frac{\dot{\varphi}_1+\dot{\varphi}_2+\dot{\varphi}_3}{3\mathrm{L}}\mathrm{dt}\ ,$$

where $[x \quad y \quad \theta]^T$ is vector containing the position and orientation of the robot. Further simplification of the third equation in (39) results in:

$$\theta = \frac{r}{3\mathrm{L}}\left[\int \dot{\varphi}_1\,\mathrm{dt}+\int \dot{\varphi}_2\,\mathrm{dt}+\int \dot{\varphi}_3\,\mathrm{dt}\right]=\frac{r}{3\mathrm{L}}(\varphi_1+\varphi_2+\varphi_3)\,. \quad (40)$$

5.5 Final Position Estimator

In order to obtain the final position estimation for the robot, both visual and odometry outputs must be fused in an appropriate fashion that would take advantage of each method to make flaws from the other one ineffective. For example, due to the inherent nature of vision-based self-localization, there is undesired jitter at its output, but, in return, odometry self-localization has smooth changes that can be used as a low-pass filter for vision-based self-localization results. Having this in mind, we came up with the following procedure for estimating the final position:

Step 1: Vision-based self-localization estimates the current position of the robot based on visual information from the current frame.

Step 2: Odometry utilizes the last computed position and determines the new position using the equation set (39).

Step 3: The position of robot is then computed as a weighted average of odometry and vision-based self-localization as:

$$\overline{P} = 0.9P_{Odometry} + 0.1P_{Vision} \tag{41}$$

Using these coefficients results in smoothing the variation (due to jitter in vision-based self localization) of final position estimation. The coefficients in equation (41) were obtained by carrying on tests on the robot position.

Step 4: The initial position for odometry in step 2 is then set to the computed robot position in the step 3 and the calculation continues for the next frame.

Since the outputs of both odometry and vision-based self- localization are prone to errors, and due to inherent random nature of these errors, a 2D Additive White Gaussian Noise (AWGN) is added to the output of a perfect self-localization block in the feedback path as shown in Fig. 6. The noise can be formulated as:

$$n_g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2})} \tag{42}$$

where σ_x and σ_y are noise deviation in X and Y directions, respectively. These values are then added to the position obtained from the self-localization module, (x0,y0), to obtain the probabilistic location of the robot, i.e. (x,y) as:

$$L(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-(\frac{(x-x_0)^2}{2\sigma_x^2}+\frac{(y-y_0)^2}{2\sigma_y^2})}. \tag{43}$$

6. Omni Directional Kicking System

For a soccer player robot, usually one direction is used for directing the ball to the goal or other destinations. Therefore, the ball handler and kicking mechanism is added in this direction which help the robot to get a suitable form for directing the ball.

Although the conventional mechanism can work for any soccer robot, it has some limitations which require additional movements for a particular behaviour. In other words, each robot has a specific head for kicking the ball and must adjust it to the proper direction during the game. These adjustments in the single head robot increase its rotation significantly and reduce its maneuverability. In order to minimize such rotations, we use two extra kicking mechanisms to form an omni directional kicking system. The position of these kicking systems is shown in Fig. 14a. As it can be seen from the figure, each kicker is assembled between two omni directional wheels and forms a system with three heads in 0°, 120°and 240° angles. Three types of soccer player robot in the form of 10 teams, which participated in Robocup, were examined in order to assess the rotation rate of each type.

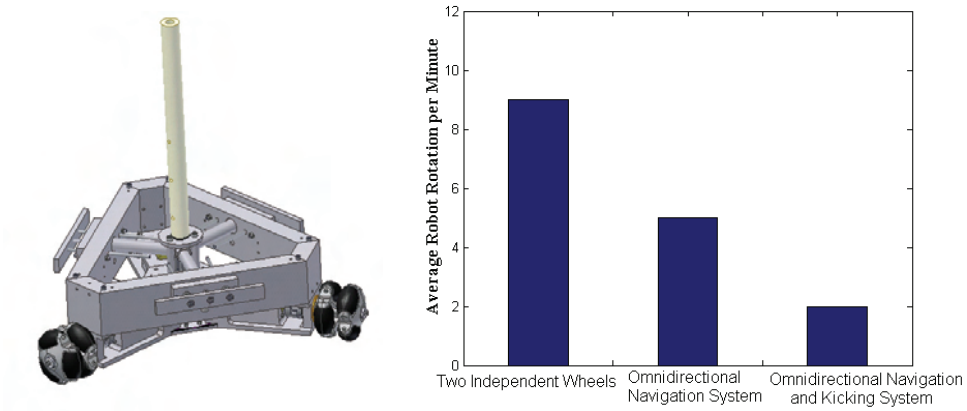


Fig. 14 (a) Omni directional kicking system of the robot, (b) Average rotation rate for three types of soccer player robots

In this assessment, the number of complete rotations for each robot in one minute was measured and the total average of them was then calculated. Fig. 14b shows these numbers for three types of robots, i.e.:

1- Two independent driving wheels mechanism

2- Omni directional navigation system (one head)

3- Omni directional navigation and kicking system (three heads)

There is a significant decrease in the rotation rate between the first type and the third type of these robots shown in Fig. 14b. The number of complete rotation per minute for omni directional single head and three heads robot are 5 and 2 respectively. It shows that using the omni-kick system is very useful and can reduce the rotation rate in robots with the same navigation system about 60%.

Essentially, fewer necessary rotations simplifies the algorithm needed for following a trajectory, cooperative behaviors and increases the speed and flexibility for directing the ball to the desired destination.

7. Artificial Intelligence

Artificial Intelligence has been of researchers' interest for many years since the need for autonomous systems emerged. Several approaches to this specific issue have been studied widely in different areas ranging from pure cognitive science outlook to pure engineering perspective; each having its own methodology. Although these approaches have different standpoints, it has shown that amalgamating the findings of each, results in very interesting achievements. In this section we will follow the engineering approach in general, for that its applications in practical engineering problems are more dominating.

From the early days of utilizing machines, systems able to make some simple decisions when needed, became necessary in some applications. Engineers overcame this type of criteria in their design with simple logics which were implemented mechanically or electronically.

But in the past few decades the need for complicated tasks by robots has called for sophisticated methods in artificial intelligence. For example, in situations where robots had to be utilized on another celestial body in the space, the remote control of such robots were not practical, so the robot needed to have its own intelligence in order to accomplish its mission successfully. As another example, assume a robot which has to perform like humans in a particular task (i.e. ping-pong player robot); as humans decide intelligently for such behaviors, such robots has to be able to decide intelligently as well.

From the discussion above, it's pretty much clear that in the field of concentration of this chapter (soccer player robots), artificial intelligence is of great importance in both agent level and team level behavior. In the following subsections the principle ideas have been developed and explored in detail. This section is organized as following, firstly the local and global world models will be discussed, and then the architecture of the proposed AI hierarchy and its modules will be explored in details. Next a discussion on communication protocols, used as a medium for transmitting data between agents, will be considered. And finally a brief discussion on trajectory computation will be carried out.

7.1 World Model Construction

Although each agent tries to extract the real world map from the fusion of visual and non-visual data as accurate as possible, but "noisy data" and "non-global optimized" algorithms reduce the reliability of processed data. First, let us clarify what is meant by "noise" and "optimized algorithms" with a few examples in a mobile robot. The flaws of color space modeling result in wrong color classification, which in turn makes the object detection algorithms prone to errors. As a result, a robot may not see an opponent because of its poor color table for the opponent's tag color, or it may see an orange T-shirt in the spectators' area as a ball! These wrong outputs are referred as "noise". By this classification, the CCD noise pattern or faulty shaft encoder samples due to motors noise are excluded. There is a trade off between speed and reliability in most algorithms. Middle size league in Robocup has a well-defined environment (e.g. distinct colors, defined sizes and etc), which can be very helpful in simplifying the design of a fast algorithm.

Since a predefined environment is assumed, any changes in this environment can more or less result in wrong movements. For example, for self-localization the width of goals are assumed to be fully viewable in close situations; when an object taller than a robot (like a human) cuts or occludes a part of a goal in the image, the output of the vision self-

localization module will not be reliable anymore. Detection of such a situation can be a very cumbersome task and making the algorithm very complicated and therefore slow.

From the discussion above, it is apparent that multi agent data fusion algorithms are necessary for constructing a better approximation of the real world. In addition to the software which resides on each robot, stand alone software for network communication, world model construction, cooperative behavior management and game monitoring need also to be developed. The world model module receives different data sets from every agent. Each data set contains different environmental information like self, ball and opponents' positions. Each data carries a 'confidence' factor; a larger confidence factor means a more reliable piece of information. The most recent data sets are then chosen for data fusion, in which the following rules and facts are applied:

- Closer object are of assumed to be of more accuracy.
- Objects further than a specific distance could be said to be totally inaccurate. (This distance is heuristically obtained)
- An object in the field cannot move faster than an extreme value.

With respect to the above facts, the module filters unwanted duplicates of objects, (e.g. many opponents close to each other seen by different agents), calculates the best approximation for ball and opponents' positions with first order Kalman filtering, gives every object a confidence factor, applies a low pass filter on data and finally constructs a complete world model. This new world model contains information about the objects which may not have been seen by each agent correctly and also enhances approximations of all environmental information. The constructed world model is then sent back to all agents so they will have a better view of the world around them!

7.2 Artificial Intelligence Architecture

The architecture proposed and used for the purpose of a soccer player team has a 3 layer hierarchical framework, namely AI Core, Role Engine and Behavior Engine (Murphy, R., 2000). These layers are completely independent with well defined interfaces to avoid complexities in further developments (i.e. adding new behaviors in order to accomplish a certain role more effective must not influence the AI Core layer). This particular 3 layer architecture enables us to decentralize the whole AI routines among a ground machine and robots in the field accordingly, which in practice means that AI Core, resides in and runs on a ground machine outside the field (along with the monitoring module), while Role and Behavior Engines run as local processes in individual robots' processing units. The following diagram shows the building blocks and their interaction in the proposed architecture.

The interaction between the modules on different machines is provided by a communication protocol which bundles commands and parameters generating command packets and interprets the incoming packets for other modules. In the following, each layer, its interface and parameters will be discussed in details. Finally the communication protocol designed for performing the interactions will be described briefly.

7.2.1 AI Core

As it can be seen from Fig. 15, AI Core is the topmost layer in our proposed architecture. This core has been implemented using case based reasoning method in which all the possible cases had been anticipated during the design process, these cases will be discussed shortly. Although no adaptation or learning takes place in this layer, clever design and useful parameter definition can give enough flexibility to this layer, while avoiding convergence problem in adaptive designs.

The objective of this layer can be simply stated as following:

- Collecting data from World Model Constructor (WMC) module.
- Collecting parameter values set by human supervisor before the game.
- Extraction of GameState from the above parameters and setting it to a member of the following set: GameState=*{HighDefence ,MedDefence, LowDefence, LowAttack, MedAttack, HighAttack}*
- Assigning each necessary role for each GameState to the robot which can execute the role better.
- Sending the role and its parameters along with world model information to selected robots. Now let's take a closer look at the algorithms performing the above steps. The GameState is uniquely derived from the following table.

Ball Region / Ball Ownership	Region 1	Region 2	Region 3	Region 4
Own Team	MedDefence	LowDefence	MedAttack	HighAttack
Opponent Team	HighDefence	LowDefence	LowAttack	MedAttack

Table 1. Derivation of GameState from world model information

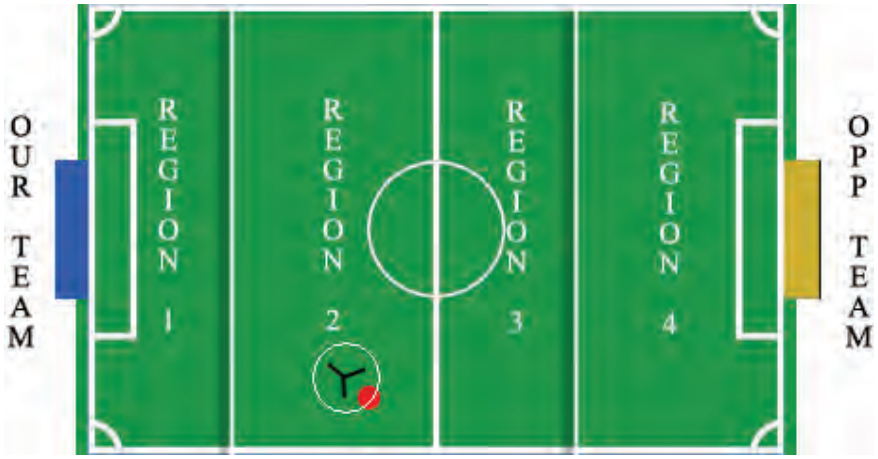


Fig. 15. AI core visual module and the defined regions

In order to avoid undesirable sudden changes of *GameState*, which can result in sudden changes in roles assigned to robots, *GameState* determination was implemented memory based. This means that the current *GameState* is selected as the most dominant *GameState* in a pool of *GameStates* from the last few seconds. In AI Core, roles are changed in a manner in which a continuity exists between current and previous assigned roles, therefore, robots never experiment sudden changes in roles (for example the role defense never changes to attack in the next cycle). Four roles are assigned for each *GameState* manually based on the evaluation of the opponent team strategy. For example, *MedAttack* might be associated with the following roles: *Goal Keeper*, *CenterDefence*, *Supporter* and *Attacker*. In a more conservative situation *Supporter* might be substituted with a *CenterFieldSupporter*.

7.2.2 Role Engine

After assignment of each necessary role to the robot which can perform it better by AI Core, the role, along with its optional parameters are sent to the agent through the communication layer. The task of the Role Engine is to:

- Initiate the new assigned role (i.e. by proper termination of the previous role)
- Initiate a thread to feed the Behaviour Layer with necessary behaviours in order to accomplish the assigned role.
- Determining the essential behaviours according to the parameters received from the AI Core, and feeding them sequentially to the Behaviour Layer.
- Watching the results of each behaviour returned from the Behaviour Layer (i.e. success or fail) and deciding the action to take according to results returned.

7.2.3 Behaviour Layer

Behaviours are the building blocks of the robot's performance which includes simple actions like rotating (*behRotate*), or catching the ball (*behCatchBall*) and etc. The Behaviour Layer is the lowest layer in our architecture.

This layer receives a sequence of behaviours along with some parameters from the upper layer (Role Engine) and executes the essential subroutines in order to accomplish a certain behaviour. These subroutines use world model information and trajectory data in order to perform necessary movements.

7.3 Cooperative Behaviour

Here we give an example of how all these layers and modules cooperate in a synchronous fashion to finally show an intelligent set of behaviours. Assume in a certain point of time during the match, AI Core evaluates the robots' positions, the ball location and the team possessing it from the global world model reported by WMC, and then concludes the state of the play to be *MedAttack* from table 1. This defines the strategy of the game which consequently requires some certain roles to be present in the field like Attacker, Supporter and Defender. The AI Core then assigns each role to the robot which is most qualified to perform that role. In order to avoid unwanted bouncing between roles of a single robot, there exists a First In Last Out (FILO) queue for each robot in the AI Core. This queue acts as

an intermediate between AI Core and each single robot, for that it fetches the roles assigned by the AI Core to each single robot and then from the previous values of its memory, determines if the new role has enough credibility to be assigned to the robot or if the current role is still the winner of the queue (having the majority of positions in the queue). This way the roles are somehow low pass filtered before assignment.

Now suppose the role Attacker is finally decided to be assigned to a robot by the FILO queue in the AI core; this role along with some parameters (i.e. shooting distance) is passed to the robot's Role Engine. The Role Engine evaluates the state of the robot by checking its global world model and determines if the robot possesses the ball or not. In case of no possession, it sends the Behaviour Layer a command to start *behCatchBall* behaviour. This routine gets the rudder of the robot, fetches the best path to the destinations from trajectory module and then issues appropriate commands for the control module to move the robot to its destination (i.e. behind ball in this example). If the behaviour was successfully finished, which means complete possession of the ball, it returns a success code to the role which has called *behCatchBall*. Having the possession of the ball, now, the role Attacker calls another behaviour *behDribbleBall* which is again a subroutine in the Behaviour Layer. The intent of this behaviour could be moving the ball toward the opponent's goal while avoiding obstacles (like opponent's robots). When this subroutine got the robot to the desired shooting distance (which was previously passed to the Role Engine by the AI Core), it returns the success code to Attacker role. This role then sends a request to the Behavior Layer to perform the *behShootToGoal* behaviour, and this process repeats as many times as needed. The key point here is that the Role Engine evaluates the state of the robot and selects which behaviour is needed; the rest is left to the Behaviour Layer to watch over proper execution of the behaviour.

8. Experimental Results

In order to evaluate the performance of the position controller suggested in section 4.1 and self-localization error, four experiments were designed.

First, PID position control was applied. The robot tracked on a straight line of 1m length near the center of the field with no rotation. Second, the PD orientation control was employed with just rotation about the Z-axis of the robot. Third, the robot was programmed to follow a sinusoidal curve ("A" in Fig. 16) with the wave-length of 5m and amplitude of 3.5m near the center of the field. Finally, the robot pursued two sinusoidal curves similar to curve A, but far from the center of the field ("B" and "C" in Fig. 16).

In the first experiment, the PID constants were set as those calculated in section 1.C. The maximum deviation from the straight-line tracking and the final position error were measured to be 8 cm and 4 cm respectively (right line in Fig. 16).

In the second experiment, again the PD controller parameters are set to the calculated values for orientation control (section 4.2). The maximum error from the set point angle was $0.03\,\pi$ radians.

These two experiments show that the final error for both tracking and pure rotating are in an acceptable level and the PID and PD controller parameters are selected properly.

In the third experiment, the robot had to track the sinusoidal curve ("A" in Fig. 16) while rotating about its Z-axis.

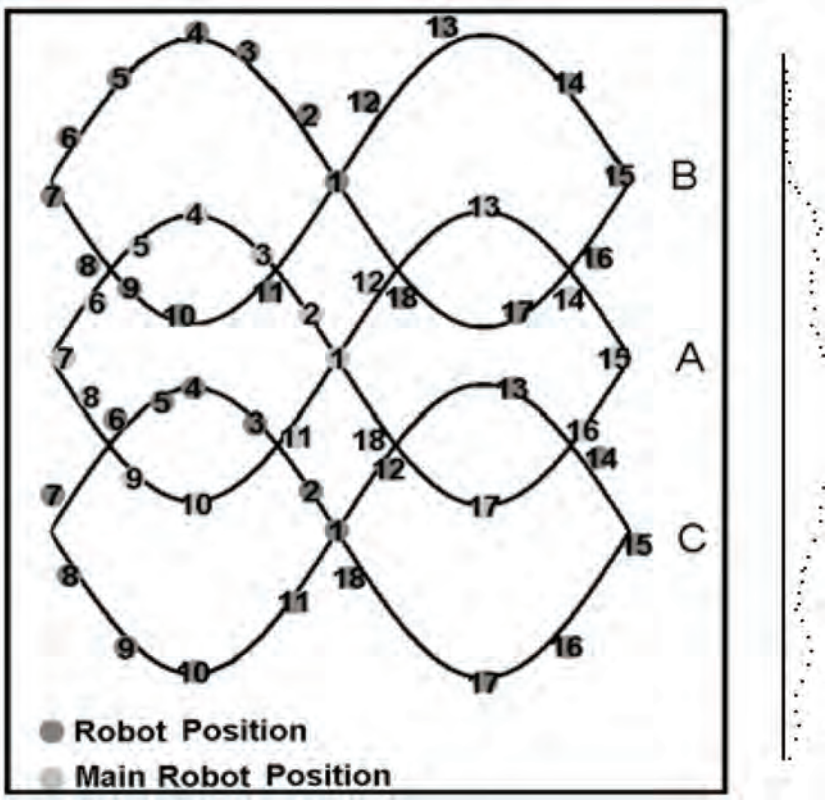


Fig. 16. A, B and C depict the robot trajectories. The numbers shows each robot position on each curve. The right picture illustrates the straight line followed by the robot

The measured errors were between 10 to 12 cm and occurred at points 4, 10, 13 and 17 in curve "A" in Fig. 16. The maximum deviation was measured to be around 12 cm that occurred in point 4.

In the last experiment, the curves were located near the edges of the field ("B, C" in Fig. 16) The maximum deviation between the real and desirable path was measured to be around 23 cm that is less than 7% for this case study.

Figure 16 shows the position of the three robots on the field at different times while Fig. 17. is a picture of the robot tracking on the curves "A", "B", and "C" in the competition field.

Fig. 17. Notations "A", "B", "C" are the robots that followed the corresponding "A", "B", and "C" curves in Fig. 21

9. Conclusion

1. In this study, we propose PID and PD controllers for position and orientation controls respectively. Then, the controller parameters were estimated using a simplified model by taking into account the effect of noise. By using these parameters in the real robot, it was shown that the strategy was appropriate for an omni directional robot.
2. Self-localization method utilized in this study used a combination of the odometry system (using a shaft encoder) and vision-based localization. Using the geometrical properties of circles, we managed to calculate the exact position of the robot in the field. Next, a sensitivity analysis was carried out to determine the inaccurate points in the field. For those regions, we used the flags as our landmarks in the corners to overcome such difficulty. An algorithm, employing the above techniques, was developed and tested on the real field.

   The test results showed that the asymmetric errors for omni directional mobile robots were reduced drastically on those areas. The improvement of performance was more than 80% in position and orientation in comparison with the time when only the original localization was used.
3. For the first time, omni directional navigation system, omni-vision system and omni-kick mechanism have been combined to create a comprehensive omni directional robot. This causes great reduction in robot rotation during soccer play.
4. The idea of separating odometry sensors from the driving wheels was successfully implemented. Three separate omni directional wheels coupled with shaft encoders placed 60 apart of the main driving wheels. The result was reducing errors such as slippage in the time of acceleration.

10. References

Asama, H.; Sato, M.; Bogoni, L.; Kaetsu, H.; Matsumoto, A. & Endo, I. (1995). Development of an omni directional mobile robot with 3 DOF decoupling drive mechanism, *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1925-1930, Japan, May 1995, IEEE, Nagoya

Borenstein, J.; Everett, H. R.; Feng, L. & Wehe, D. (1997). Mobile robot positioning: Sensors and techniques, *J. Robot. Syst.*, Vol. 14, (April 1997) (231–249), 0741-2223

Carlisle, B. (1983). An Omni-Directional Mobile Robot, In: *Development in Robotics*, (79-87), IFS Publication Ltd., 0444865942, England

Chenavier, F.; Crowley, J. (1992). Position estimation for a mobile robot using vision and odometry, *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 2588-2593, France, May 1992, IEEE, Nice

Gaechter, S. (2001). Mirror Design for an Omni directional Camera with a Uniform Cylindrical Projection when Using the SVAVISCA Sensor, In: *The Technical Description of Deliverable D2*, (45-52), Czech Technical University, Czech Republic

Ishiguro H. (1998). Development of Low-cost Compact omni directional sensors and their applications, *Proceedings of International Conference on Information Systems, Analysis and Synthesis*, pp. 433-439, USA, July 1998, Florida

Jung, M.; Kim, J. (2001). Fault tolerant control strategy for OmniKity-III, *Proceedings of IEEE International Conference on Robotics Automation*, pp. 3370-3375, Korea, May 2001, IEEE, Seoul

Kalmar-Nagy, T.; Ganguly, P. & D,Andea, R. (2002). Real-time trajectory generation for omni directional vehicles, *Proceedings of the American control conference*, pp. 285-291, USA, May 2002, IEEE, AK

Kitano H. (1997a). RoboCup: The robot world cup initiative, *Proceedings First International Conference on Autonomous Agents*, pp. 340-347, USA, Feb. 1997, Springer-Verlag, CA

Kitano, H. (1997b). *RoboCup 97: Robot Soccer World Cup I*, Springer-Verlag, 3540644733, New York

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; Osawa, E. & Matsubara, H. (1997). RoboCup: A challenge problem for AI, *AI Magazine*, Vol. 18, No. 1, (April 1997) (73-85), 0738-4602

Kitano, H. (1998). Research program of RoboCup, *Applied Artificial Intelligence*, Vol. 2, No. 2-3, (March 1998) (117-125), 0883-9514

Kortenkamp, D.; Bonasso, P. & Murphy, R. (1998). *Artificial Intelligence and Mobile Robotics,* AAAI Press / MIT Press, 0262611376, USA

Mackworth, A.K. (1993). On seeing robots, In: *Computer Visions: Systems, Theory and Applications*, Basu, A.; Li, X., (1-13), World Scientific, 9810213921, Singapore

Menegatti, E.; Pagello, E. & Wright, M. (2002). Using Omni directional Vision within the Spatial Semantic Hierarchy, *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 908-914, USA, May 2002, IEEE, Washington DC

Muir, P.F.; Neuman, C.P. (1987). Kinematic modeling for feedback control of an omni directional wheeled mobile robots, *Proceedings of IEEE International Conference on Robotics & Automation*, pp. 1772-1786, USA, March 1987, IEEE, NC

Murphy, R. R. (2000). *An Introduction to AI Robotics,* MIT Press, 0262133830, USA

Killough, S.M. (1994). A new family of omni directional and holonomic wheeled platforms for mobile robots, *IEEE Transaction on Robotics and Automation,* Vol. 10, No. 4, (August 1994) (480-493), 0882-4967

Nakano, E.; & Koyachi, N. (1993). An Advanced Mechanism of the Omnidirectional Vehicle (ODV) and Its Application to the Working Wheel Chair for the Disabled, *Proceedings of 83th International Conference on Advanced Robotics*, pp. 277-284, USA, May 1993, IEEE, Atlanta

Olson, C. F. (2000). Probabilistic Self-localization for Mobile Robots, *IEEE Transactions on Robotics and Automation,* Vol. 16, No. 1, (Feb. 2000) (55-66), 0882-4967

Paromatchik, I.; Rembold, U. (1994). A practical Approach to motion Generation and Control of Omni directional Mobile Robot, *Proceedings of IEEE International Conference on Robotics & Automation,* pp. 2790-2795, USA, May 1994, IEEE, CA

Samani, H.; Abdollahi,A.; Ostadi, H.; Ziaie Rad, S. (2004). Design and development of a comprehensive omni directional soccer player robot, *International Journal of Advanced Robotic Systems*, Vol. 1, No. 3, (Sept. 2004) (191-200), 1729-8806

Stroupe, A.; Sikorski, K. & Balch, T. (2002). Constraint-based landmark localization, *Proceedings of RoboCup 2002: Robot Soccer World Cup IV,* pp. 447-453, Japan, 2002, Springer-Verlag, Fukuoka

Talluri, R. & Aggarwal, J. K. (1993). Position estimation techniques for anautonomous mobile robot—A review, In: *Handbook of Pattern Recognition and Computer Vision*, Chen, C. H.; Pau, L. F. & Wang, P. S. P., (769–801), World Scientific, 9812561053, Singapore

Watanabe, K. (1998). Control of an omni directional mobile robot, *Second International Conference on Knowledge Base Intelligent Electronic Systems*, pp. 51-60, Australia, April 1998, Adelaide

West, M. & Asada, H. (1992). Design a Holonomic Omni Directional Vehicle, *Proceedings of IEEE International Conference on Robotics & Automation,* pp. 97-103, France, May 1992, IEEE, Nice

Weigel, T.; Gutmann, J.-S.; Dietl, M.; Kleiner, A.; Nebel, B. (2002). CS Freiburg: coordinating robots for successful soccer playing**,** *IEEE Transactions on Robotics and Automation*, Vol. 18, (Oct. 2002) (685-699), 0882-4967

Weiss, G. (2000). *Multiagent systems: A Modern Approach to Distributed Artificial Intelligence,* MIT Press, 0262232030, USA

Yachida, M. (1998). Omni directional Sensing and Combined Multiple Sensing, *Proceedings of IEEE and ATR Workshop on Computer Vision for Virtual Reality Based Human Communications*, pp. 20-27, India, Jan. 1998, IEEE, Bombay

Yagi, Y. (1999). Omni directional Sensing and its Applications, *IEICE TRANS ,INF. & SYST,* Vol. E82-D, No.3, (March 1999) (568-579), 0916-8532

19

Event-driven Hybrid Classifier Systems and Online Learning for Soccer Game Strategies

Yuji Sato
Hosei University
Japan

1. Introduction

The field of robot soccer is a useful setting for the study of artificial intelligence and machine learning. By considering the learning processes used in multi-agent systems, such as cooperative action learning with multiple agents, optimization of strategies for new opponents, robust handling of noise and other disturbances, and real-time learning during live gameplay, it is possible to grasp real-world problems in a fairly abstract way. For this reason, there has recently been active research and exchange of information concerning a robot soccer game contest called RoboCup. Meanwhile, in simulations using robots, it is necessary to tackle noise and to address the issues involved in processing the signals obtained from multiple sensors, and it is not always possible to evaluate and analyze this information effectively. When focusing on game strategy learning, it is often effective to perform a priori evaluation and analysis by computer simulation. In this section we introduce an idea for autonomous adaptive evolution with respect to the strategies of opponents in games, and we present the results of evaluating this idea. Specifically, we start by introducing a hybrid system configuration of classifier systems and algorithmic strategies. Then, with the aim of implementing real-time learning in mid-game, we introduce a bucket brigade algorithm which is a reinforcement learning method for classifiers, and a technique for restricting the subject of learning depending on the frequency of events. And finally, by considering the differing roles assigned to forwards, midfielders and defenders, we introduce a technique for performing learning by applying differences to the reward values given during reinforcement learning. We pitted this technique against soccer game strategies based on hand-coded algorithms, and as the results show, our proposed technique is effective in terms of increased win rate and the speed of convergence on this win rate.

2. Soccer Video Game and Associated Problems

2.1 Overview of Soccer Video Game

The type of soccer game that we will deal with here is a software-driven video game with soccer as its theme in which two teams battle for the most points. Figure 1 shows a typical game scene targeting the area around the current position of the ball. The screen also

includes a diagram showing a total view of the game in the lower right hand corner. The size of the field was set to 20.0×110.0 grid world, considering that the size of the actual soccer pitch is 20 m long ×110 m wide. Positions on the field and the locations of objects placed on the field are defined using three-dimensional coordinates in the width (x), length (y) and height (z) directions. Data on the field is all processed using floating-point values. The movement of the soccer ball is controlled by physical computations. The ball state is determined by its position vector V_{bp}, direction vector V_{bd}, speed V, and acceleration V_a. The position vector V_{bp} and speed V are updated in each cycle of the environment according to Equation (1) below:

$$V_{bp} = V_{bp} + V_{bd} \times V \qquad \text{and} \qquad V = V + V_a. \tag{1}$$

The soccer players can be in any of three states — stationary, accelerating, or moving — and are assigned a position vector V_p, a direction vector V_d, and information about the actions they are performing. Changes of state occur when a player takes some kind of action based on the information input from the environment.

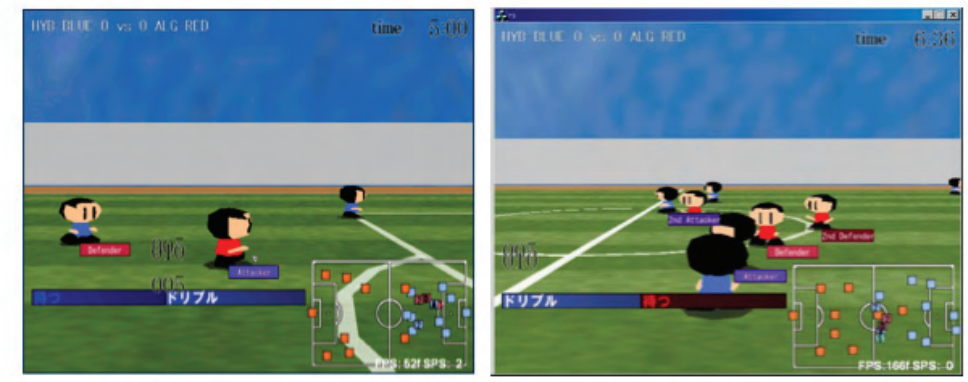


Fig. 1. Example of typical game scene targeting the area around the current position of the ball

Each team has 11 players, and the movements of the 11 players of one team are controlled by computer. The algorithm to control player action is thought up beforehand by a game designer and programmed as a set of control rules in IF-THEN (condition-action) format. Figure 2 shows an example of a rule written in IF-THEN format and the corresponding scene. The rule states "If the ball is right in front of me while I am in front of the goal and if two players of the other team are between me and the goal, then pass the ball to an unmarked player on my team." The program for determining player action consists of a detector, a decision-making section, and an effector. Based on information input from the environment, the detector determines the position and state of each player, the position of the ball, the distance between a player and the goal, etc., and passes these results to the decision-making section. This section then determines player actions according to an algorithm described in IF-THEN format as described above. Examples of player actions

include kick, trap, and move in accordance with current circumstances. The effector finally executes these actions in the environment based on instructions received from the decision-making section. Now, the operation of all or some of the 11 players making up the opposing team is performed by the user, that is, the game player. If the game player is in charge of operating only some of the players on his team, the actions of the remaining players will be controlled by the same algorithm as that of the team controlled by computer.

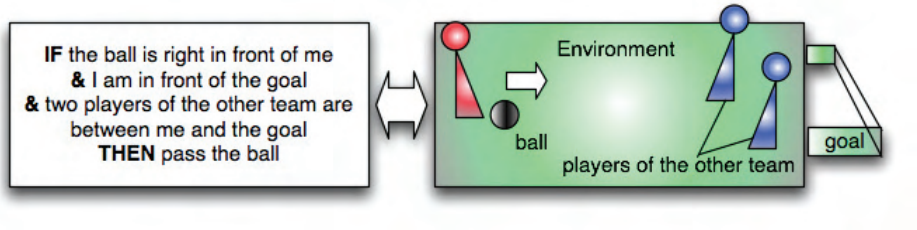


Fig. 2. Example of a rule written in IF-THEN format and corresponding scene

2.2 Problems with Conventional Technique

As described above, the conventional approach to producing a soccer video game is to have a game designer devise the algorithm for controlling player action and to then describe and program that algorithm as a set of rules in IF-THEN format. Recently, however, the Internet is making it easier for anyone to participate in video games and the number of game users is increasing as a result. This development is generating a whole new set of problems. First, the increasing number of users means that the differences in strategies that users prefer and excel in can no longer be ignored and that multiple strategy algorithms must be simultaneously supported. Second, the appearance of users with advanced techniques has generated a need for decision-making algorithms under even more complicated environments. And finally, as the Internet makes it easy for new users to appear one after another, it must be possible to provide and maintain bug-free programs that support such complex decision-making algorithms in a time frame much shorter than that in the past. In other words, the human- and time-related resources required by development and maintenance work are increasing dramatically while the life cycle of each game is shortening. The conventional technique is hard pressed to deal with this situation.

3. Hybrid Decision-making System

We have studied the equipping of game programs with machine learning functions as an approach to solving the above problems. This is because incorporating machine learning functions in an appropriate way will enable the system to learn the game player's strategy and to automatically evolve a strong strategy of its own. It will also eliminate worries over program bugs and significantly reduce the resources required for development and maintenance. A number of techniques can be considered for implementing machine learning functions such as neural networks, Q-learning (Sutton & Barto, 1998) and genetic algorithms (GAs), and we have decided, in particular, on incorporating functions for acquiring rules based on classifier systems (Holland, 1992). We came to this decision considering the many examples of applying evolutionary computation to the acquisition of robot decision-making

algorithms (Gustafson & Hsu, 2001; Luke, 1998; Pietro et al., 2002) in the world of robot soccer games such as RoboCup (Kitano et al., 1997; RoboCup), learning classifier systems takes advantage of GAs and reinforcement learning (Sutton & Barto, 1998) to built adaptive rule-based systems that learn gradually via online experiences (Holmes et al., 2002; Huang & Sun, 2004; Kovacs, 2002), and considering the compatibility between the IF-THEN production-rule description format and classifier systems and the resulting ease of program migration.

At the same time, the bucket brigade algorithm (Belew & Gherrity, 1989; Goldberg, 1989; Holland, 1986; Riolo, 1987a; Riolo, 1987b; Riolo, 1989) used as a reinforcement learning scheme for classifier systems needs time to obtain an effective chain between classifiers. As a result of this shortcoming, the bucket brigade algorithm is not suitable for learning all strategies from scratch during a game. A conventional algorithm, on the other hand, provides solid strategies beforehand assuming fixed environmental conditions, but also includes a rule that states that a player encountering undefined environmental conditions must continue with its present course of action. In light of the above, we decided to apply classifier-based learning to only conditions/actions not described by an explicit algorithm. In short, we adopted a hybrid configuration combining a conventional algorithm and a learning section using a classifier system (Sato & Kanno, 2005).

Figure 3 shows the basic idea of the hybrid decision-making system using a classifier system. This hybrid system is achieved by embedding a conventional algorithm into a classifier system as a base. The conventional algorithm is unaffected by learning and is implemented as a set of "privileged classifiers." Specifically, the reliability (credit or strength) of a privileged classifier is set to the highest possible value and is not targeted for updating by learning. If, after analyzing a message list, there are no privileged classifiers in the classifier list that match a current condition, the strength of a classifier that does is updated. Classifiers can also be discovered here using genetic algorithms.

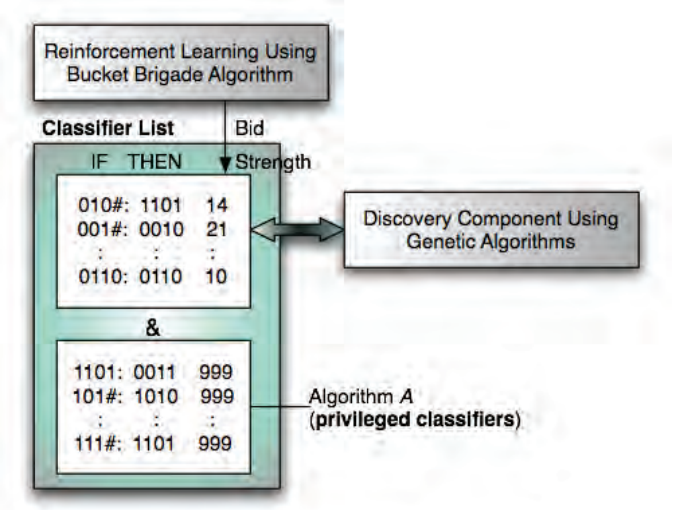


Fig. 3. Basic idea of the hybrid decision-making system using a classifier system

4. Event-driven Learning Classifier System

The preliminary experiments revealed that a hybrid-type system has the potential of exceeding a human-designed algorithm provided that search space can be contracted by limiting the target of learning to actions. On the other hand, having humans select conditions beforehand does nothing to eliminate the problems associated with the conventional way of generating conditions.

To solve this dilemma, we decided to switch the rules to be learned for each game player (user) that the computer opposes. This is because the total possible search space in theory need not be the target of learning if only the strategy of the game player in the current match can somehow be dealt with. Furthermore, it was decided that all of the current player's strategies would not be targeted for learning but rather that the number of events targeted for learning would be limited to that that could be completed in real time. Figure 4 shows the configuration of the proposed event-driven classifier system (Sato & Kanno, 2005). This system differs from standard classifier systems in three main ways. First, the proposed system adds an event analysis section and creates a table that records event frequency for each game player. Second, the classifier discovery section using genetic algorithms targets only actions while conditions are generated by adding new classifiers in accordance with the frequency of actual events. Third, the system updates the strength of classifiers by the bucket brigade algorithm starting with high-frequency events and continuing until learning can no longer be completed in real time. The proposed system also adopts a hybrid configuration combining a conventional algorithm and classifier system as before. Finally, the system provides for two types of rewards that can be obtained from the environment: a large reward obtained from winning or losing a game and a small reward obtained from succeeding or failing in a single play such as passing or dribbling the ball. In short, the above system focuses only on strategy that actually occurs with high frequency during a game and limits learning space to the range that learning can be completed in real time.

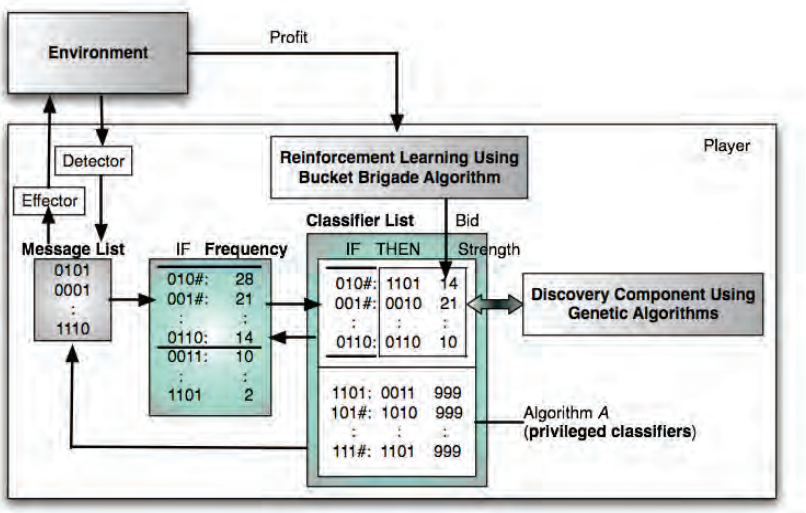


Fig. 4. The configuration of the event-driven hybrid learning classifier system

5. Reward Allotment Based on the Role of Each Position

Table 1 shows the success rewards for each play that were used in these tests. Preliminary tests were performed to make a prior survey of the number of times the players performed pass and dribble actions in a single game, on the basis of which the rewards for passing and dribbling were set so that the product of the success reward for passing and the number of passes made was more or less equal to the product of the success reward for dribbling and the number of dribble actions performed. The goal-scoring success reward was set to a high value because goal-scoring is of great importance to the outcome of a game. For all the in-game agents apart from the goalkeeper, learning was performed by applying success rewards to each play without any particular regard to differences in the role of each position. For example, when a pass was made successfully, bucket brigade learning was performed so that the same reward value (16) was obtained by each player irrespective of whether the player was assigned to a forward, midfielder or defense role. And when a player takes the ball from a member of the opposing team, bucket brigade learning is performed by obtaining the same reward value (15) regardless of the difference in roles between the players involved.

	GETGOAL	DRIBBLE	PASS	GETBALL	LOSTBALL	TOTAL
Forwards	60	4	16	15	-50	45
Midfielders	60	4	16	15	-50	45
Defense	60	4	16	15	-50	45

Table 1. The success rewards for each play that were used in a team *H*1

On the other hand, in real soccer games, the forward, midfielder and defense players are assigned different roles and emphasize different aspects of their play depending on these assigned roles. Accordingly, it is thought that giving different success rewards to each player considering the role assignments of forward, midfielder and defense players might lead to a better game winning rate. These role assignments into consideration might lead result in cooperative learning that contributes to a better winning rate. Table 2 shows the basic concept for determining the success rewards for each player (Sato et al., 2006). For example, a forward should take as many shots at goal as possible in order to gain points. Forwards are therefore given a large success reward for shots at goal, while their reward for stealing the ball from the opposing team is made relatively small. Conversely, the main duty of defense players is to prevent the opposing team from being able to take shots at goal. Defense players are thus given greater rewards for stealing the ball from the other team, and relatively small rewards for successful shots at goal. Meanwhile, the role of midfielders is to move the ball forwards to connect between the defense and forward players, and to act as surrogate defense or forward players when necessary. Accordingly, their success rewards are more evenly spread, with extra emphasis on actions such as passing and dribbling.

	GETGOAL	DRIBBLE	PASS	GETBALL	LOSTBALL
Forwards	Large	Average	Average	Small	Average
Midfielders	Average	Large	Large	Average	Average
Defense	Small	Average	Average	Large	Average

Table 2. The basic concept for determining the success rewards for each player

6. Evaluation Experiments

6.1 Experiment on Event-driven Hybrid Learning Classifier Systems

6.1.1 Evaluation Method

We prepared three strategy algorithms beforehand to help generate data for evaluation purposes. The first one is strategy-algorithm *A* as a product prototype. The remaining two are strategy-algorithm *B* and strategy-algorithm *C* both based on strategy-algorithm *A* but modified to place weight on offense and defense, respectively. These three strategies were made to play against each other beforehand and each was set to have about the same winning percentage.

In the experiments, the outcome of games played between two teams in a soccer environment was observed. The players on one team used one of the above conventional algorithm-type decision-making systems while those on the other team used the event-driven hybrid classifier system proposed in Section 4. The first 20 seconds during a single game was time for learning and applied to constructing a classifier system. Each pair of teams played 10,000 matches and team effectiveness was evaluated from its winning rate R_w defined as the following equation.

$$R_w = N_w \,/\, (N_t - N_d), \tag{2}$$

where N_t, N_d, and N_w are total number of matches, number of draws, and number of wins respectively. The experiments evaluated the ability of the proposed event-driven classifier system to deal with a diverse environment and to adapt to a dynamic environment.

First, Fig. 5 summarizes the experiment for evaluating the ability to deal with a diverse environment, that is, the ability to deal with more than one strategy algorithm. Specifically, event-driven classifier system *H*1 incorporating algorithm *A* was made to play against strategy-algorithms *A*, *B*, and *C*, and the outcomes of the resulting matches were observed to see whether learning could be performed to give *H*1 a winning rate better than 50% against all of these strategies.

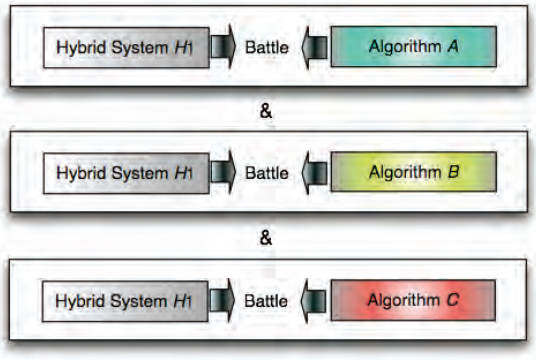


Fig. 5. The experiment for evaluating the ability to deal with a diverse environment

Next, Fig. 6 summarizes the experiment for evaluating the ability to adapt to a dynamic environment, that is, the ability to adapt to changes in strategy. Here, event-driven classifier system *H*1 incorporating algorithm *A* was first made to play against strategy-algorithm *A*. Next, given that system *H*1 had evolved into system *H*1′ having a strategy that could adequately deal with strategy-algorithm *A*, system *H*1′ was made to play against strategy-algorithm *B* to see whether it could further evolve to achieve a winning rate better than 50%. Similarly, given that system *H*1′ had evolved into system *H*1″ having a strategy that could adequately deal with strategy-algorithm *B*, system *H*1″ was made to play against strategy-algorithm *C* to see whether it could again evolve to achieve a winning rate better than 50%. In short, match outcomes were observed to see whether the hybrid system had the ability to adapt to intermittent changes in strategy along the time axis.

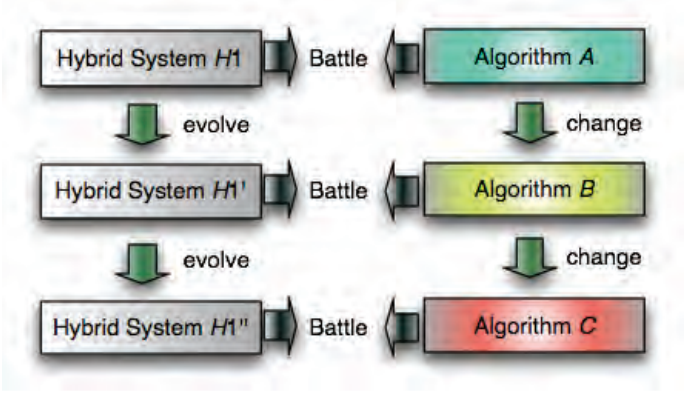


Fig. 6. The experiment for evaluating the ability to adapt to a dynamic environment

6.1.2 Experimental Results and Discussion

Dealing with a diverse environment:
Table 3 shows the results of evaluating the ability to deal with a diverse environment. This table shows the results of 10,000 matches. Against algorithm *A*, system *H*1 won 32%, lost 17%, and drew 51% of the games played. Against algorithm *B*, it won 32%, lost 14%, and drew 54% of the games played. And finally, against algorithm *C*, it won 23%, lost 12%, and drew 65% of the games played. In other words, system *H*1 exhibited a degree of learning resulting in a winning percentage better than 50% against all three algorithms. Figure 7 shows the relationship between number of matches played and winning rate R_w. This figure shows the first 500 matches for each pairs of teams. These results show that system *H*1 could adapt to each of the three algorithms in several ten matches.
The results shown in Table 3 and Fig. 7 tell us that event-driven classifier system *H*1 incorporating algorithm *A* could achieve a winning rate better than 50% against strategy-algorithms *A*, *B*, and *C* by learning. System *H*1 therefore has the ability of dealing with a diverse environment.

	Won	Lost	Drew
Algorithm *A* vs. *H*1	32%	17%	51%
Algorithm *B* vs. *H*1	32%	14%	54%
Algorithm *C* vs. *H*1	23%	12%	65%

Table 3. Ability to deal with a diverse environment. This table shows the results of 10,000 matches

Adapting to a dynamic environment:
Table 4 shows the results of evaluating the ability to adapt to a dynamic environment. This table shows the results of 10,000 matches. Here, classifier system *H*1′ is the result of learning by playing against strategy-algorithm *A*, and from the table, we see that it also adapted to algorithm *B* by playing against that algorithm to the point of winning 31%, losing 14%, and drawing 55% of the games played. Likewise, classifier system *H*1″, the result of adapting to algorithm *B*, also adapted to algorithm *C* by playing against that algorithm to the point of winning 22%, losing 12%, and drawing 66% of the games played. Figure 8 shows the relationship between number of matches played and winning rate R_w for system *H*1′ with respect to algorithm *B* and for system *H*1″ with respect to algorithm *C*. This figure shows the first 500 matches for each pairs of teams.

	Won	Lost	Drew
(*A* ->) *B* vs. *H*1′	31%	14%	55%
(*B* ->) *C* vs. *H*1″	22%	12%	66%

Table 4. Ability to dynamic environment. This table shows the results of 10,000 matches

As for the experiment on evaluating the ability to adapt to a dynamic environment, the results of Table 4 and Fig. 8 show that the event-driven classifier system could evolve and achieve a winning rate better than 50% in the face of intermittent changes in strategy along the time axis.
At the same time, Figs. 7 and 8 show that this system requires about 80 matches to evolve to a point where it can either deal with a diverse environment or adapt to a dynamic environment. To achieve a practical, working system, though, it is desirable that the system be able to adapt with fewer learning steps. Making learning more efficient with a smaller number of matches is a topic for future research.
In either case, the event-driven classifier system could adapt to the conventional algorithm in question in about 80 matches. As reference, Fig. 9 shows the relationship between number of matches played and success rate of dribbling, and shooting. These results reveal that an event-driven classifier system can improve the success rate of dribbling and shooting by about 5%.

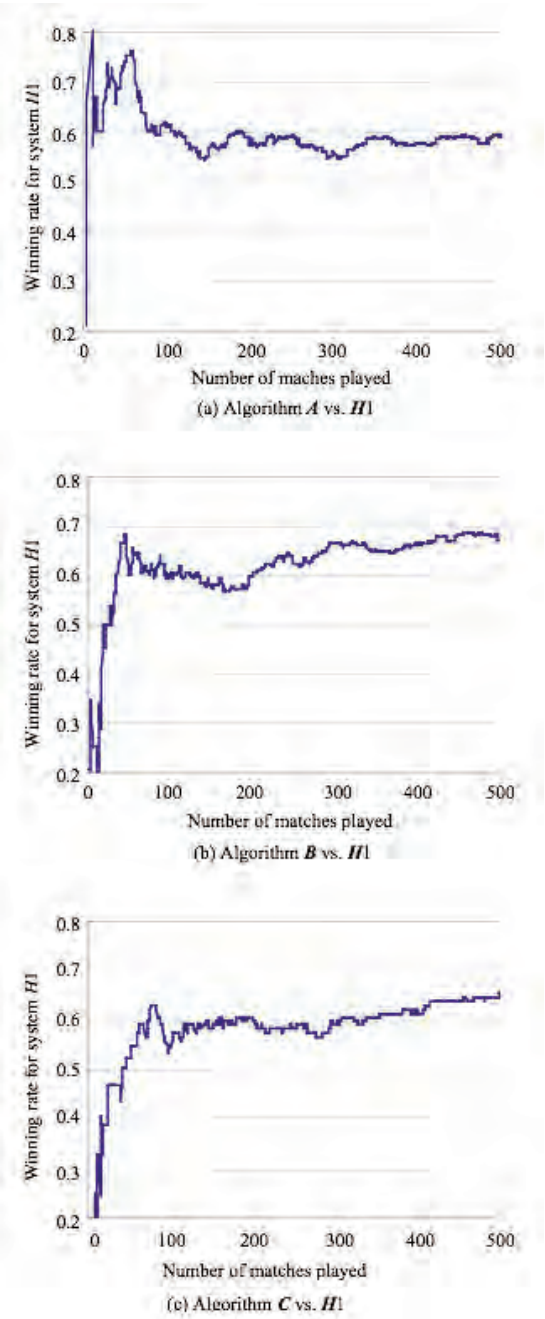


Fig. 7. The relationship between number of matches played and winning rate of *H*1

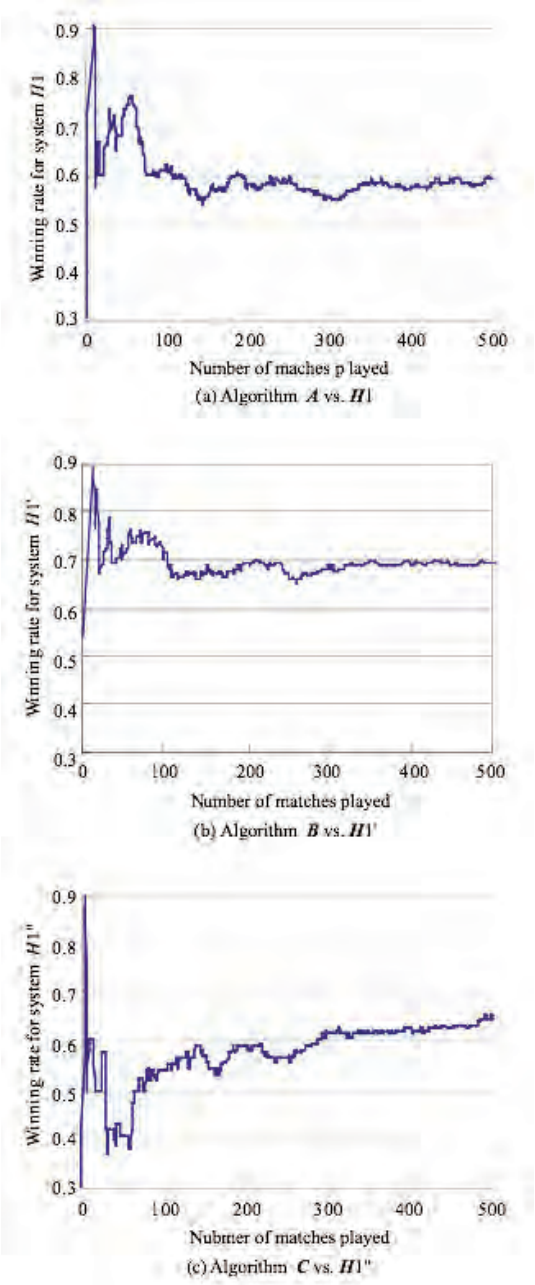


Fig. 8. The relationship between number of matches played and winning rate for system *H*1, *H*1′, and *H*1″

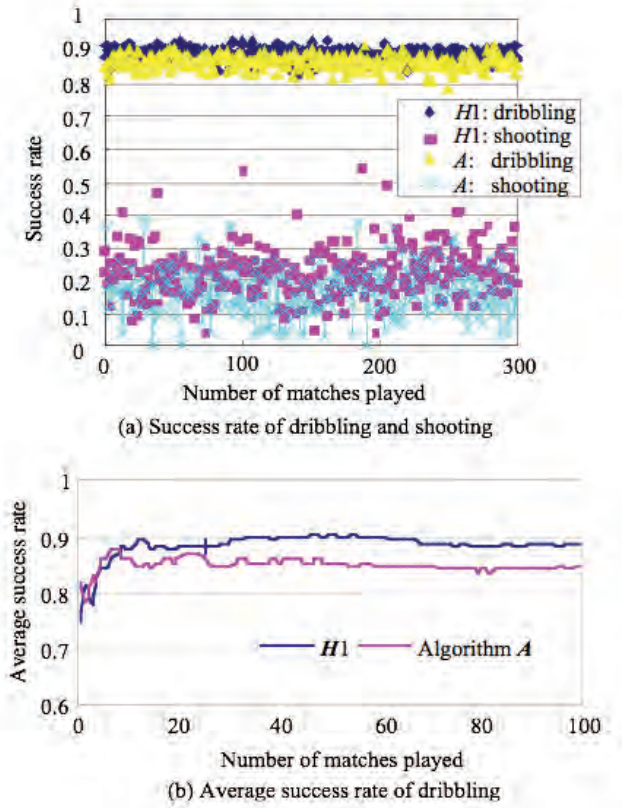


Fig. 9. The relationship between number of matches played and success rate of dribbling and shooting

6.2 Experiment on Reward Allotment Based on the Role of Each Position

6.2.1 Evaluation Method

For the evaluation data, we used three different algorithmic strategies that were employed in earlier trials. The tests involved playing matches between two teams in a soccer environment and observing the number of games won and lost. An algorithmic decision-making system was used for the players of one team, while an event-driven classifier system was used for the players of the other team. The event-driven classifier system was evaluated by using a number of teams in which each player was set with different success reward values for plays such as passing and dribbling, according to the aims of the test.

In practice, we investigated whether or not changes in the game winning rate are caused by giving each player different success rewards based on the role assignments of different positions. We also investigated whether or not there were any changes in the game winning rate by changing the balance of success rewards of each type of play.

Three event-driven classifier systems incorporating algorithm *A* were prepared with differences in the rewards used for bucket-brigade learning. Specifically, these were a team *H*2 in which the success rewards of each player were set considering the role assignments shown in Table 5 in line with the basic policy mentioned above in section 3.3, a team *H*3 in which the success rewards of each player were set as shown in Table 6 based on the opposite idea to the above mentioned basic policy, and a team *H*1 which was set with the rewards used in the prior tests shown in Table 1. These three teams were each made to battle against algorithmic strategies *A*, *B* and *C*, and we comparatively evaluated them by determining the final asymptotic winning rates and the speed with which they converged on these final rates.

	GETGOAL	DRIBBLE	PASS	GETBALL	LOSTBALL	TOTAL
Forwards	80	2	8	5	-50	45
Midfielders	60	4	16	15	-50	45
Defense	40	2	8	45	-50	45

Table 5. The success rewards for each play that were used in a team *H*2

	GETGOAL	DRIBBLE	PASS	GETBALL	LOSTBALL	TOTAL
Forwards	40	2	8	45	-50	45
Midfielders	60	2	8	8	-50	45
Defense	80	2	8	5	-50	45

Table 6. The success rewards for each play that were used in a team *H*3

Next, we will describe the test method used to evaluate the relationship between the winning rate and the balance of success rewards of each type of play. The event-driven classifier system incorporating algorithm *A* provides a total of four teams — two different teams in which the success rewards of each player are set considering their role assignments, and two different teams in which no particular consideration is given to role assignments. Specifically, we provided two new teams — *H*4, in which the balance of success rewards for each play is modified as shown in Table 7 based on the rewards shown in Table 1, and *H*5, in which the balance of success rewards for each play is modified as shown in Table 8 based on the rewards shown in Table 5. Each of these four teams was matched against algorithmic strategies *A*, *B* and *C*, and we compared them with each other in terms of the eventual asymptotic winning rate and the speed of convergence on this rate.
We also investigated the relationships between the success rewards and the success rates of each play and between the winning rate and the success rate of each play, with the aim of using this information to analyze the strategies acquired through learning by the event-driven classifier system.

	GETGOAL	DRIBBLE	PASS	GETBALL	LOSTBALL	TOTAL
Forwards	60	2	8	10	-35	45
Midfielders	60	2	8	10	-35	45
Defense	60	2	8	10	-35	45

Table 7. The success rewards for each play that were used in a team *H*4

	GETGOAL	DRIBBLE	PASS	GETBALL	LOSTBALL	TOTAL
Forwards	80	4	16	5	-60	45
Midfielders	60	8	22	15	-60	45
Defense	40	4	16	45	-60	45

Table 8. The success rewards for each play that were used in a team *H*5

6.2.2 Experimental Results and Discussion

Position role assignments and winning rate:
Figures 10-12 show the results of evaluating the relationships between the position role assignments and winning rates achieved by team *H*1, *H*2, and *H*3. In these figures, each point represents the average result obtained by playing 200 successive games 30 times. From Figures 10-12, the event-driven classifier systems *H*1– *H*5 incorporating algorithm A were able to perform learning to achieve a winning rate of more than 50% with all three of the algorithmic strategies *A*, *B* and *C*. Also, in all the matches with algorithms *A*, *B* and *C*, the winning rates were highest and converged the fastest with team *H*2, where the success rewards of each play were set as shown in Table 5 considering the roll assignments. The lowest rising speed was achieved with team *H*3, where the success rewards of each play were set as shown in Table 6 using weightings opposite to those of the basic strategy. The event-driven classifier system thus seems to be able to contend with opponents having a wide variety of strategies, and it seems that conferring different success rewards to each type of play considering the role assignments of forward, midfielder and defense players results in a better winning rate and faster convergence.

The balance of success rewards of each play and the winning rate:
Figures 10-12 also show the results of evaluating the relationship between the balance of success rewards of each play and the winning rate. In the matches played with all three algorithms *A*, *B* and *C*, team *H*1 ultimately converged on a higher winning rate than team *H*4, and the winning rate also rose at a faster rate. Team *H*2 ultimately converged on a higher winning rate than team *H*5, and its winning rate rose at a faster rate. Our results show that the winning rate and speed of convergence differ significantly when changes are made to the balance of success rewards for each play, regardless of whether or not the position role assignments are taken into consideration.
On the other hand, with regard to the tests for evaluating the relationship between the winning rate and the balance of success rewards for each play, Figs. 10-12 show that differences in the winning rate and the rate of convergence were caused by changing the balance of success rewards for each play independently of whether or not position role assignments were considered. Accordingly, by conferring different success rewards to each type of play by considering the role assignments, and by carefully setting the balance of success rewards for each type of play, it is thought that it is possible to gain further increases in the rate at which games are won and the rate of convergence. On the other hand, with regard to which specific value should be set, no explicitly determined procedure is set in particular. Although it can be determined by trial and error, it is also possible to consider determining the success reward values for each type of play by applying a procedure such as evolutionary computation. Further study will be needed relating to techniques for finding optimal values for the success rewards for each type of play.

Success rate of each play:
Figures 13–15 respectively show the ball possession rates, the number of pass per game, and the number of successful goals per game achieved by each team. The ball possession rates and the number of pass per game exhibit no particular correlation to the winning rate, but were highest for teams *H*1 and *H*2, while teams *H*3 produced lower values of magnitude. As for the number of successful goals per game, all the teams eventually converged on a rate of about 0.6, but our results show that this value rose much faster for team *H*2 which had a high winning rate.
Figures 13 and 14 show that the ball possession rate and the number of pass per game had no particular bearing on the winning rate, with team *H*2 achieving higher values than team *H*1, and teams *H*3 producing low values. On the other hand, in Tables 1, 5 and 6, the sum total of the values of the success rewards for dribbling awarded to forward, midfielder and defense players are found to be 12 for team *H*1, 8 for team *H*2, and 6 for teams *H*3, which corresponds to the order of the ball possession rates in Fig. 13. Also, with regard to the success reward values for passes, the sum total values were 48 for team *H*1, 32 for team *H*2, and 24 for teams *H*3, which corresponds to the ordering of the number of pass per game in Fig. 14. Specifically, it seems that the rate of success of individual play actions has a strong tendency to be dependent on the sum total of the success rewards for each type of play for forward, midfielder and defense players, independently of whether the game is won or lost. On the other hand, the number of successful goals per match was 180 for teams *H*1 and *H*2, and 200 for team *H*3, which does not correspond with the ordering in Fig. 15. In Fig. 15, the goal success rate of team *H*2, which has the highest winning rate, rises the fastest. In order to successfully score a goal, it is impossible to ignore the relationships with other forms of play, and it is thought that rather than being determined solely by the value of the success reward for an individual play, it is strongly related to whether the game is won or lost.

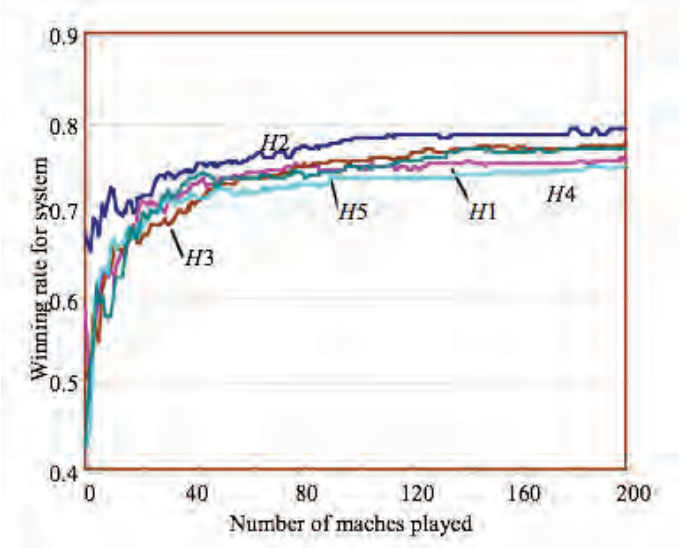


Fig. 10. The relationship between the position role assignments and winning rate (Algorithm *A* vs. *H*1 – *H*5)

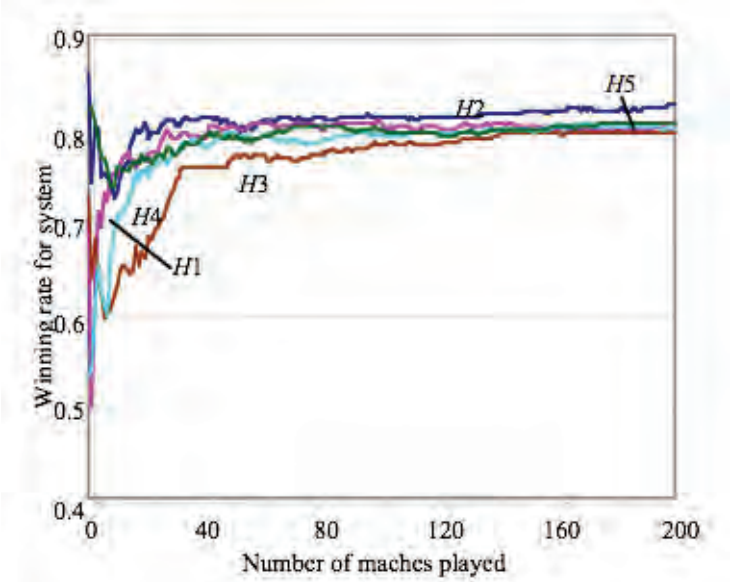


Fig. 11. The relationship between the position role assignments and winning rate (Algorithm *B* vs. *H*1 – *H*5)

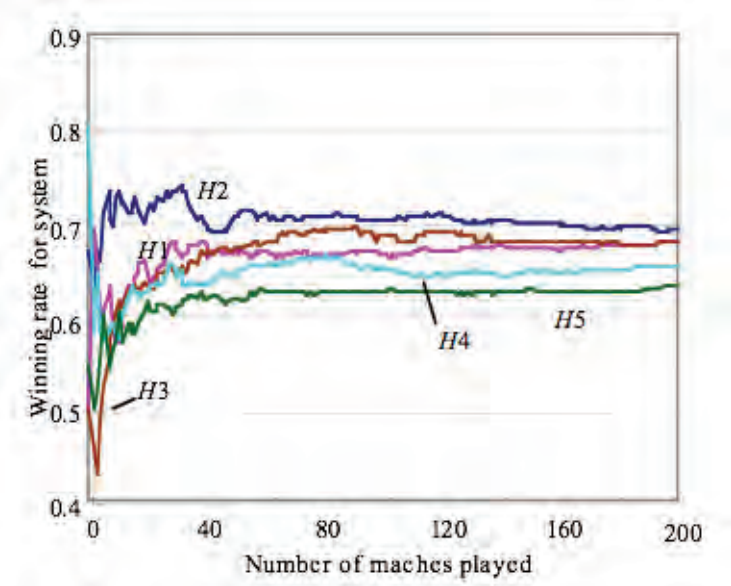


Fig. 12. The relationship between the position role assignments and winning rate (Algorithm *C* vs. *H*1 – *H*5)

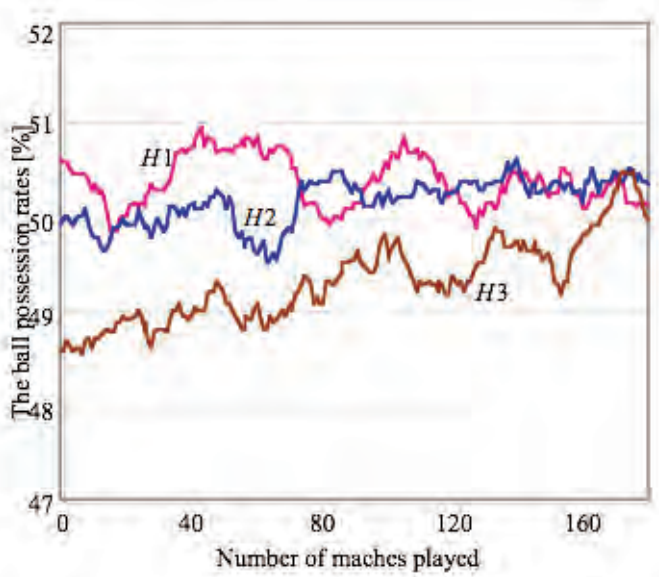


Fig. 13. The ball possession rates achieved by team *H*1, *H*2, and *H*3 (vs. Algorithm *A*)

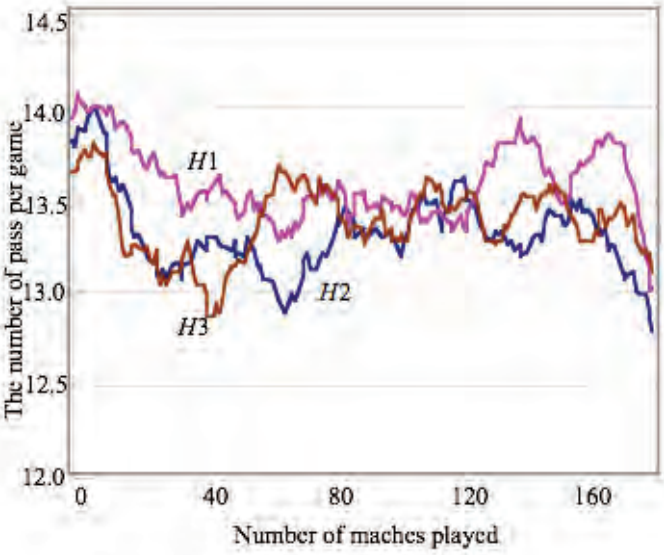


Fig. 14. The number of pass per game achieved by team *H*1, *H*2, and *H*3 (vs. Algorithm *A*)

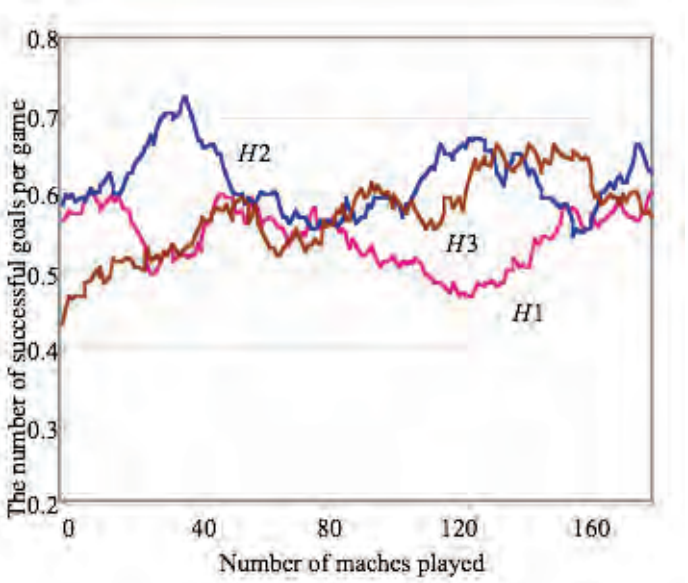


Fig. 15. The number of successful goals per game achieved by team *H*1, *H*2, and *H*3 (vs. Algorithm *A*)

7. Conclusion

In this section we have reported on the results of applying a classifier system to the acquisition of decision-making algorithms by agents in a soccer game. First, we introduced the hybrid system configurations of the existing algorithms and a classifier system. Then, in order to implement real-time learning while a game is in progress, we introduced a bucket brigade algorithm that implements reinforcement learning for the classifier, and a technique for selecting the subject of learning depending on the frequency of events. And finally, we introduced a method for performing learning by awarding players different reward values during reinforcement learning depending on whether they are assigned the role of forward, midfielder or defender. We played this technique against an existing soccer game with hand-coded algorithms, and we evaluated the win rate and the speed of convergence. As a result, we demonstrated that this is an effective means for autonomous adaptive evolution to deal with the opponent's strategies in mid-game. It should be stressed that the technique introduced here has only been evaluated by computer simulation in a video game. When it is applied to a robot soccer game, there are other factors that have to be considered, such as processing information input from multiple sensors and dealing with noise. However, by employing an algorithm that was effective in previous RoboCup contests as the existing algorithm implemented inside the hybrid system, it ought to be an effective technique even in robot soccer games.

8. References

Barry, A. (2003). Limits in Long Path Learning with XCS, In *Proceedings of the Fifth Annual Genetic and Evolutionary Computation Conference*. Vol. 2, pp. 1832-1843, LNCS 2724, Springer-Verlag, Berlin, Heidelberg.

Belew, R.K and Gherrity, M. (1989). Back Propagation for the Classifier System, In *Proceedings of the Third International Conference on Genetic Algorithms*. pp. 275-281, Morgan Kaufmann Publishers, CA.

Bull, L., Wyatt, D., and Parmee, I. (2002). Towards the Use of XCS in Interactive Evolutionary Design, In *Proceedings of the Fourth Annual Genetic and Evolutionary Computation Conference*. pp. 951, Morgan Kaufmann Publishers, San Francisco, CA.

Butz, M.V., Goldberg, D.E., and Lanzi, P.L. (2004). Gradient-Based Learning Updates Improve XCS Performance in Multistep Problems, In *Proceedings of the Sixth Annual Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 751-762, LNCS 3103, Springer-Verlag, Berlin, Heidelberg.

Dawson, D. (2003). Improving Performance in Size-Constrained Extended Classifier Systems, In *Proceedings of the Fifth Annual Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 1870-1881, LNCS 2724, Springer-Verlag, Berlin, Heidelberg.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA.

Gustafson, S.M., and Hsu, W.H. (2001). Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem, In *Proceedings of the Fourth European Conference on Genetic Programming*, pp. 291-301.

Holland, J.H. (1992). *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, MA. The University of Michigan Press, Ann Arbor, 1975.

Holland, J.H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems, In Michalski, R.S. et al. (eds.): *Machine Learning II*, pp. 593-623, Morgan Kaufmann Publishers, CA.

Holmes, J.H., Lanzi, P.L., Stolzmann, W., Wilson, S.W. (2002). Learning Classifier Systems: New Models, Successful Applications. *Information Processing Letters*, Vol. 82, pp. 23-30.

Huang, C-H. and Sun, C-T. (2004). Parameter Adaptation within Co-adaptive Learning Classifier Systems, In *Proceedings of the Sixth Annual Genetic and Evolutionary Computation Conference*, Vol. 2, pp. 774-784, LNCS 3103, Springer-Verlag, Berlin, Heidelberg.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H. (1997). RoboCup: A challenge problem for AI, *AI Magazine*, Vol. 18, pp. 73-85.

Kovacs, T. (2002). What Should a Classifier System Learn and How Should We Measure It? *Journal of Soft Computing*, Vol. 6, No. 3-4, pp. 171-182.

Luke, S. (1998). Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup 97, In *Proceedings of the Third Annual Genetic Programming Conference*, pp. 204-222, Morgan Kaufmann Publishers, San Francisco, CA.

Pietro, A.D., While, L., and Barone, L. (2002). Learning in RoboCup Keepaway using Evolutionary Algorithms, In *Proceedings of the Fourth Annual Genetic and Evolutionary Computation Conference*, pp. 1065-1072, Morgan Kaufmann Publishers, San Francisco, CA.

Riolo, R.L. (1987a). Bucket brigade performance: I. Long sequences of classifiers, genetic algorithms and their application, In *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 184-195, Lawrence Erlbaum Associates, Publishers.

Riolo, R.L. (1987b). Bucket brigade performance: II. Default hierarchies, In *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 196-201, Lawrence Erlbaum Associates, Publishers.

Riolo, R.L. (1989). The emergence of coupled sequences of classifiers, In *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 256-263, Morgan Kaufmann Publishers, CA.

RoboCup web page. http://www.robocup.org/

Sato, Y., and Kanno, R. (2005). Event-driven Hybrid Learning Classifier Systems for Online Soccer Games, In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, pp. 2091-2098, IEEE Press, Edinburgh, Scotland.

Sato Y., Akatsuka Y., and Nishizono T. (2006). Reward Allotment in an Event-driven Hybrid Learning Classifier System for Online Soccer Games, In *Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, pp. 1753-1760, ACM Press, Seattle, WA.

Sutton, R.S., Barto, A.G. (1998). *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, MA.

Wilson, S.W. (1995). Classifier Fitness Based on Accuracy, *Evolutionary Computation*, Vol. 3, No. 2, pp. 149-175, The MIT Press, Cambridge, MA.

Wilson, S.W. (1998). Generalization in the XCS Classifier System, In *Proceedings of the Third Annual Genetic Programming Conference*, pp. 665-674, Morgan Kaufmann Publishers, San Francisco, CA.

20

Robust and Accurate Detection of Object Orientation and ID without Color Segmentation

Hironobu Fujiyoshi, Tomoyuki Nagahashi and Shoichi Shimizu
Chubu University
Japan

1. Introduction

To give optimal visual-feedback, which helps to control a robot, it is important to make its vision system more robust and accurate. In the RoboCup Small Sized League (F180), a global vision system that is robust to unknown and varying lighting conditions is especially important. The vision system, which is in common use, processes an image that identifies and locates robots and the ball. For low-level vision, the color segmentation library called CMVision (J. Bruce et al., 2000) has been used to segment color and to connect component analysis to return colored regions in real time without special hardware. After color is segmented, objects are identified based on the color segmentation results, and then the robot's pose is estimated. To improve the vison system's robustness to varying light conditions, color (A. Egorova et al., 2004) must be calibrated in advance, but the system that does this requires minimal set up time.

In this chapter, we describe a robust and accurate pattern matching method for simultaneously identifying robots and estimating their orientations that does not use color segmentation. To search for similar patterns, our approach uses continuous DP matching, which is obtained by scanning at a constant radius from the center of the robot. The DP similarity value is used to identify object, and to obtain the optimal route by back tracing to estimate its orientation. We found that our system's ability to identify objects was robust to variation in light conditions. This is because it can take advantage of the changes in intensity only.

Related work and our approach are described in section 2. Section 3 describes the method for robust and accurate object identification. The experimental results are presented in section 4. Section 5 discusses some of the advantages of the proposed method. Finally, section 6 concludes the chapter.

2. Related Work

In the RoboCup Small Size League, one team must have 50 mm blue circles centered on the top of its robots, and the other team's robots must have 50 mm yellow circles. To detect the robot's orientation and to identify it, teams are allowed to add extra patches with up to three colors. Figure 1 shows patterns that are currently being used.

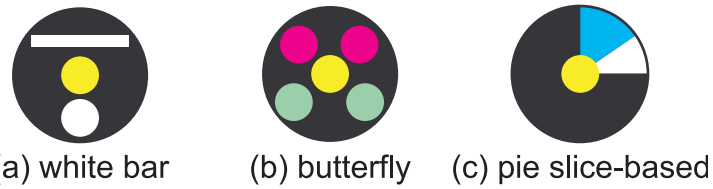


Fig. 1. Example of general identification patterns

Pattern (a), called "white bar", is used to precisely calculate the pose of robot. The robot's orientation is calculated using a white stripe and the least squares method (K. Murakami et al., 2003) or second moment (D. Ball et al., 2004). Other sub patches are also used for identification.
Pattern (b), called "butterfly", has been reported in (J. Bruce & M. Veloso, 2003). Geometric asymmetry can be used to find the rotational correspondence for orientation estimation.
Pattern (c), called "pie slice-based", is unique and is described in (S. Hibino et al., 2002). The method that uses this patch scans the circular pattern using markers on the robot. The low angle resolution is not adequate (8 degrees).
These methods use information from color segmentation to determine a robot's identity. Such colors have problems with changes in brightness and nonuniform color intensity, including sharp shadows, over the field.

2.1 Proposed Patch Pattern

Our approach uses only the changes in intensity obtained by scanning at a constant radius from the center of the robot and does not use the color segmentation results. Therefore, we can paste suitably-colored patches on top of the robot, as shown in Fig. 2. This approach makes possible a large number of different patterns for identification and makes it easy to modify patch patterns. Moreover, preparing the rule-based reference table by user for object identification is no longer necessary.

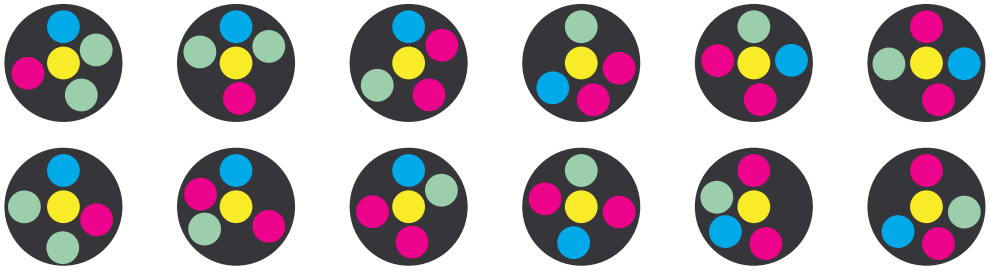


Fig. 2. Examples of our ID plates

3. Object Identification

DP matching calculates the similarity between a reference pattern and an input pattern by matching the intensity changes of the robot's markers. After the DP matching has been done, a similarity value is used to identify the robot. Correspondence of the optimal route obtained by back tracing is used to determine its orientation. The flow of the proposed

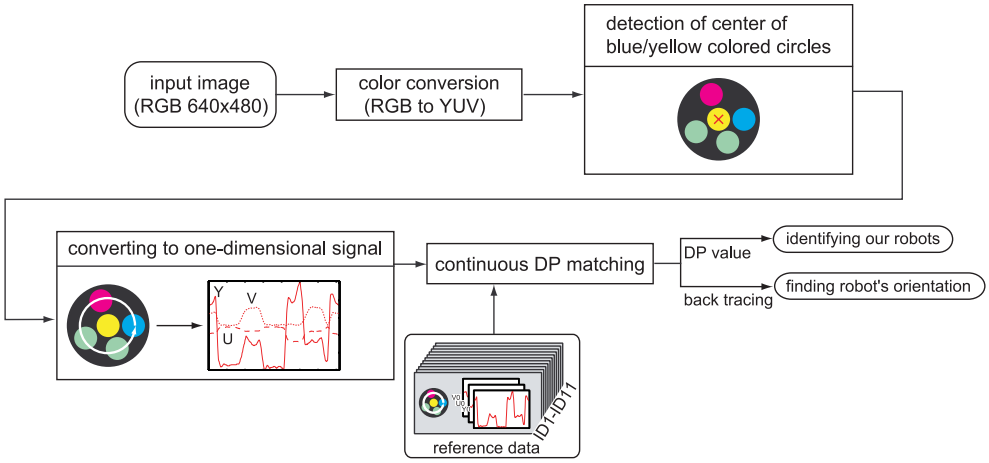


Fig. 3. Overview of our vision system

method is as follows and shown in Fig. 3.

1. Color conversion (RGB to YUV)
2. Detection of the center of blue/yellow colored circles
3. Converting to one-dimensional signal by scanning at some constant radius from the center of the robot
4. Identifying our robots by continuous DP matching
5. Finding the robot's orientation by back tracing

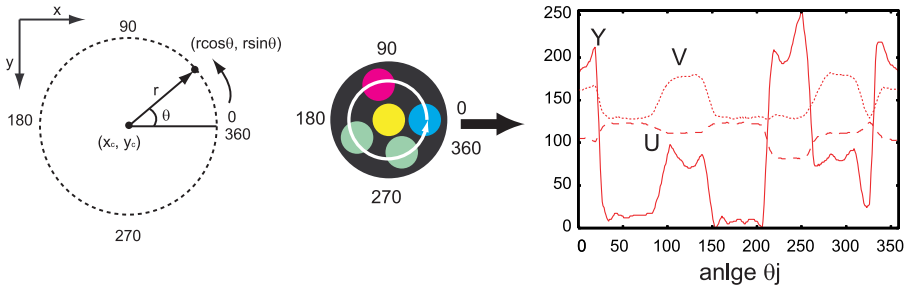
3.1 Detection of the Center of Blue/Yellow Colored Circle

It is important to detect the center of the blue/yellow circle because our approach uses this center position to convert to one-dimensional signals for object identification. The following describes an algorithm used to determine the center of a circle given three points on a plane. The three points determine a unique circle if, and only if, they are not on the same line. The relationship of these three points is expressed as:

$$(x_c - x_i)^2 + (y_c - y_i)^2 = (x_c - x_j)^2 + (y_c - y_j)^2 = (x_c - x_k)^2 + (y_c - y_k)^2 , \quad (1)$$

where (x_c, y_c) is a center coordinate and three points on the image are (x_i, y_i) (x_j, y_j) (x_k, y_k). Equation (1) is a linear simultaneous equation. Thus, (x_c, y_c) is determined by Gaussian elimination using the following steps:

Step1 Detect blue/yellow colored circle.
Step2 Extract contour points of the circle.
Step3 Randomly select three points from the contour points and calculate center position $(x_c, y_c)$by equation (1).
Step4 Increment a count in the accumulator at point (x_c, y_c).
Step5 Repeat Steps 3 and 4 100 times.
Finally, the spot with the most votes is determined to be the center of the main marker.



(a) scanning at a constant radius (b) converting to one-dimensional signal

Fig. 4. Converting to one-dimensional signal

3.2 Converting to One-dimensional Signal

The intensity values of YUV on the top of the robot are obtained by scanning at a constant radius (r=10 pixel) from the detected center of the circle, as shown in Fig. 4. It is impossible to obtain the 359 points (1 degree each) on the circle's perimeter because of the low-resolution of the image. To solve this problem, we apply the bilinear interpolation to estimate the robot's orientation with sub-pixel accuracy.

Image coordinate (x, y) for an angle θ is obtained by

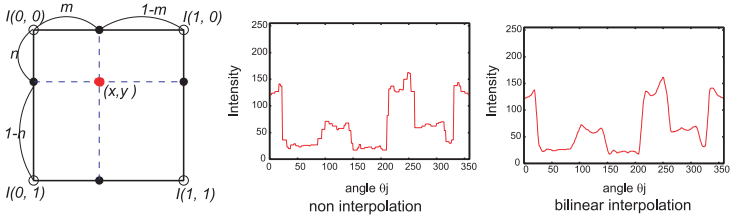
$$x = r\cos\theta + x_c, \quad y = r\sin\theta + y_c, \tag{2}$$

where (x_c, y_c) is the center position on the image coordinate. Since the values of (x, y) are real numbers, the intensity value $I(x, y)$ is interpolated by the bilinear interpolation method used for two-dimensional operations, for instance magnifying an image. The interpolated intensity value is calculated as shown in Fig. 5 (a).

$$I(x,y) = (1-n)((1-m)I(0,0) + mI(1,0)) + n((1-m)I(0,1) + mI(1,1)) . \tag{3}$$

Figure 5 (b) shows the interpolated intensity values of Y. This can be useful in estimating the orientation angle with sub-pixel accuracy. Finally, the intensity values of Y normalized to 0-255, U and V are obtained as a one-dimensional signal from the circle patches on the robot as shown in Fig. 4 (b), and these values are expressed as:

$$I(\theta_j) = I(r\cos\theta_j, r\sin\theta_j) \quad j = 0,\cdots,359 . \tag{4}$$



(a) bilinear interpolation (b) example of interpolation

Fig. 5. Bilinear interpolation

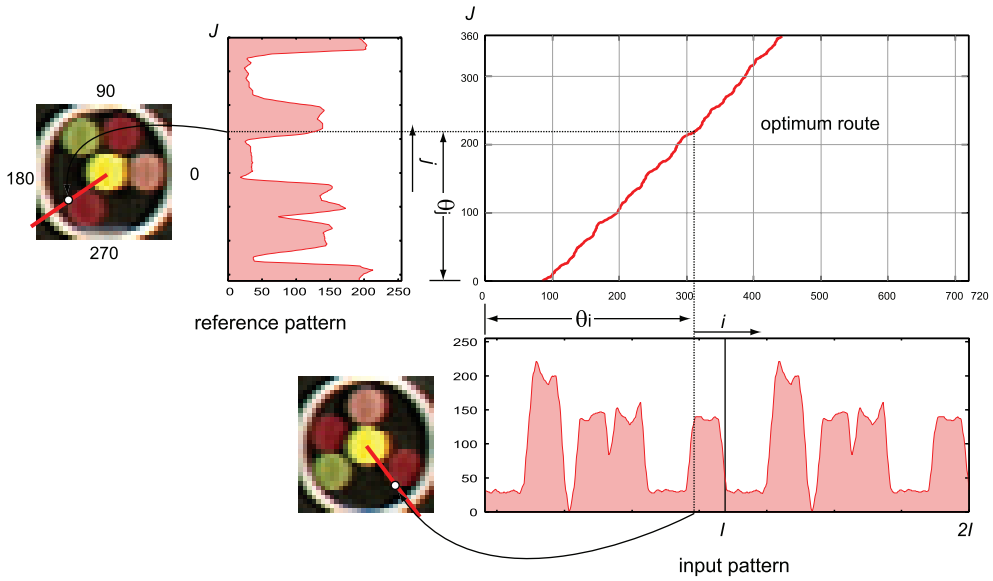


Fig. 6. Example of back tracing

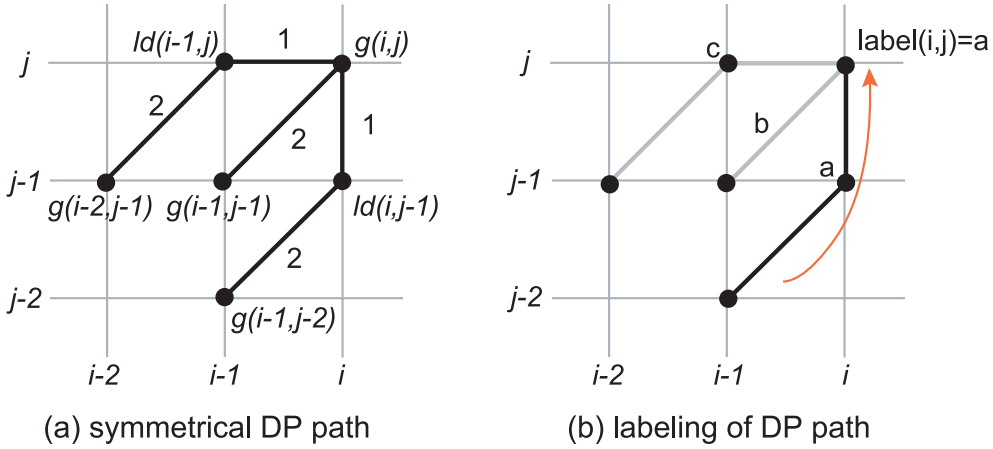


Fig. 7. Symmetrical DP path

3.3 Identifying our Robots by Continuous DP Matching

To uniquely distinguish a robot, intensity values $I(\theta_j)$ as a reference pattern for each robots, are registered initially by clicking with a mouse on points in the direction of the robot's front to help to assign an ID to each robot. Continuous DP matching is done to calculate a similarity between the reference patterns and the input pattern of the current image.

Continuous DP matching DP matching has been used in various areas such as speech-recognition (H. Sakoe et al., 1978). DP matching is a pattern matching algorithm with a nonlinear time-normalization effect. Timing differences between two signal patterns are eliminated by warping the axis of one, so that the maximum coincidence is attached as the minimized residual distance between them. The starting point of the input pattern provided by scanning, as described in section 3.2, is not at the same position as the reference pattern. Therefore, continuous DP matching can be useful in computing the similarity distance by considering the lag of each starting point. The input pattern is repeated twice as ($1 < i < 2I$) and this handling is shown in Fig. 6.
In this implementation, the symmetrical DP path, shown in Fig. 7 (a), is used. Minimum accumulated distance is calculated by the following equations. The vertical axis represents reference pattern frame j, and the horizontal axis represents input pattern frame i. Initial conditions are given as:

$$\begin{cases} g(i,0) = 0 & (i = 0,1,...,I) \\ g(0,j) = \infty & (j = 1,2,...,J) \end{cases}, \tag{5}$$

where I and J are the lengths of the patterns. The minimum accumulated distance g(i, j) on the i frame and j frame are calculated by:

$$g(i,j) = \min \begin{Bmatrix} g(i-1,j-2) + 2 \cdot ld(i,j-1) : (a) \\ g(i-1,j-1) + ld(i,j) \quad\quad : (b) \\ g(i-2,j-1) + 2 \cdot ld(i-1,j) : (c) \end{Bmatrix} + ld(i,j) . \tag{6}$$

Local distance $ld(i, j)$ on the point of (i, j) is computed as:

$$ld(i,j) = (I_t(\theta_t) - I_{t-1}(\theta_j))^2 . \tag{7}$$

The length for the optimal route:

$$c(i,j) = \begin{cases} c(i-1,j-2) + 3 \mid if\,(a) \\ c(i-1,j-1) + 2 \mid if\,(b) \\ c(i-2,j-1) + 3 \mid if\,(c) \end{cases}, \tag{8}$$

is used to obtain the normalized accumulated distance by:

$$G(i) = \frac{g(i,J)}{c(i,J)} . \tag{9}$$

Object ID recognition The continuous DP matching is done to calculate similarity distances for each reference pattern, when a robot's blue/yellow circle is detected. The identity of the robot is determined by selecting the reference pattern which is given the minimum value of G.

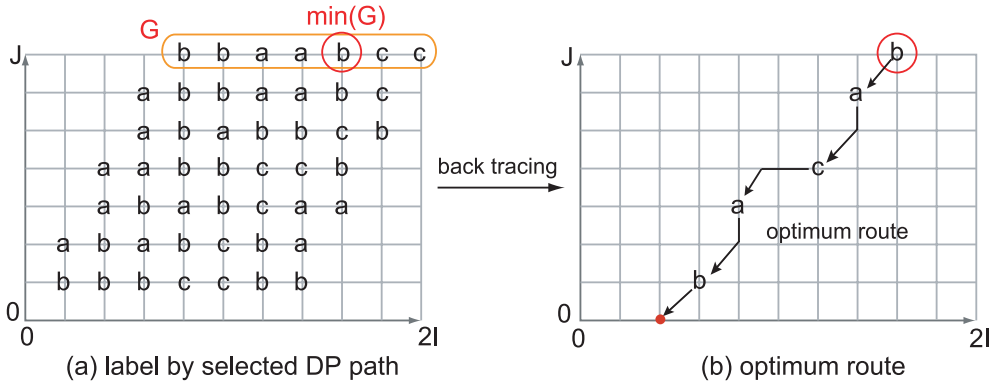


Fig. 8. Back tracing

3.4 Identifying our Robots by Continuous DP Matching

To detect the robot's orientation, back tracing, i.e., computations of local corresponding points of input and reference patterns by reference to the selected DP path, is done as follows:

1. DP matching and labelling of the selected DP path
   While computing the minimum accumulated distance, the path selected by equation (6) is memorized with label a/b/c as shown in Fig. 7 (b).
2. Back tracing
   After normalizing minimum accumulated distance, the minimum value of $G(i, J)$ is selected as a starting point for the back tracing as shown in Fig. 8 (a).

$$i' = \underset{(J/2 \le i \le 2I)}{\arg\min} G(i, J) . \tag{10}$$

The optimum route is tracked by referring to the label, either 'a', 'b', or 'c' at each node as shown in Fig. 8 (b). The DP path labelled 'a' means insert, and 'c' means delete.

Path 'b' means that frame i and j are a pair of corresponding points. When path 'b' appears on the optimum route, the orientation of the current robot θ is estimated by:

$$\theta = \theta_i - \theta_j , \tag{11}$$

where θ_i is the orientation angle of input pattern and θ_j is reference pattern. This process is finished when the route, by back tracing, reaches the end point (j = 0), and the angle θ points at the robot's orientation are averaged (front direction).

As we mentioned above, object orientation and ID are determined by continuous DP matching and not by color segmentation.

Noise	Proposed method		General method	
	SSD	DP	Least-squares method	Second moment
0	0.30	0.76	0.85	1.08
1	1.71	1.10	-	-
2	4.20	1.75	-	-

Table 1. Average of absolute errors of orientation estimation in simulation experiments [degree]

	Proposed method		General method	
	SSD	DP	Least-squares method	Second moment
White bar	0.30	0.76	1.17	0.96
Patch pattern	1.71	1.10	-	-

Table 2. Average of absolute errors of orientation estimation in real experiments [degree]

4. Results

The robustness and accuracy performance of the proposed method in varying light conditions was evaluated by simulation as well as by experiment.

4.1 Results for Orientation Estimation

Simulation results To determine the accuracy of the orientation estimation, the estimated angle using the proposed method is compared to ground truth. Table 1 shows the simulation results from evaluations of 360 patterns (1 degree each). Our method more accuracy estimates orientation than general methods based on the least-squares method (K. Murakami et al., 2003) and the second-moment method (Ball D. et al., 2004) using the white bar ID plate.

The accurate center position of the blue/yellow colored circle for main marker can not be obtained, when there is noise in the circle's perimeter. In this case, we evaluated the robustness of our method using the pattern in which the center positions of the circle translate to its neighbors. Noise 1 in Table 1 is an area of 3x3 pixels, except for the center. Noise 2 is an area of 5x5 pixels, except for the center and noise 1. Five pixels represent 25 millimeters. The SSD in Table 1 indicates the method of linear matching using the sum of squared differences to estimate the orientation. The SSD method is more accurate than the proposed method when a very accurate center position (noise 0) is obtained. However, our method is effective with respect to errors in the center position of the circle because the DP warping function can obtain the optimum correspondence against the gap.

Experimental results Table 2 shows results for experiments using the real vision system, in which a camera is mounted at a height of 4,000 mm. It can be seen that our method performs almost as well as the general method, and that it works well with respect to the white bar. This shows that our method can determine which way the opponent robot is facing.

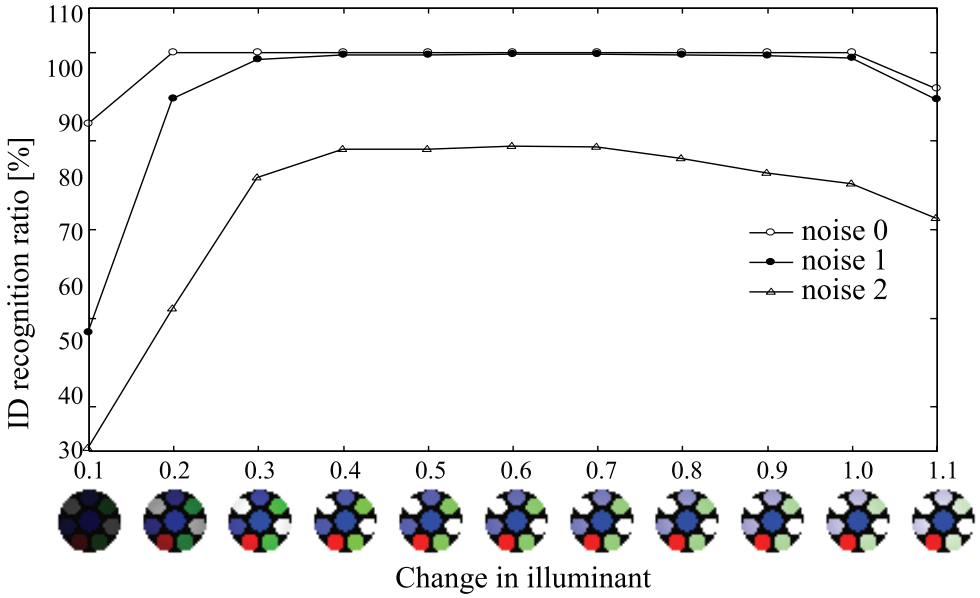


Fig. 9. Results for ID recognition in simulation experiments

This information is useful for intercepting a ball that is being passed.

4.2 Results for Object Identification

Simulation results To determine our methods robustness to variations in light condition, we created a model of illuminant and the marker using CG. We varied the pixel intensity of the input image by changing the illuminant. Figure 9 shows ID patterns under illuminant changes in the simulation and identification performance with 11 unique robots. Our system's performance is stable against the change in lighting conditions. However, recognition performance suffers at noise 2. When there is an error of two pixels in the center, the center is near the edge of the main marker. Therefore, it is difficult to calculate the one-dimensional signal, and the recognition ratio decreases.

Experimental results Figure 10 shows images captured at illuminance ranging from 100-to-2,900 lux. In the experiment, we evaluate 330 images for 11 unique robots in varying light conditions (100~3,000 lux). Figure 11 shows object identification ratios for the 330 images. Note that here, general method means color segmentation-based object identification adjusting the threshold to obtain high performance for lighting condition between 600 to 1,000 lux. On the other hand, for reference patterns of our method, only images captured under 1,000 lux light are registered. It is clear that our method performs better with respect to varying light conditions, because our approach is not based on color segmentation but on matching using changes in intensity obtained by scanning at a constant radius from the center of the robot.

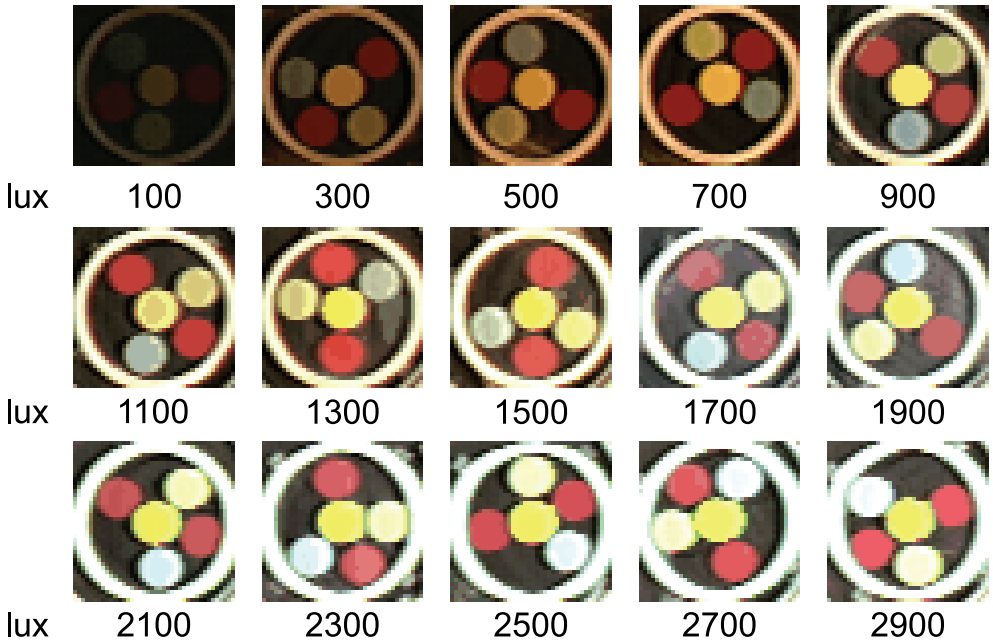


Fig. 10. Images captured at the illuminance from 100 to 2,900 lux

5. Discussion

This section describes some of the benefits of the proposed method.

- Easy set up
  To register reference patterns for each robot's ID, orientation is obtained by clicking on the fronts of the robots to assign an ID to each one. There is no need to make a rule-based reference table to identify objects.
- Easy patch modification
  Since the white bar is used to estimate the robot's orientation in the general method, there is less space in which to paste sub-marker patches. This means that our method allows for more space on the top of the robot and that is very easy to modify the patch pattern because of its easy set up.
- Robustness with respect to varying light conditions
  There is no perfect color segmentation. Even if the lighting conditions change, because of the weather, our method can work well because the changes in intensity are used to detect a robot's orientation and identity.
- Determining the direction of an opponent robot
  Our method for estimating the robot's orientation works well with any shaped-patch patterns such as white bar or butterfly. Therefore, it is possible to know which way the opponent robot is facing. This means that our robot can intercept a ball being passed between opponent robots.

Our method has a disadvantage associated with conversion to a one-dimensional signal. If

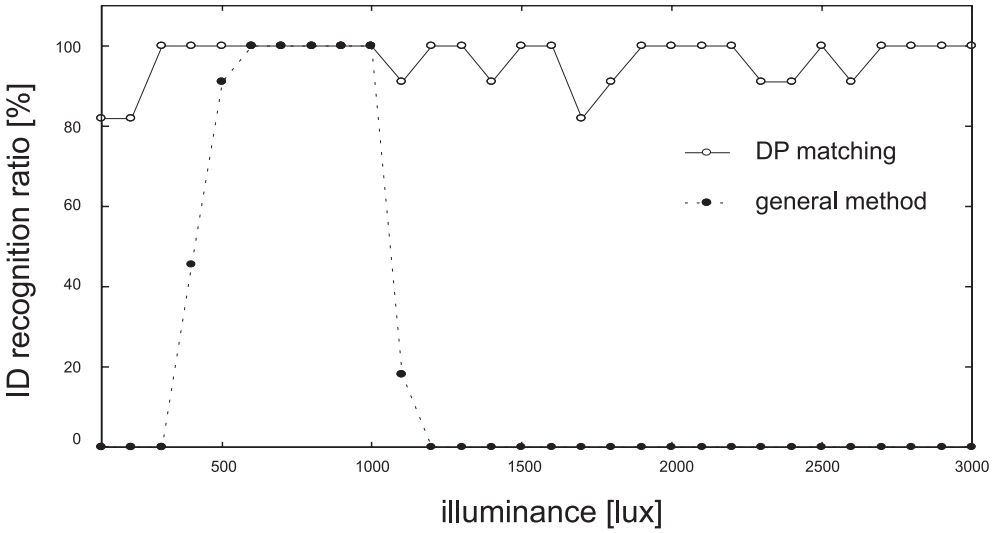


Fig. 11. Experimentals ID recognition results

the center of the circle cannot be accurately calculated, it is difficult to accurately convert to it to a one-dimensional signal. To calculate accurately, the object's identity and orientation must be known, so that the error of the center position within two pixels can be suppressed. Moreover, when determining the ID, it is necessary to compare the input pattern to all reference patterns. Therefore, as the number of robots increases the computational cost is increases.

6. Conclusion

This chapter described a novel approach to detecting orientation and identity of robots without color segmentations. We showed that the proposed method more accurately estimates orientation than the general method, that it is robust to varying light conditions. The system using the proposed method runs in real time on a Xeon 3.0 GHz PC, so the system can be completely setup in a short amount of time by a single operator.
Future works will focus on more automation in the reference pattern registration procedure.

7. References

J. Bruce.; T. Balch. & M. Veloso. (2000). Fast and Inexpensive Color Image Segmentation for Interactive Robots, In Proceedings of IROS-2000, Japan, 2000.

A. Egorova.; M. Simon.; F. Wiesel.; A. Gloye. & R. Rojas. (2004). Plug & Play: Fast Automatic Geometry and Color Calibration for Cameras Tracking Robots, Proceedings of ROBOCUP2004 SYMPOSIUM, 2004.

K. Murakami.; S. Hibino.; Y. Kodama.; T. Iida.; K. Kato.; S. Kondo.; & T. Naruse. (2003). Cooperative Soccer Play by Real Small-Size Robot, RoboCup 2003: Robot Soccer World Cup VII, Springer, pp. 410-421, 2003.

Ball D.; Wyeth G.; & Nuske S. (2004). A Global Vision System for a Robot Soccer Team,

Proceedings of the 2004 Australasian Conference on Robotics and Automation (ACRA), 2004.

J. Bruce. & M. Veloso. (2003). Fast and Accurate Vision-Based Pattern Detection and Identification, Proceedings of ICRA-03, the 2003 IEEE International Conference on Robotics and Automation (ICRA'03), May, 2003.

S. Hibino.; Y. Kodama.; Y. Nagasaka.; T. Takahashi.; K. Murakami. & T. Naruse. (2002). Fast Image Processing and Flexible Path Generation System for RoboCup Small Size League, RoboCup 2002: Robot Soccer World Cup VI, Springer, pp. 53-64, 2002.

H. Sakoe. & S. Chiba. (1978). Dynamic Programming Algorithm Optimization for Spoken Word Recognition, IEEE Trans. Acoust., Speech, and Signal Process., Vol.ASSP-26, pp. 43-49, 1978.

21

Behavior Acquisition in RoboCup Middle Size League Domain

Yasutake Takahashi and Minoru Asada
Osaka University
Japan

1. Introduction

The RoboCup middle size league is one of the leagues that have the longest histories in RoboCup. This league has unique features, for example, bigger robots (around 45cm square) plays on the largest field (say, 18m×12m in 2007), any global sensory system is not allowed to use, all robots have on-board vision systems and controllers. Each robot plays based on its own sensory information, and it can share some information with teammates and a coach box located outside the playing field over wireless communication, then, shows some cooperative behaviors among them during the game.

This chapter briefly introduces research activities in RoboCup middle size league. A variety of research topics have been attacked in this league. Some of them are common to other real robot leagues such as small size and 4-legged leagues. For example, robust real-time on-board vision system, precise localization based on vision system, and design of cooperative behavior are actively investigated in RoboCup middle size league. On the other hand, skill and cooperative/competitive behavior acquisition/emergence based on machine learning techniques is also well-studied. The latter is focused on in this chapter.

First, a purposive behavior acquisition of a single robot based on machine learning technique is introduced. Reinforcement learning is one of machine learning techniques and extensively studied to be applied to acquisition of robot behavior like shooting a ball into a goal. It has a simple framework and algorithm to be applied to robots however some difficulties exist in practical use because of its simplicity. In order to overcome these problems, some modular learning and hierarchical systems have been proposed. Not only reinforcement learning but also evolutional methods have been investigated as well. Some examples will be shown.

Next, studies on cooperative/competitive behavior acquisition based on machine learning techniques are introduced. Application of machine learning to multi-agent system usually has some difficulties because of complex dynamics of the system. The complexity is induced by decision making by multi-players, growing amount of information to decide an action by an individual, perceptual aliasing, and so on. In order to reduce the complexity, wireless communication between teammates is commonly used. In case of unavailability of communication between players, for example, lack of communication with opponents,

methods of situation estimation and acquisition of appropriate behavior in the estimated situation has been proposed. Finally, discussions and future work are given.

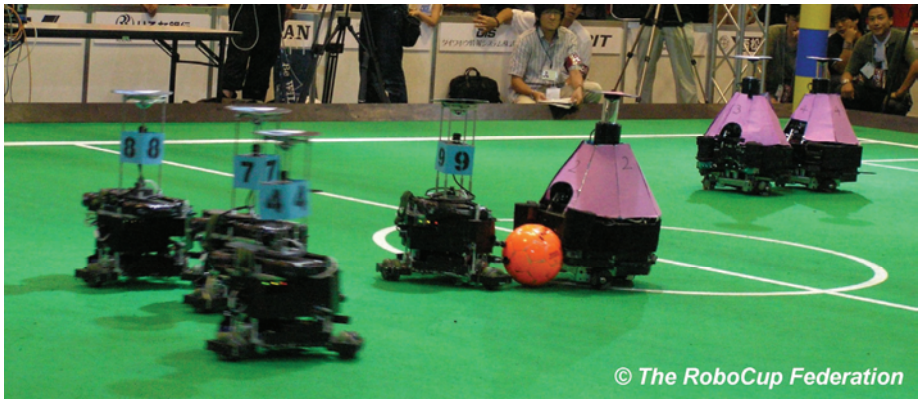


Fig. 1. A game scene of middle size league from RoboCup2005

2. RoboCup Middle Size League

Robots of no more than 50 cm square play soccer in teams of up to 6 robots with an orange soccer ball on a field whose size is 18×12 meters. Matches are divided in 15-minute halves. Overhead cameras or external sensors around a field are forbidden in this league. Almost robots have omni-directional cameras and some of them have additional ordinary cameras in order to detect a ball and localize themselves on the field. A referee box system is introduced for conveying referee decisions to robot players without interception of operators from the teams. It successfully enhances the autonomy of the game. Robots have omni-directional vehicles with 3 or 4 omni-wheels and maximum speed of robots is up to 4 m/sec, then, precise motion control and localization are necessary. All robots have ball kicking devices and speed of a kicked ball is up to 11 m/sec. The devices can kick a ball upward, therefore, fast and better image understanding and price control are required especially for a goalie to prevent a loop shoot. Because the size of the field has become larger and larger, more cooperative behaviors among teammates should be taken in a game.

3. Behavior Acquisition Through Trials and Errors

We briefly review reinforcement learning scheme for various behavior acquisition/executions first then introduce some examples of acquisition of basic skills.

3.1 Layout of Manuscript

Fig.2 shows a basic framework of reinforcement learning. An agent can discriminate a set S of distinct world states. The world is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the agent based on a policy π.

The agent receives reward r_t at each step t. State value V^π, the discounted sum of the reward received over time under execution of policy π, will be calculated as follows:

$$V^\pi = \sum_{t=0}^{\infty} \gamma^t r_t \tag{1}$$

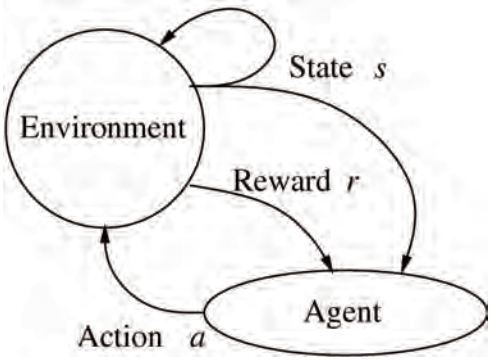


Fig. 2. Agent-environment interaction

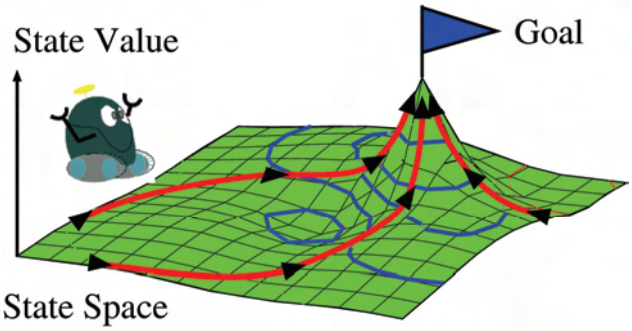


Fig. 3. Sketch of state value function

Fig.3 shows a sketch of a state value function where a robot receives a positive reward when it stays at a specified goal while zero reward else. The state value becomes highest at the state where the agent receives a reward and discounted value is propagated backward to the most recent states.

The state value increases if the agent follows a good policy π. The agent updates its policy through the interaction with the environment in order to receive higher positive rewards in future. Analogously, as animals get closer to former action sequences that led to goals, they are more likely to retry it. For further details, please refer to the textbook of Sutton and Barto(Sutton and Barto, 1998) or a survey of robot learning(Connell and Mahadevan, 1993).

3.2 Basic Behavior Acquisition

As an early study of applications of reinforcement learning techniques to a real soccer robot, acquisition of shooting behavior is investigated (for example, (Asada et al., 1996)). Recent investigation has been continued by teams. For example, Brainstormer-Tribot team has

applied reinforcement learning to dribbling behavior of a real robot without any computer simulation based on physical world and robot models, and the robot acquires an appropriate motion for the behavior within reasonable time.
Asada et al.(Asada et al., 1996) presented a method of vision-based reinforcement learning by which a robot learns to shoot a ball into a goal. Several issues in applying the reinforcement learning method to a real robot with vision sensor by which the robot can obtain information about the changes in an environment were discussed. A state space was constructed in terms of size, position, and orientation of a ball and a goal in an image of an ordinary camera (shown in Fig.4) and an action space is designed in terms of the action commands to be sent to the left and right motors of a vehicle. In order to speed up the learning time, a mechanism of Learning from Easy Missions (or LEM) is implemented.
LEM reduces the learning time from exponential to almost linear order in the size of the state space. At this moment, the behavior was acquired in a soccer simulator on a workstation and the acquired behavior is implemented on a real robot.

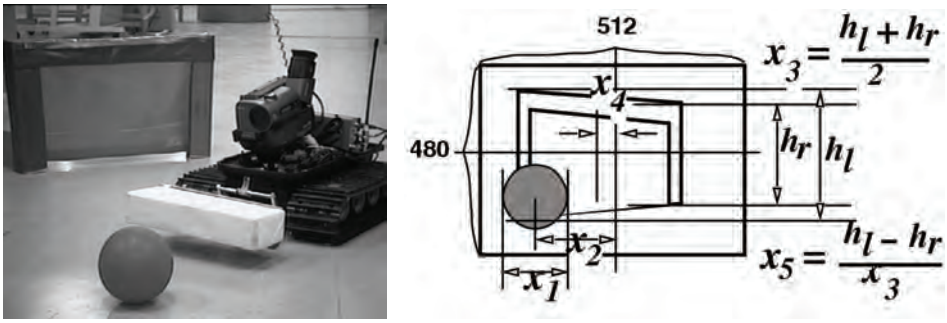


Fig. 4. A picture of the robot and image features composing an input vector

3.3 State Space Construction

Reinforcement learning has been investigated as a method for robot learning with little or no a priori knowledge and higher capability of reactive and adaptive behaviors. However, there are two major problems in applying it to real robot tasks: how to construct a state space, and how to reduce the learning time. Robot learning such as reinforcement learning generally needs a well-defined state space in order to converge. However, to build such a state space is one of the main issues of the robot learning because of the inter-dependence between state and action spaces, which resembles to the well known "chicken and egg" problem.
Asada et al.(Asada et al., 1995) proposed a method of robot learning by which a set of pairs of a state and an action are constructed to achieve a goal. A state is defined as a cluster of input vectors[1] from which the robot can reach the goal state or the state already obtained by a sequence of one kind action primitive regardless of its length, and that this sequence is defined as one action. The input vectors are clustered as hyper ellipsoids so that the whole state space is segmented into a state transition map in terms of action from which the optimal action sequence is obtained (see Fig.4).

[1] An input vector usually consists of the sensory information. An example is shown at the right side of Fig.4.

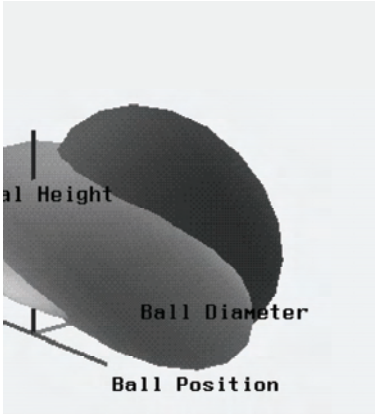


Fig. 5. An example of state space construction

Takahashi et al (Takahashi et al., 1996a,b) presented a method by which a robot learns a purposive behavior within less learning time by incrementally segmenting the sensor space based on the experiences of the robot. The incremental segmentation is performed by constructing local models in the state space, which is based on the function approximation of the sensor outputs to reduce the learning time, and on the reinforcement signal to emerge a purposive behavior. The method is applied to a soccer robot that tries to shoot a ball into a goal. The experiments with computer simulations and a real robot were carried out. As a result, a real robot has learned a shooting behavior within less than one hour training by incrementally segmenting the state space.
Uchibe et al. (Uchibe et al., 1998a,b; Asada et al., 1998, 1999) proposed a method that estimates the relationships between learner's behaviors and other agents' ones in the environment through interactions (observation and action) using the method of system identification. In order to identify the model of each agent, Akaike's Information Criterion is applied to the results of Canonical Variate Analysis for the relationship between the observed data in terms of action and future observation. Then, reinforcement learning based on the estimated state vectors is performed to obtain the optimal behavior. The proposed method is applied to a soccer playing situation, where a rolling ball and other moving agents are well modeled and the learner's behaviors are successfully acquired by the method.

3.4 Towards Complex Behavior Acquisition

A simple and straightforward application of reinforcement learning methods to complex behavior acquisition is considerably difficult due to its almost endless exploration of which time easily scales up exponentially with the size of the state/action spaces, which seems almost impossible from a practical viewpoint. One of the potential solutions might be application of so-called modular learning and multi-layered system in which a set of expert modules learn and one gating system weights the output of the each expert module for the final system output. This idea is very general and has a wide range of applications. Stone and Veloso (Stone and Veloso, 1998) has proposed to introduce layered learning system

with basic skills such as “shootGoal”, “shootAway”, “dribbleBall”, and so on. Kleiner et al (Kleiner et al., 2002) has also proposed multi-layered learning system for behavior acquisition of a soccer robot. Their experimental results show that the performance of the acquired behavior learned by lower and higher modules simultaneously is better than the one of the behavior that lower and higher modules are trained separately. However, we have to consider the following two issues to apply it to the real robot tasks:

- Task decomposition: how to find a set of simple behaviors and assign each of them to a learning module or an expert in order to achieve the given initial task. Usually, human designer carefully decomposes the long time-scale task into a sequence of simple behaviors such that the one short time-scale subtask can be accomplished by one learning module.
- Abstraction of states and/or actions for scaling up: To accomplish a complex behavior, much sensory information is taken into account and the exploration space for learning behavior based on the information easily becomes huge. In order to cope with complicated real robot tasks, more abstraction of the states and/or actions is necessary.

A basic idea to cope with the above two issues is that any learning module has a limited resource constraint and this constraint of the learning capability leads us to introduce a multi-module and multi-layered learning system. That is, one learning module has a compact state-action space and acquires a simple map from the states to the actions, and a gating system enables the robot to select one of the behavior modules depending on the situation. More generally, the higher module controls the lower modules depending on the situation. The definition of this situation depends on the capability of the lower modules because the gating module selects one of the lower modules based on their acquired behaviors. From the other viewpoint, the lower modules provide not only the rational behaviors but also the abstracted situations for the higher module; how feasible the module is, how close to its subgoal, and so on. It is reasonable to utilize such information in order to construct state/action spaces of higher modules from already abstracted situations and behaviors of lower ones. Thus, the hierarchical structure can be constructed with not only experts and gating module but also more layers with multiple homogeneous learning modules.

Takahashi and Asada (Takahashi and Asada, 2000) proposed self-construction of hierarchical structure with purely homogeneous learning modules. Since the resource (and therefore the capability, too) of one learning module is limited, the initially given task is automatically decomposed into a set of small subtasks each of which corresponds to one of the small learning modules, and also the upper layer is recursively generated to cover the whole task. In this case, the all learning modules in the one layer share the same state and action spaces although some modules need only the part of them.

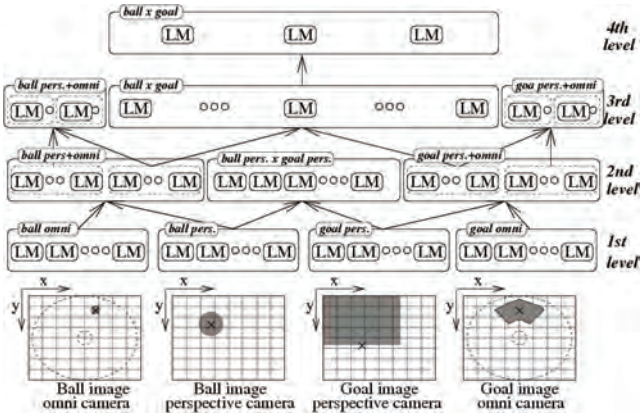


Fig. 6. A multi-layered learning system for vision based behaviors (Takahashi and Asada, 2001)

Their following work (Takahashi and Asada, 2001, 2003) focused on the state and action space decomposition according to the subtasks to make the learning much more efficient. Further, Takahashi et al, (Takahashi et al., 2003b, 2005c; Nishi et al., 2006) realized unsupervised decomposition of a long time-scale task by finding the compact state spaces, which consequently leads the subtask decomposition.

4. Cooperative/Competitive Behavior

Cooperative/competitive behavior realization is one of the most interested topics in RoboCup community. Peter stone et al.(Stone et al., 2005) proposed "keep away" task in RoboCup simulation league and many investigations have been done on the task (for example, (Kuhlmann and Stone, 2004; Taylor and Stone, 2004; Stone et al., 2005)). The topic also interests people participating in RoboCup middle size league. In this section, related works are briefly introduced.

4.1 Cooperation Via Environmental Dynamics

Cooperation is one the most important issues in multiagent systems. There is a trade-off between the centralized control and the distributed one from the performance viewpoint of cooperation. Takahashi et al.(Takahashi et al., 2001) proposed a method to emerge cooperative behaviors via environmental dynamics caused by multi robots in a hostile environment without any planning for cooperation. Each robot has its own policy to achieve the goal with/without explicit social behavior such as yielding. Co-existence of such robots in a dynamic, hostile environment produces various environmental dynamics, in which the heterogeneous robots can be seen as cooperating each other. Fig.7 shows an example of how the two robots recover each others' failures quickly. Two type robots, A and B, were prepared in this case. Type A robot is selfish, skillful and careful to shoot a ball. On the other hand, Type B is moderate and not so skillful but has a much fast shooting behavior. (1) indicates that the two different robots follow a ball. Type B robot tries to shoot a ball to the opponent goal at (2). But it failed at (3) because the ball handling skill of type B is not so good, and type A robot recovers the failure soon. Type A robot tries to shoot the ball, but the

opponent goalie defends it at (4). Type A robot tries to shoot the ball from left side of the goal at (5) and (6), but unfortunately fails again while type B robot moves its position behind type A robot. Type B robot tries to recover the failure of type A robot's shooting at (7), and it shoots the ball successfully after all at (8).

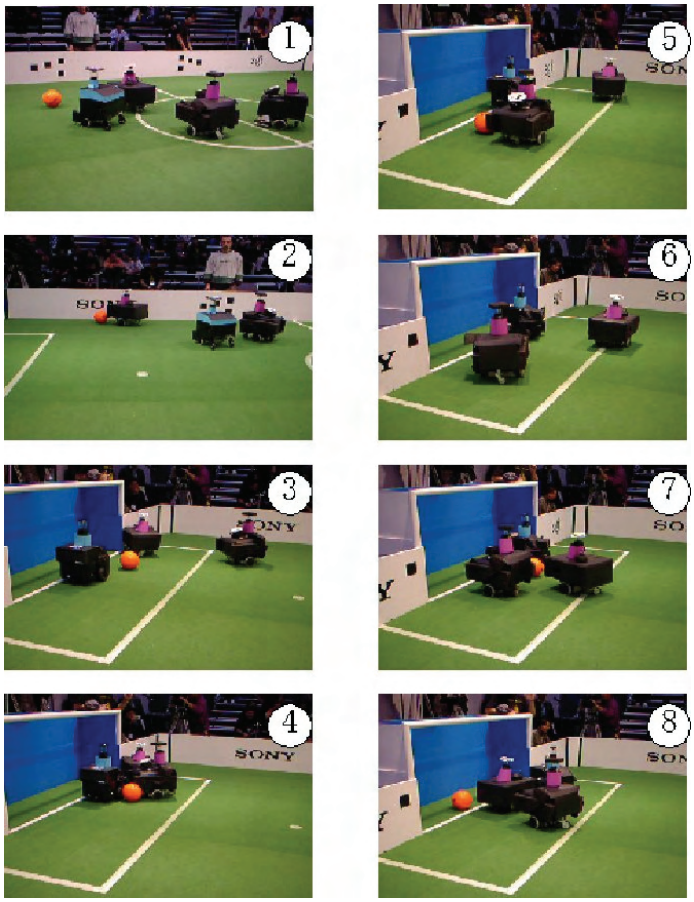


Fig. 7. A sequence of a failure recovery behavior among two robots

4.2 Strategy Learing for a Team

Team strategy acquisition is one of the most important issues of multiagent systems, especially in an adversary environment. RoboCup has been providing such an environment. A deliberative approach to the team strategy acquisition seems to be difficult for applying in such a dynamic and hostile environment. Takahashi et al.(Takahashi et al., 2002b) presented a learning method to acquire team strategy from a viewpoint of coach who can change a combination of players each of which has a fixed policy. Assuming that the opponent has the same choice for the team strategy but keeps the fixed strategy during one match, the coach estimates the opponent team strategy (player's combination) based on game progress

(obtained and lost goals) and notification of the opponent strategy just after each match. The trade-off between exploration and exploitation is handled by considering how correct the expectation in each mode is. A case of 2 to 2 match was simulated and the final result (a class of the strongest combinations) was applied to RoboCup-2000 competition.

4.3 Emergence of Cooperative Behavior Though Co-evolution

Co-evolution has been investigated as a method for multi agent simultaneous learning. Uchibe et al.(Uchibe et al., 1998c,d; Uchibe and Asada, 2006) discussed how multiple robots can emerge cooperative behaviors through co-evolutionary processes. As an example task, a simplified soccer game with three learning robots is selected and a GP (genetic programming) method is applied to individual population corresponding to each robot so as to obtain cooperative and competitive behaviors through evolutionary processes. The complexity of the problem can be explained twofold: co-evolution for cooperative behaviors needs exact synchronization of mutual evolutions, and three robot co-evolution requires well-complicated environment setups that may gradually change from simpler to more complicated situations so that they can obtain cooperative and competitive behaviors simultaneously in a wide range of search area in various kinds of aspects.

4.4 Dynamic Roll Assignment Based on Module Conflict Resolution

It is necessary to coordinate multiple tasks in order to cope with larger-scaled and more complicated tasks. However, it seems very hard to accomplish the multiple tasks at the same time. Uchibe et al.(Uchibe et al., 2001) proposed a method to resolve a conflict between task modules through the processes of their executions. Based on the proposed method, the robot can select an appropriate module according to the priority. In addition, they applied the module conflict resolution to a multiagent environment. Consequently, multiple tasks are automatically allocated to the multiple robots.

4.5 Coping with Behavior Alternation of Others

Existing reinforcement learning approaches have been suffering from policy alternation of others in multi-agent dynamic environments that may cause sudden changes in state transition probabilities of which constancy is needed for behavior learning to converge. A typical example is the case of RoboCup competitions because behaviors of other agents may change the state transition probabilities. The keys for simultaneous learning to acquire competitive behaviors in such an environment are

- a modular learning system for adaptation to the policy alternation of others; and
- an introduction of macro actions for simultaneous learning to reduce the search space.

Takahashi et al.(Takahashi et al., 2005a,b; Edazawa et al., 2004; Takahashi et al., 2003a, 2002a) presented a method of modular learning in a multi-agent environment in which the learning agents can simultaneously learn their behaviors and adapt themselves to the resultant situations by the others' behaviors.

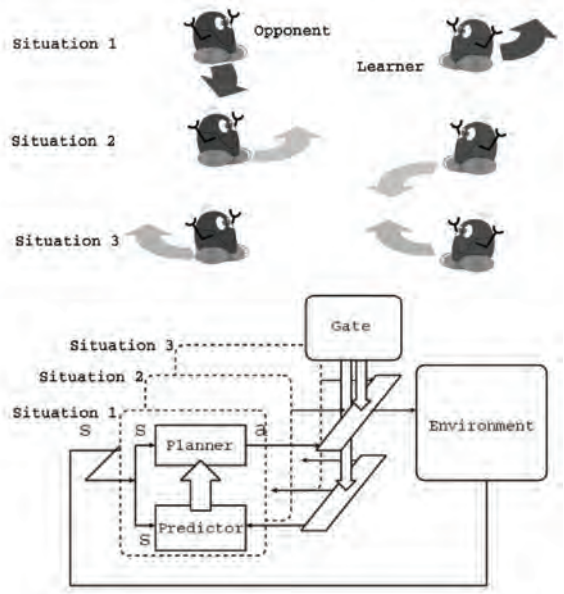


Fig. 8. A multi-module learning system in multi-agent environment

4.6 Behavior Based on Estimation of Status of Others

The existing reinforcement learning approaches have been suffering from the curse of dimension problem when they are applied to multiagent dynamic environments. The keys for learning to acquire cooperative/competitive behaviors in such an environment are as follows:

- a two-layer hierarchical system with multi learning modules is adopted to reduce the size of the sensor and action spaces. The state space of the top layer consists of the state values of the individual modules at the lower level that indicate how close to the goals, and the macro actions are used to reduce the size of the physical action space, and further,
- other's state estimation modules by observation are added in order to estimate to what extent the other agent task has been achieved and the estimated state values are used in the top layer state space to accelerate the cooperative/competitive behavior learning.

Takahashi et al. (Takahashi et al., 2006) showed a method of modular learning involving the above two issues, by which the learning agent can acquire cooperative behaviors with its team mates and competitive ones against its opponents. The method is applied to 4 on 5 passing task, and the learning agent successfully obtained the desired behaviors.

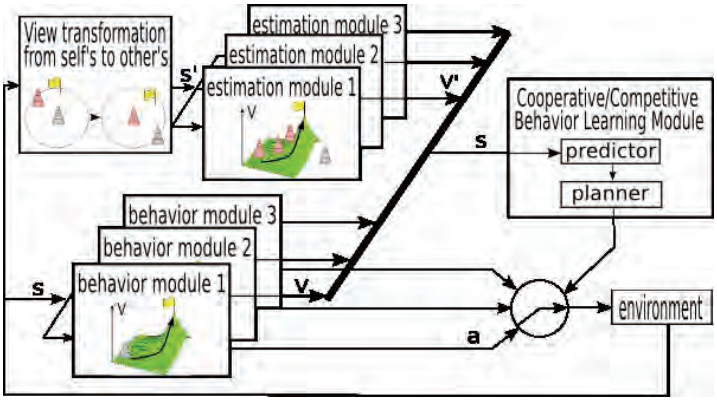


Fig. 9. A hierarchical system for behavior acquisition in multi-robot environment

5. Discussion and Future Work

This chapter briefly overviewed research activities, especially on behavior acquisition/ emergence based on machine learning techniques, in RoboCup middle size league. This research area has kept attracting people and been investigated not only in middle size league but also other ones such as simulation soccer and 4-legged leagues. Many results of applications of machine learning techniques to robots in the RoboCup domain show promising contributions to generate adaptive behaviors in real situations. On the other hand, many difficulties in practical use have also been unveiled so far. For example, selection of important features for purposeful behaviors, purposive behavior discovery through observation of others, self task decomposition and integration, rapid team strategy adaptation during a game, and so on, are to be investigated furthermore. One of the goals of RoboCup is "By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team."(Federation) A game of a middle size league robot team v.s. a human team was demonstrated in RoboCup2007 Atlanta USA. The human team showed much better performance than the robot team although it won the championship in the middle size league this year. Taking achievements in the past decade into consideration, however, we foresee that robots will play soccer with human players, learn many skills, cooperative/competitive behaviors, team coordination, positioning in the teams, fast adaptation of team strategy, and so on through interaction during games, and finally a robot team beats the human world soccer champion team.

Fig. 10. A demonstration game scene of robot and human teams

6. References

Asada, M., Hosoda, K., and Suzuki, S. (1998). Vision-based learning and development for emergence of robot behaviors. In Shirai, Y. and Hirose, S., editors, *Robotics Research, The Seventh International Symposium*, pages 327–338. Springer.

Asada, M., Noda, S., and Hosoda, K. (1995). Non-physical intervention in robot learning based on lfe method. *In Proc. of Machine Learning Conference Workshop on Learning from Examples vs. Programming by Demonstration*, pages 25–31.

Asada, M., Noda, S., Tawaratumida, S., and Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303.

Asada, M., Uchibe, E., and Hosoda, K. (1999). Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence*, 110:275–292.

Connell, J. H. and Mahadevan, S. (1993). *ROBOT LEARNING*. Kluwer Academic Publishers.

Edazawa, K., Takahashi, Y., and Asada, M. (2004). Modular learning system and scheduling for behavior acquisition in multi-agent environment. *In RoboCup 2004 Symposium papers and team description papers*, pages CD–ROM.

Federation, T. R. Robocup. http://www.robocup.org/. Kleiner, A., Dietl, M., and Nebel, B. (2002). Towards a life-long learning soccer agent. In Kaminka, G. A., Lima, P. U., and Rojas, R., editors, *The 2002 International RoboCup Symposium Pre-Proceedings*, pages CD–ROM.

Kuhlmann, G. and Stone, P. (2004). Progress in learning 3 vs. 2 keepaway. In Polani, D., Browning, B., Bonarini, A., and Yoshida, K., editors, *RoboCup-2003: Robot Soccer World Cup VII*. Springer Verlag, Berlin.

Nishi, T., Takahashi, Y., and Asada, M. (2006). Incremental purposive behavior acquisition based on modular learning system. In Arai, T., Pfeifer, R., Balch, T., and Yokoi, H., editors, *Intelligent Autonomous Systems 9*, pages 702–712. IOS Press. ISBN 1-58603-595-9.

Stone, P., Sutton, R. S., and Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*. Vol. 13, No. 3, 165-188 (2005)

Stone, P. and Veloso, M. (1998). Layered approach to learning client behaviors in the robocup soccer server. Applied Artificial Intelligence, 12(2-3). Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Takahashi, Y. and Asada, M. (2000). Vision-guided behavior acquisition of a mobile robot by multi-layered reinforcement learning. *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 395–402.

Takahashi, Y. and Asada, M. (2001). Multi-controller fusion in multi-layered reinforcement learning. *In International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI2001)*, pages 7–12.

Takahashi, Y. and Asada, M. (2003). Multi-layered learning systems for vision based behavior acquisition of a real mobile robot. *In Proceedings of SICE Annual Conference 2003 in Fukui*, volume CD-ROM, pages 2937–2942.

Takahashi, Y., Asada, M., and Hosoda, K. (1996a). Reasonable performance in less learning time by real robot based on incremental state space segmentation. *In Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS '96)*, pages 1518–1524.

Takahashi, Y., Asada, M., Noda, S., and Hosoda, K. (1996b). Sensor space segmentation for mobile robot learning. *In Proceedings of ICMAS'96 Workshop on Learning, Interaction and Organizations in Multiagent Environment.*

Takahashi, Y., Edazawa, K., and Asada, M. (2002a). Multi-module learning system for behavior acquisition in multi-agent environment. *In Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages CD–ROM 927–931.

Takahashi, Y., Edazawa, K., and Asada, M. (2003a). Behavior acquisition based on multi-module learning system in multi-agent environment. In Kamink, G. A., Lima, P. U., and Rojas, R., editors, *RoboCup 2002: Robot Soccer World Cup VI*, pages 435–442. Springer. LNAI 2752 ISBN 3-540-40666-2.

Takahashi, Y., Edazawa, K., Noma, K., and Asada, M. (2005a). Simultaneous learning to acquire competitive behaviors in multi-agent system based on a modular learning system. *In Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 153–159.

Takahashi, Y., Edazawa, K., Noma, K., and Asada, M. (2005b). Simultaneous learning to acquire competitive behaviors in multi-agent system based on modular learning system. *In RoboCup 2005 Symposium papers and team description papers*, pages CD–ROM.

Takahashi, Y., Hikita, K., and Asada, M. (2003b). Incremental purposive behavior acquisition based on self-interpretation of instructions by coach. *In Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 686–693.

Takahashi, Y., Kawamata, T., and Asada, M. (2006). Learning utility for behavior acquisition and intention inference of other agent. *In Proceedings of the 2006 IEEE/RSJ IROS 2006 Workshop on Multi-objective Robotics*, pages 25–31.

Takahashi, Y., Nishi, T., and Asada, M. (2005c). Self task decomp osition for modular learning system through interpretation of instruction by coach. *In RoboCup 2005 Symposium papers and team description papers*, pages CD–ROM.

Takahashi, Y., Tamura, T., and Asada, M. (2001). Cooperation via environmental dynamics caused by multi robots in a hostile environment. *In The Fourth IFAC Symposium on Intelligent Autonomous Vehicles*, pages 413–418.

Takahashi, Y., Tamura, T., and Asada, M. (2002b). Strategy learning for a team in adversary environments. In Birk, A., Coradeschi, S., and Tadokoro, S., editors, *RoboCup 2001: Robot Soccer World Cup V*, pages 224–233. Springer. ISBN 3-540-43912-9.

Taylor, M. E. and Stone, P. (2004). Speeding up reinforcement learning with behavior transfer. *In AAAI 2004 Fall symposium - Real Life Reinforcement Learning session*.

Uchibe, E. and Asada, M. (2006). Incremental coevolution with competitive and cooperative tasks in a multirobot environment. *In Proceedings of the IEEE*, volume 94, pages 1412–1424.

Uchibe, E., Asada, M., and Hosoda, K. (1998a). Cooperative behavior acquisition in multi mobile robots environment by reinforcement learning based on state vector estimation. *In Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 1558–1563.

Uchibe, E., Asada, M., and Hosoda, K. (1998b). State space construction for behavior acquisition in multi agent environments with vision and action. *In Proc. of International Conference on Computer Vision*, pages 870–875.

Uchibe, E., Kato, T., Asada, M., and Hosoda, K. (2001). Dynamic task assignment in a multiagent/multitask environment based on module conflict resolution. *In Proc. of IEEE International Conference on Robotics and Automation*, pages 3987–3992.

Uchibe, E., Nakamura, M., and Asada, M. (1998c). Co-evolution for cooperative behavior acquisition in a multiple mobile robot environment. *In Proc. Of IEEE/RSJ International Conference on Intelligent Robots and Systems 1998 (IROS '98)*, pages 425–430.

Uchibe, E., Nakamura, M., and Asada, M. (1998d). Cooperative behavior acquisition in a multiple mobile robot environment by co-evolution. In Asada, M., editor, *RoboCup-98: Robot Soccer World Cup II, Proc. of the second RoboCup Workshop*, pages 237–250.

22

Multicriterial Decision-making Control of the Robot Soccer Team

Petr Tucnik
University of Hradec Kralove
Czech Republic

1. Introduction

This text is a summarization of a series of shorter articles published about this topic (and related problems) during the years 2005-2007 (E.g. Tucnik et al. 2006a, Tucnik et al. 2006b). In previous articles we have been commonly using the term "agent" when referring to the MiroSot robots. We will continue in this practice in this text as well. We believe that there is no conflict in using this term because the idea of autonomous entity (in the way the agent is usually defined, see below) is well-corresponding with the role of the MiroSot player. Therefore, whenever the term of "agent" or "robot" or "player" is used, we are referring to the same entity. Any exceptions will be explicitly mentioned.
The principles of the robot soccer game are well-known, but it is usually a good idea to begin with the basics. Comprehensive description of all rules and specifications is available at www.fira.net (FIRA – Federation of International Robot-soccer Association). Therefore, only basic information will be provided, when we feel it is necessary or important. We will be dealing with the MiroSot (micro-robot soccer tournament) Middle League (5vs5 players). This league of the tournament is played on the pitch of the size of 220cm x 180cm, robots must be smaller than 7.5cm x 7.5cm x 7.5cm and the robot's weight must not exceed 650g.
Together with other categories of the robot soccer (HuroCup, KheperaSot, NaroSot, AndroSot, RoboSot and SimuroSot, for details see the same webpage as in the above paragraph), the MiroSot league also serves a certain purpose: "*MiroSot initiative gives a good arena for multi-agent research, dealing with research subjects such as cooperation protocol by distributed control, effective communication and fault tolerance, while having efficiency of cooperation, adaptation, robustness and being real-time.*" (citation taken from http://www.fira.net/about/overview.html, date 20th July 2007). We will try to follow this aim, as it was defined by FIRA, in our work. Illustrational photograph of the MiroSot robot is presented at the Fig. 1. During the game, there is always a coloured patch on the topside of the robot. This colour patch will identify the robots in the team.
This chapter will be focused on the control mechanism and decision-making issues only. We are not interested in image processing or data transmission problems. We have decided that it is no purpose of this text to discuss these matters. Therefore, we will dare to assume that these parts of the system work without any errors or malfunctions and we are obtaining all the data needed.

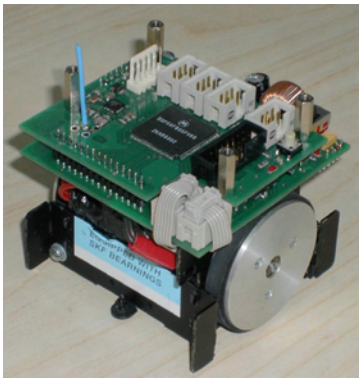


Fig. 1. A MiroSot robot

For the both strategic (team-oriented) and tactical (individual) control of the team, we have decided to implement the multicriterial decision-making (MDM) principle. The more common approach to the robot soccer problem may be found in (Kim et al., 2004). The multicriterial decision-making approach has an excellent adaptability potential but requires perception of the whole game and its environment from another perspective. From the agent's point of view, the environment is perceived as a set of (more or less important, see below) influences and the agent is continually trying to find the most appropriate, the most profitable reaction to the given situation. The description of the MDM algorithm is presented in the third part of this chapter.

2. Terminology and Basic Definitions

The use of multicriterial approach requires a definition of terms that will be used during the explanation of its principles. Explanation of the general MDM algorithm will be provided here (its application will be presented for better understanding on the goalkeeper example in the part 9).

The time given for the decision-making process (DMP) is strictly limited by the time of image processing speed. Generally, this is the most time-demanding part of the whole decision-making process. Therefore, all time-subordinate tasks have to be done in the given time frame. The image processing speed is quoted in frames per second (FPS) and the higher this value is, the better. It may happen that it is a team that has a better reaction time which wins over the team with better strategy, because it is capable of faster reactions. It is an utmost necessity to try to keep image processing speed as fast as possible. The time frame given by the image processing speed and cycle of activities performed during the decision-making process together define the **iteration of the game**. In other words, the iteration is a complete set of activities beginning with image processing tasks, continuing with situation analysis, action, role and task assignment and instruction processing and ending with the transmission of the instructions to the robot, all in the given time frame. E.g. when we have reached the speed of 50FPS in image processing tasks, then we have to be able to make all strategic and tactical decisions in 1/50 [sec] to keep up pace with the rest of the system. Any delay would cause a serious trouble.

2.1 Attributes

The environment is perceived by the agent as a finite set of attributes. The **attribute** is a numerically represented and limited (see below) description of the measure of presence of a certain characteristic of the environment. E.g. a position of the agent on the playground's grid is an attribute of the environment. However, the attributes' domain is not limited to the "tangible" information only. Any information that has a descriptive value may be perceived as an attribute, if we are able to determine its highest and lowest value possible. For analysis purposes, it may be useful to divide the attributes into four groups, according to their descriptive domain, as it is presented at the Fig. 2.

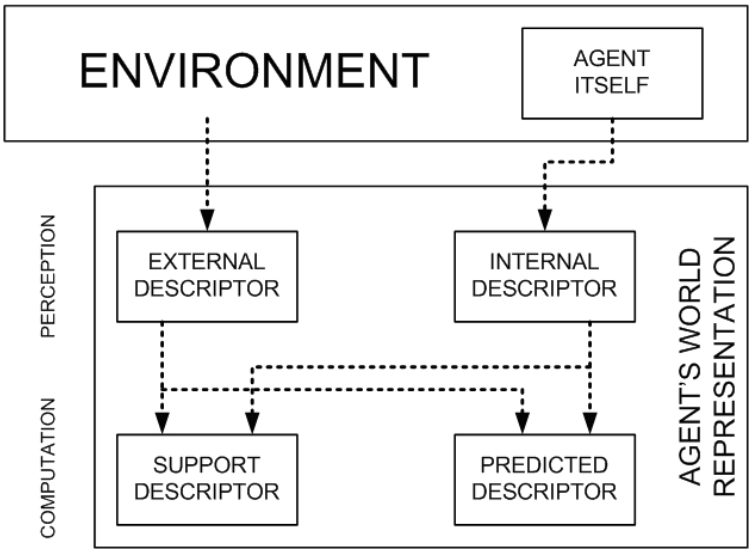


Fig. 2. Attributes are divided into four groups

The agent is an integral part of the environment for both itself and other agents. Other agents are taken as a part of the environment. The agent's representation of the world is formed by the four indicated types of descriptors. When the agent is obtaining an attribute's value, either perception (sensors) or computation is used during the process.

The **external descriptors** are attributes used to describe characteristics of the real-world environment. E.g. speed of other robots, position of the ball (or robot) on the playground, etc. This type of information provides the cornerstone for DMP for the derived descriptors are based on it. During the game, this type of information has to be *reliably* provided by the image processing module (see part 5 – Strategic Control Module). In other words, this is the reliable base for decisions to be made. If we are unable to satisfy the reliability requirement and do not have a set of solid facts which we may use for decision-making, than all decision-making effort is futile. The "real-world environment" expression, used at the beginning of this paragraph, is valid for the environment of the MiroSot game. For the purposes of SimuroSot, the formulation "software environment" should be used and so on, depending on the problem domain.

The **internal descriptors** are used to describe internal states of the agent. E.g. velocities of its wheels, battery level, etc. For our problem domain, they usually have only limited impact on the DMP. E.g. the velocities of the robot's wheels are, in fact, output of the DMP. Also, the other internal characteristics of the agent are designed to fulfil the needs of the robot soccer game perfectly and therefore are not so important. But they may play an important role when we are trying to apply this decision-making method to another type of problem (other than robot soccer). In certain cases, the internal states of the agent may have a critical decision-making importance. This type of attributes has to be *reliable* as well.

The **support descriptors** are based on the real-world information (both external and internal nature) and are derived from it. Support descriptors are category of attributes computed on the basis of reliable data from the environment and agent itself. They are used for supplemental calculations (if they are needed) when planning movement, collisions, turning, etc. The realization of many tasks depends on these support calculations.

The **predicted descriptors** are used to take the effect of the action into account when making decision or planning. The predicted position of the robot is an example of this type of descriptor. This type of (predicted) information is, in general, *unreliable* ("Is unreliable" or "may be unreliable" – it is depending on the given attribute. Own actions, for instance, may be predicted precisely, actions of opponent's agents very imprecisely, as we have no way how to affect them directly.) to a certain degree, but is important for making decisions, that needs to reflect possible impact of an action. Prediction has gradually lower reliability as we try to predict further in future. When the speed of the game is taken into account, it is reasonable to predict up to maximum (rather less) time of approximately 1-2 [sec] (depending on the both team's capabilities). Prediction of more distant future is useless and may lead to incorrect conclusions in the DMP. Precise time interval of applicable prediction is a question of extensive testing and image processing speed.
It is important to stress up the fact that we are always dealing with a certain degree of imprecision in the real-world cases. This is caused by the imprecision of our sensors used for monitoring of attributes, wrong calibrations, rounding of numbers in calculations and other undesirable influences. There are several major sources of uncertainty in the environment (Decker, 1995): (1) uncertainty of the environment – monitoring of environment by sensors is made with imprecision. It may happen that the agent is unable to recognize the ball or other players on the pitch for a moment; (2) uncertainty of other agents – the opponent's agents next actions are not certain (also it is possible that hardware malfunction occurs in the own team); (3) uncertainty of actions – the actuator apparatus of the agent may fail to perform the action as planned.
Nevertheless, it is possible to keep these effects in reasonable boundaries. Detection of failures and consequent actions has to be implemented in the decision-making mechanism. However, for the purposes of this text we will avoid further discussion about the failure detection and handling, for these are complicated problems unrelated to the decision-making and control issues which are our main aim. Much useful information regarding the topic of failure detection and solving may be found in (Aguilera et al., 1997; Byrne & Edwards, 1996; Kit, 1995; Menzies, 1999; Tichý, 2003).
All four types of the descriptors are only special types of attributes. The agent's internal concept of the world is formed by all four types of attributes. This separation into groups is

useful for analysis purposes only and apart from the reliability of the information value there is no other influence on the DMP. The whole set of all attributes A used in the DMP may be formally described:

$$A = (a_1,...,a_z),\ \forall a_i \in A : Dom(a_i) \in \langle a_{i\min}, a_{i\max} \rangle, -\infty < a_{i\min} < a_{i\max} < \infty \tag{1}$$

A is a set of all attributes and every attribute has a finite minimal and maximal value. Every attribute is normalized before it is used in DMP. The following formula is used for normalization:

$$norm(a_i) = \left| \frac{a_i - a_{i\min}}{a_{i\max} - a_{i\min}} \right|,\ a_i, a_{i\min}, a_{i\max} \in R \tag{2}$$

After application of the *norm* function for each attribute, a following is valid:

$$norm(a_i) \in \langle 0;1 \rangle \tag{3}$$

Therefore, a comparison of attributes of different units of measurement is possible. After normalization, no measurement units are used and normalized attribute value is just a number.

2.2 Environment

The definition of the attribute and its properties was discussed in the part 2.1. The environment is defined by a limited set of attributes. The whole environment has to be perceptible by an agent (in case of derived descriptors, the agent has to be able to compute them). Attributes that are not perceptible by an agent may be omitted for they have no influence on the DMP. From the agent's point of view, the environment E may be formally defined as:

$$E = (norm(A)) \tag{4}$$

Therefore, the agent is working with normalized attributes when making a decision. Although the set of all attributes A was defined by (1), we present following specification for better understanding (these sub-sets are representing types of descriptors):

$$A = A_{EXTERNAL} \cup A_{INTERNAL} \cup A_{SUPPORT} \cup A_{PREDICTED} \tag{5}$$

An important role in the DMP plays time. The temporal aspect of DMP may not be omitted for the effects of the actions have to be taken into account. For these purposes a **configuration** C is defined by the following formula:

$$C^T = \left(a_1^T, \ldots, a_z^T\right) \tag{6}$$

T stands for the game iteration (time) index. The configuration C of the game represents a state of all attributes (their values) in the given moment. The time index will always be positioned in superscript position in the following text.

Note: A total number of game iterations may be calculated from the game length and video processing speed. E.g. for 10 [min] of the game and image recognition speed of 50 [FPS], we have a 10 x 60 x 50 = 3000 iterations of the game. This is a 3000 potential decision-making moments, although in many iterations there will be no decision-making at all (agents will be processing designated tasks). Important is that the number of iterations is always finite and data about the game and decision-making process should be stored in some kind of log or data storage medium for the game analysis after the match.

2.3 Definition of Agent

The universal definition of an agent, according to (Ferber, 1999), is following:

"An agent is a physical or virtual entity
(a) which is capable of acting in an environment,
(b) which can communicate directly with other agents,
(c) which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize),
(d) which possesses resources on its own,
(e) which is capable of perceiving its environment (but to a limited extent),
(f) which has only a partial representation of this environment (and perhaps none at all),
(g) which possesses skills and can offer services,
(h) which may be able to reproduce itself,
(i) whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representations and the communications it receives."

In the case of the MDM agent (agents that are using MDM algorithm for decision-making), the main deviations from this universal definition are in (e) and (f). The MDM agent possesses all the information about the environment that it needs to make a qualified decision. This is both necessary and sufficient condition that must be fulfilled in order to function properly. All the information types are defined in the design phase of the agent's development process and there should be no other information needed. Otherwise, it is a design flaw.
The definition mentioned above is very universal and very general and it is not very suitable for our purposes. We would like to present a more specific (formal) definition of the agent. We will proceed from the formalized definition of the (reactive) agent as it is presented in (Wooldridge, 2002; Genesereth & Nilsson, 1987; Kelemen, 2001; Kubik, 2004), but some significant changes have to be made in agent's formalized definition for the MDM-based agent. The reactive agent basis will be sufficient enough at this moment because we will try

to keep it as simple as possible for comprehensibility purposes. Further extension of this definition is possible to reflect social aspect of the game. Following definition is translated specifically from (Kubik, 2004), but similar versions are also present in other sources (Wooldridge, 2002; Genesereth & Nilsson, 1987; Kelemen, 2001) (for perception P and environment E):

$$percept : E \rightarrow P \tag{7}$$

"Perception of the agent has immediate effect on its actual state:"

$$change_of_state : P \times I \rightarrow I \tag{8}$$

"Every action from the set of actions A, which is agent able to perform, the agent performs on the basis of application of the perception function on the actual internal state of the set I (set of all states of the environment)*:"*

$$action : P \times I \rightarrow A \tag{9}$$

"Apart from the change of internal state have actions also impact on the environment:"

$$impact : A \times E \rightarrow E \tag{10}$$

"The agent is a 6-tuple:"

$$\{P, A, I, percept, change_of_state, action\} \tag{11}$$

"The set of goals of the agent C *is a subset of the set* I*. The goal may be characterized as a* ***maintaining goal*** *and may be theoretically running infinitely (to be in a state of readiness and continual functionality), or* ***reaching goal****, which is defined by a state which achieving is characterized as a goal accomplishment."*

This is the end of the definition taken from (Kubik, 2004).

For the purposes of the MDM implementation, the cited definition has to be modified.

The MDM-oriented agent is a 6-tuple:

$$\{C, S, P, T, V, W\} \tag{12}$$

Where:

C ... set of game configurations (defined by (6)),

S ... set of internal states of the agent (definition is following),

P ... set of perception functions (definition is following),

T ... transition relation for transition between states (definition is following),

V ... set of all possible variants of solutions (definition is following),

W ... set of weight functions (definition is following).

In the case of the MDM-based agent, the goal orientation of the agent is specified by its actual state. **State** is, in fact, an algorithm, performing certain operations in the environment, but it is more useful to use the abstraction of the term "*state*" for easier understanding. Thus, whenever the "action" or "state" of the agent is mentioned, we are referring to the same thing.

The **set of states** S contains two subsets S_S and S_T. Thus, $S = S_S \cup S_T$. The **set of stable states** S_S contains maintaining goals (continual functionality of an agent). The agent is remaining in the stable state and there are two situations which may force him to change it: (a) an exceptional situation occurred – the agent has to take actions leading to correction of the exceptional state of the environment in order to be able to be in its stable state again; (b) the game situation requires agent to change its basic behavioural pattern and to do something else (E.g. the agent in the role of an attacker was waiting for the pass and as soon as he gets the ball under its control, its is trying to score a goal). In the case (a), the agent is making a transition to the temporal state, in the case (b) to another stable state. It is important to remain as much as possible in the stable states for these are defining functional behaviour leading to the highest benefit of the team.

The **system of sets of temporal states** S_T is containing states used to handle exceptional situation which is disrupting standard behavioural pattern, defined by the stable state. There is a special type of dependency of temporal states on stable states. Formally, it may be defined this way:

For every agent, there exists the set of stable states $S_S = \{s_{S1},\ldots,s_{Sn}\}$ and the system of sets $S_T = \{S_{T1},\ldots,S_{Tm}\}$. $\forall s_{Si} \in S_S$ has assigned a subset $S_{Ti} \subseteq S_T$. For $\forall s_{Tj} \in S_{Ti}$ is (s_{Si}, s_{Tj}) an ordered tuple. $\forall S_{Ti} \in S_T : \bigcup_{i=1}^{m} S_{Ti} = S_T$ and subsets $S_{Ti} \subseteq S_T, S_{Tk} \subseteq S_T$ are pairwise disjoint. In other words, each stable state has assigned a set of temporal states (it may be even an empty set). For better understanding, Fig. 3 is representing this situation.

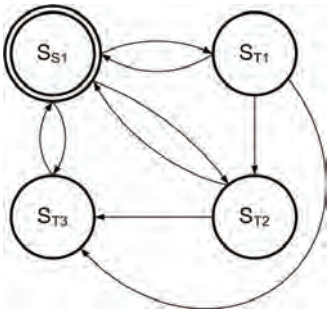


Fig. 3. Stable state has assigned a set of temporal states used to handle exceptional conditions that may occur

Another part of the agent is the **set of perception functions** P. P contains functions $p_j \in P$ representing agent's sensors. The definition from (Ferber, 1999), mentioned at the beginning of the part 2.3, is describing agent's definition of the world in this way: *"...the agent (e) is capable of perceiving its environment (but to a limited extent) and (f) has only a partial representation of this environment (and perhaps none at all)"*. There is an important difference in the MDM-based agent case. The MDM-agent has all the information needed for making the decision. Because the environment $E = norm(A)$ is from the agent's point of view all the information needed (fulfilment of this condition is a matter of proper design and testing), then following expression must be valid for the agent in order to be able to function properly:

$$\forall a_j \in A, j \in (1,...,z) \exists p_j \left| p_j(a_j) = norm(a_j) \right. \tag{13}$$

In other words, all attributes of the environment are monitored by perception functions. Following expression is valid for $p_j \in P$:

$$\forall p_j \in P : Dom(p_j) \in \langle k,l \rangle \left| -\infty < k < l < \infty \right. \tag{14}$$

$$\forall p_j \in P : \mathrm{Im}(p_j) \in \langle 0,1 \rangle \tag{15}$$

Thus, every percept a_j is normalized to interval $\langle 0;1 \rangle$.

The **transition relation** T is used to set transitions between states. At the Fig. 4 is shown the situation of the stable state with maximum number of transitions for temporal states and two transitions to other stable states from the state S_{S1}. Because the transitions between temporal states themselves are not so important (after finishing the behaviour in temporal state, the agent continues its functioning in another state), we are interested in a transition

index T_i, which is a number of possible ways beginning in given stable state S_{Si} and ending either in itself or another stable state. Transition relation must follow these restrictions: (a) There must be no cycles between temporal states, (b) the state must not have a transition directly leading to itself, (c) total maximum possible number of ways how to solve the situation (every transition is a reaction to the configuration in the given moment) is represented by *complicatedness* value, given by the formula:

$$complicatedness(s_{Si}) = (2^a - 1) + b, s_{Si} \in S_S \tag{16}$$

Where a is a number of temporal states related to the given state s_{Si} in the means of definition in the part regarding the system of sets of temporal states S_T (see above). b represents a total number of stable states which are accessible from s_{Si}. This is important because we are able to specify total computational capacity needed for considering reactions (making prediction computations during the DMP).

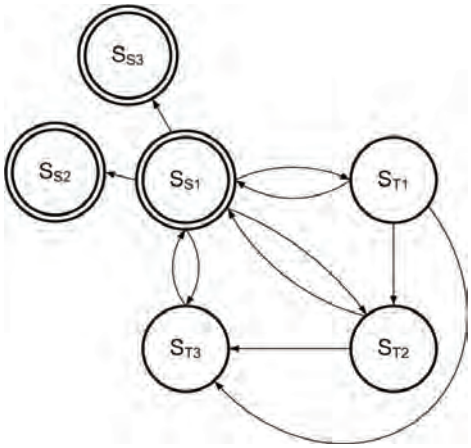


Fig. 4. A maximum number of transitions between states is finite

The three restrictions mentioned in the above paragraph specify the characteristics of the T relation: (a) T is not symmetric, (b) T is not reflexive, (c) T is transitive.

The **set of variants of possible solutions** V contains s-tuples of attributes subsistent to every transition between two states. Formally:

$$\forall s_i, s_j : s_i \xrightarrow{T} s_j \left| \exists v_r \in V : v_r = (a_1, ..., a_s), a_i \in A, i = (1, ..., s), r, s \in N \right. \tag{17}$$

Elements of the set V are used for the calculation of convenience (see below).

The **set of weight functions** W contains weight functions assigned to the each attribute of every $v_r \in V$. Formally:

$$\forall a_i \in A : \exists v_r \in V : v_r = (a_1,...,a_s) \left| \exists w_i \in W : w_i(a_i) \right. \tag{18}$$

For further explanation of weight functions, see part 3.2: "Attributes and Weight Functions".

2.4 Other Agent's Characteristics

Agent was defined in the part 2.3. In this part we would like to present some other characteristics and design conditions in addition to this definition.

For every state $s \in S$ must be true, that it has at least one transition defined:

$$\forall s_i \in S : \left| V_i \right| \geq 1 \tag{19}$$

In this case:

$$\exists s_i \in S : \left| V_i \right| = 0 \tag{20}$$

There is a danger of deadlock, because there is a state with no transition defined. Agent would not be able to do anything else.

3. Principle of the MDM, Attributes and Weight Functions

Attributes describing environment must have certain properties. Above all, they must be numerically represented. Issues regarding attributes´ design are discussed in this part. Examples of attributes and their weight functions will be provided here.

There are some limitations in the attribute design process. Following conditions must be fulfilled before the attribute may be implemented in the system: (a) attribute must be numerically represented, (b) attribute must be limited (see (1)) and (c) the agent must be able to obtain the attribute's value from the environment (see (13)). The (a) condition is the most restrictive, because many influences are difficult to be represented numerically. In human decision-making, for instance, the emotions play an important role. Although it may be possible to implement it somehow, it is obvious that there is a big problem for analysis and design and the solution will always be very questionable.

For many influences we would like to implement in the system and which have a problem with the numerical representation, there exists an emergency solution. We may use techniques used for fuzzy expert systems definition (resp. fuzzy sets definition) where we use the subjective division of linguistic variables into sets, which returns a numerical value for the variable. Still, the subjective nature of this approach brings some degree of unreliability into decision-making process because we may never be sure if the subjective

definition was made properly. However, the most of the attributes is usually of "technical" nature and numerical representation is not very problematic issue, fortunately.

3.1 Principle of Multicriterial Decision-making

The MDM is used on two levels in the robot soccer game: (a) individual agent's decisions, (b) team behaviour decisions. However, the principle is same in both situations. The following formula is used to compute the convenience value for any solution v_r (this formula is an adaptation of standard formulas used in multicriterial decision-making, presented in (Ramik, 1999; Fiala et al., 1997)):

$$convenience(v_r^T) = \sum_{i=1}^{s} w(norm(a_i^{T+x})), v_r \in V, a_i \in A, i = (1,...,s), r,s \in N \tag{21}$$

Where x is time needed to perform the action v_r and we are considering the game situation in a certain time moment T. The w represents a weight function, assigning to each attribute its importance and it is described in the next paragraph.

Before the attribute $a_i, a_i \in A$ is evaluated and we obtain its convenience value, the weight function $w_i(a_i), i \in (1,...,z)$ must be applied on it. Thus, every attribute in the environment has at least one weight function assigned. One attribute may have assigned more than one weight function depending on the number of variants it participates in. The utility of usage of the weight function is in the setting of attribute importance value for the DMP and decisions are reflecting reality more accurately. All attributes taken into account during the decision-making are supposed to have some influence on it. However, the one attribute's importance is not equal to another. Therefore, the application of the weight functions may not be omitted.

The formula (21) is the most critical for the MDM-agent because it represents the most fundamental basics of the multicriterial decision-making principle.

3.2 Attributes and Weight Functions

The form of the weight function is depending on the ideal state of environment we are trying to achieve and on the extent of tolerance for the difference between the actual and ideal state. In many cases, the ideal state is only a theoretical abstraction, but it is providing a reasonable orientation for the agent in the means of what is good and what is not good for it.

Example: The ideal would be to move to the desired position as fast as possible. Therefore, the ideal solution is consuming neither time nor energy. This is, of course, physically impossible, but the agent is trying to get as close to is as it is possible, saving as much time and energy as it can.

There is also a difference whether the agent is trying to process an action and reach the desired state of the environment or whether it is trying to remain in its state except when the exception occurs. The former case is typical for the temporal states of the agent (see the definition in the text above); the latter is typical for the stable states, where the agent is trying to perform its behaviour as long as possible. For better understanding, see Fig. 5.

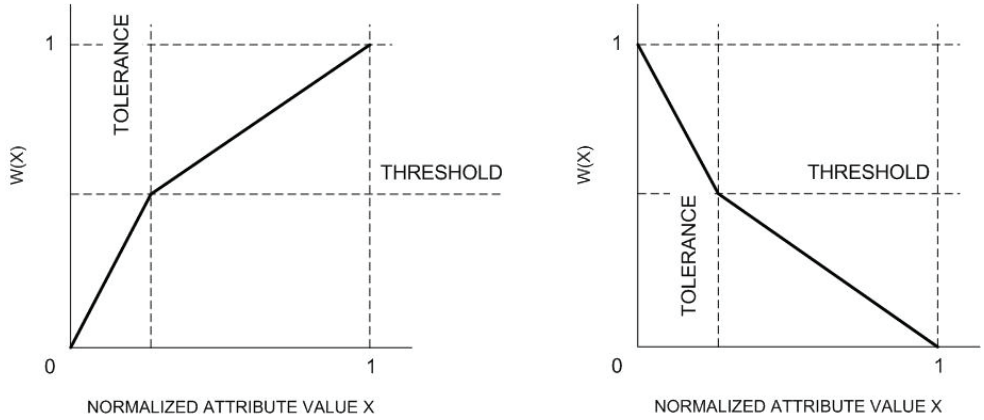


Fig. 5. Weight functions of attributes. The function on the left is motivating the agent to change its stable state to solve the exceptional situation, the function on the right is representing agent in the temporal state, trying to change the attribute's value to the tolerable level

For each type of the state, a threshold value is set. In the case of stable states, the threshold is used to ignore minor differences between the actual and ideal state. In other words, percept may be not intensive enough for the agent to be reasonable to react to it. On the other hand, there are also temporal states which are used to solve the exceptional situations. We need to know when the state of the environment is suitable enough to continue in the stable state algorithm processing. In both cases, the threshold value is assigned to the state.
In the part 9 of this text, the case study describing behaviour of the goalkeeper agent is presented. Further explanation of problems opened in this part will be provided there.

4. Environment Description and Attribute Composition and Decomposition

The proper description of the environment is essential for the function of the MDM control. The agent-environment interaction has special requirements that have to be met and such matters are discussed at the end of this part.
The composition and decomposition of attributes is important in the analysis phase of the development process. It has no influence on the decision-making itself. In fact, the decision-making should be giving the same results after application of composition/decomposition technique. The benefit is in the easier understanding of the environment for the human operators because they may use any level of the differentiation as they see fit.

4.1 Agent - Environment Interaction

The range of the agent's actuators is limited. Usually, agent is not able to change all attributes of its environment. This is generally valid for all agents in the real-world situation. E.g. the human is able to perceive much more than he is able to influence directly. The situation is similar in the robot soccer environment. The way it works is described on the Fig. 6:

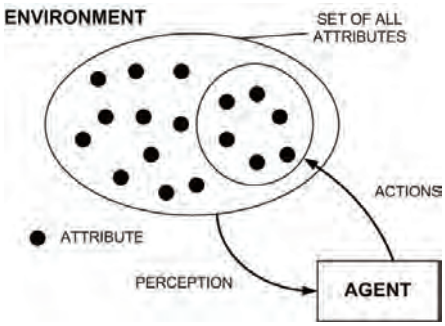


Fig. 6. Agent-environment interaction

The set of attributes (that we are able to identify and define) is often not providing description of the environment with sufficient precision. On the other hand, many minor influences are forming a significant power together, which is important for the decision-making process only as a sum of these negligible influences.

The solution to problem how to get a description of the environment that suits our purposes is represented in a technique of **composition and decomposition of attributes**. It is common that the final value of an attribute is obtained as a sum of several less important attributes. While using the MDM principle, we are able to regulate the description for our purposes while maintaining its descriptive power.

4.2 Composition and Decomposition of Attributes

Let us begin with an example which is not connected with the robot soccer problem. We believe that this example is well-suitable for the explanation purposes. Let us consider the environment of our agent described by just one attribute "FINAL PRICE" which does not provide as much descriptive potential as we need. We would like to be able to work with it, but with more precision than it is now possible. We know that the "FINAL PRICE" is composed of two constituents "COST" and "PROFIT", as you can see at the Fig. 7.

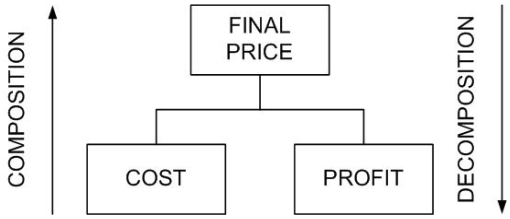


Fig. 7. The composition and decomposition of an attribute

Let us leave aside the exact participation of lower-level attributes on the upper-level attribute's value. It is simply composed of two constituents of lower-level of description. The lower level is more precise. It is same the other way. General situation is presented at the Fig. 8:

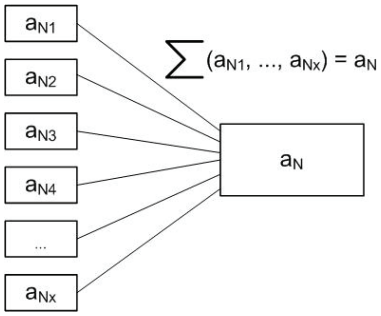


Fig. 8. An attribute a_N is composed of lower-level attributes a_{N1}, ..., a_{Nx}.

The variable $a_N \in A$ is any attribute of the environment and it is a composition of constituents $a_{Ni}, i = (1,...,x)$. In general, we may use the decomposition/composition as we see fit, but there are two restrictions:

$$w(a_N) = \sum_{i=1}^{x} w(a_{Ni}) \tag{22}$$

The weight of an attribute must remain the same as it was before the decomposition and:

$$a_N \xrightarrow{DECOMPOSE} (a_{N1},...,a_{Nx}) \xrightarrow{COMPOSE} a_N \tag{23}$$

The process of decomposition may be inverted by the process of composition without any change (in the means of empiric or descriptive value).

5. Strategic Control Module

The schematic representation of the strategy control module (and the rest of the system) is presented in this part of the chapter. It is necessary to perceive control mechanism as a part of a whole. The whole system controlling the multi-agent team must act co-ordinately.
Basically, only three modules are needed to control the robot soccer team. The information needed for the decision-making process are provided by the video recognition module, the control module is responsible for decision-making tasks and transceiver module is sending instructions to the robots on the pitch. The work of the first and third module is automated; the control module's activity is depending on the complexity of the situation on the playground. The basic scheme described above is represented on the Fig. 9.

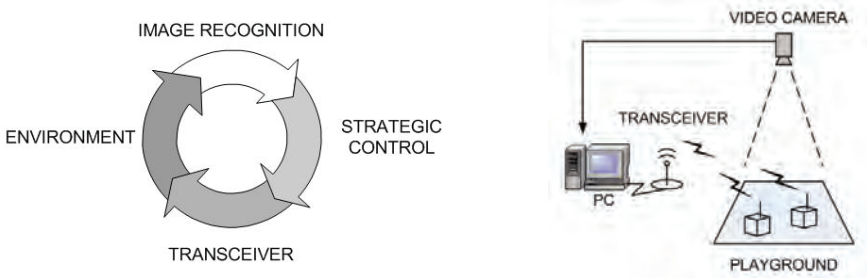


Fig. 9. An information-flow circle (on the left) and the decision-making cycle (on the right)

However, the situation is not that simple. There is a whole sequence of activities in the strategic control step, which is most important to us. More detailed description of the control process' communication is described in the part 5.1. of this chapter. Explanation of the strategic module functioning is in the part 5.2.

5.1 Communication

The system is communicating according to the following scheme (Fig. 10):

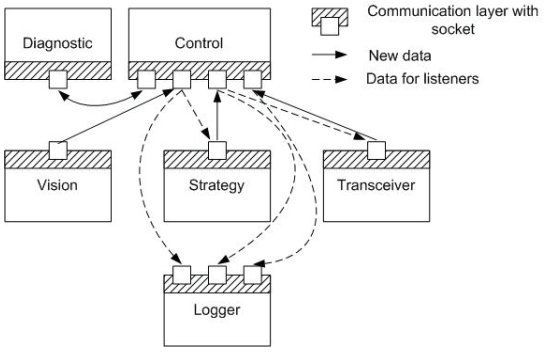


Fig. 10. Scheme of communication

The architecture client-server is used for communication. There is only one server in the system, represented by the "control" module. Clients are not communicating directly, but through the "control" module only. Clients are connected to the "control" module and set into listening mode. Information obtained by the "control" module is distributed to the respective modules. This is providing the necessary data-flow inside the system. As the system is divided into separated modules, the physical location of them is no longer of any interest to us due to the use of socket communication. When the dedicated network is used, we are not limited by the other form of data traffic and the control system is able to fully utilize the computational potential of the terminal it is running on.

Every module uses obtained data and is attaching own part of information to it. Exception is the transceiver module which is sending data on the field without any additional information.

The two parts of the system are not necessary: The diagnostic module and the **logger module**. For the latter, it was already mentioned that it is useful for us to have some kind of feedback from the system for subsequent analysis of the game. The **diagnostic module** is monitoring communication flow and in the case of any error, the human operator is informed. However, these two parts are unnecessary for the decision-making process and have only a support role.
The decision-making cycle is formed by the following sequence of activities: (1) the image from the video camera is processed and positions of all players and of the ball are recognized; (2) the strategic module is using this information to compute all the remaining attributes needed to make a decision and appropriate reaction is decided upon them; (3) the output of the strategic module – instructions for the robots – is send by a transceiver module to all players; (4) new information is being obtained from the game by the video camera. The situation is shown at the Fig. 9.

5.2 Strategic Control Module

From our point of view, the most important part of the system is the strategic control module. It was mentioned above that there is a sequence of activities present, as it is shown at the Fig. 11.

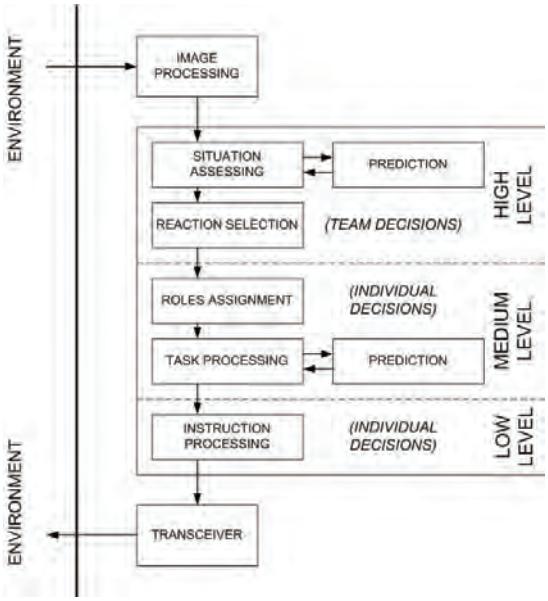


Fig. 11. Strategy control module

There are three levels of decision-making present. At the **high level**, the game configuration recognition is made in the situation assessing step. In other words, the system is trying to recognize any known pattern of behaviour and react to it. The most important attribute of the game is the possession of the ball. Whoever controls the ball controls the game. Accordingly, the set of all possible strategies is divided into three subsets: (1) offensive

strategies (our team has the ball); (2) defensive strategies (opponent has the ball); (3) conflict strategies (no one has the ball or the possession of it is controversial – both teams have players very close to it). The most appropriate reaction is selected by the MDM, while the attributes are calculated or predicted as needed in order to be able to make a qualified decision. When the reaction is selected, every agent has basic conception of its behaviour in the next step of the game. The decision-making process is continuing at the **medium level**. At this level, the individual adaptations are made to the assigned behaviour. The agent has assigned a role, which is determining its coordinating position (if it is subordinate to the other agent or not). The specification of a reaction is vague and it must always be adjusted to the actual situation. Again, as the agent is adapting the parameters of the task to its actual situation, it may predict the course of actions as needed. At the **low level**, there is in fact no decision-making at all. The action is divided into atomic operations and these are just adjusted to the physical limitation of the robot's kinetic apparatus (we need to avoid slipping, falling, going into a skid, etc.). These atomic operations adjusted to physical parameters of the environment are called **instructions**. Instructions are, in fact, nothing more than commands used to setting the wheels velocities of the agent. This is the output of the strategic module and the instructions are send by a transceiver to all respective agents on the pitch.

5.3 Prediction

As it was mentioned before, the agents are using a prediction to approximate the future development of the game. This is not unusual; the systems dealing with the robot soccer problem are often using some kind of simulation mechanism for possible solution predictions, computations or verifications. The most important is prediction of the movement of the opponent's agents (We do not need to predict movement of our agents; we have this information stored in the memory. Therefore, for our agents, the prediction is used only for evaluation of possible situations or computing positions in coming iterations.). The situation is shown at the Fig. 12.

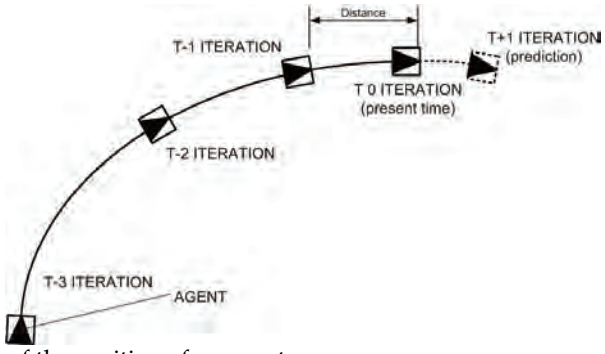


Fig. 12. Prediction of the position of an agent

We are predicting future position from the previous iterations of the game (three or four previous iterations are providing enough information to predict position in the next iteration of the game). For the illustration purposes, we present general for of formulas primarily used for the quadratic extrapolation:

$$x(t) = x_0.l_0(t) + x_1.l_1(t) + x_2.l_2(t), \tag{24}$$

$$y(t) = y_0.l_0(t) + y_1.l_1(t) + y_2.l_2(t), \tag{25}$$

For calculation of $l_0(t)$, $l_1(t)$ and $l_2(t)$ are used following formulas:

$$l_0(t) = \frac{(t-t_1)(t-t_2)}{(t_0-t_1)(t_0-t_2)} \tag{26}$$

$$l_1(t) = \frac{(t-t_0)(t-t_2)}{(t_1-t_0)(t_1-t_2)} \tag{27}$$

$$l_2(t) = \frac{(t-t_0)(t-t_1)}{(t_2-t_0)(t_2-t_1)} \tag{28}$$

t_0 is the oldest iteration.

6. Team Strategy and Individual Behaviour

There are two points of view on the control responsibility – the team strategy decisions and individual actions. Excellent team must find the equilibrium between both approaches. The database of possible strategic formations and movements is used for coordination.

It is a common solution, that the team is maintaining a formation. Each member of the team has assigned a position on the pitch and is responsible for certain area. The goalkeeper's purpose is to guard the goal and passing the ball forward if it is possible. The defender is covering the goal from the angles where it cannot be done by the goalkeeper and is interfering with the opponent's players. The attacker is trying to get the ball under control, avoid the opponent's players and score the goal. Although all agents act individually, the basic pattern is defined centrally.

For the MDM principle, there is no other way than use the combined centralized-autonomous approach where the centralized decision sets the common course of action and every agent act individually, but in correspondence to other agents'. To use some sets of rules to maintain formations (or similar solutions) would make the whole decision-making extremely difficult, complicated and human-control-proofed. Moreover, the analysis and design would be immensely difficult.

Therefore, the principle of coordination is based on the progressive change of behaviour from centralized control to the individual decisions. As the decision-making process is continuing and reasoning is proceeding to lower levels of the strategy module (see Fig. 11),

the agent has gradually greater influence on the form of final realization. Besides, the agent has to adapt universal form of algorithm to actual situation every time.

7. Modular Strategic Control

The actions of the robot players may be decomposed to atomic tasks. The strategic scenarios may also be decomposed to actions is a similar way. In this part, the possibility of change of the set of actions taken into account while making decision is discussed.
The strategic scenario is a coordinated reaction of (one or more) agents to the situation on the pitch. It is useful to have pre-prepared scenarios to most common situations. If the situation on the playground cannot be recognized or if there is a large probability of incorrect reaction, some kind of default behaviour should be used.
There is an advantage hidden in the state-organized structure of behaviour. Once the algorithms are prepared (for either stable or temporal state) the expansion or constriction of the set of states is easily done. When we have the behavioural structure for all states, it may be similar to the one presented at the Fig. 13.

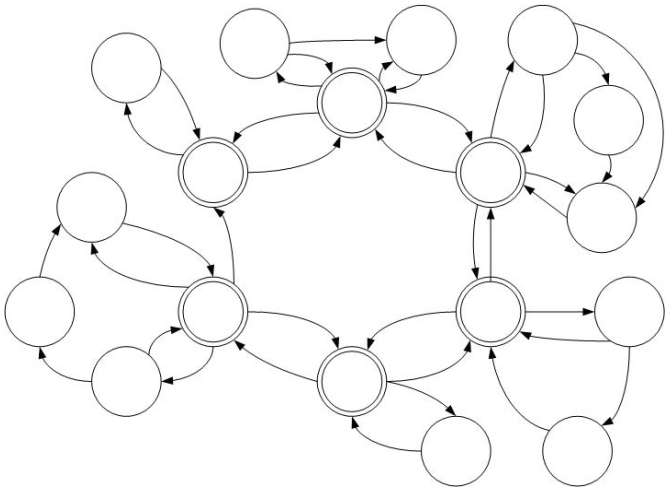


Fig. 13. Example of the behavioural pattern of an agent

To change the behaviour of an agent, some transitions may be suppressed. However, conditions (19), (20) must be fulfilled. The structure presented at the Fig. 13. is called the **behavioural pattern** of an agent and it is defined individually for every type of the agent on the pitch (attacker, goalkeeper, defender). E.g. there would be another behavioural pattern defined for the aggressive strategy and for defensive strategy.
The great advantage is the modularity of the behavioural pattern. New states (algorithms) may be easily incorporated into existing structure and there is no need for changes in the basic structure. Also, this feature is invaluable for the testing and weight function tuning purposes.

7.1 Machine Learning

The most appropriate learning method for the MDM principle is the reinforced learning, according to the MDM principle requirements. Explanation of this method can be found in (Ferber, 1999) and (Kubik, 2004). However, it is important, that it is only a high level of strategic module that is capable of learning. On the medium and low level, the actions must be already debugged and performed precisely; the high level is responsible for the situation recognition and this is a process where the learning may be proven useful.

8. Synchronization of Team Activities

During the robot soccer game, a group of mobile agents must often coordinate their actions to follow the team strategy. Therefore, there is a need to adapt the plan of each participating agent in order to act co-ordinately. Although the basic set of actions, which would be taken, is known from the high level of the strategy module, the final way of realization is depending on the situation on the playground.

As the most appropriate solution was selected the application of hierarchy for the coordination purposes. Generally, it is possible to decide which player has the best position on the pitch. Therefore, such an agent is taking the lead for the execution of co-ordinated action. We have only two types of roles for this purpose. The role of **leader** is assigned to the agent in the best or most important position. The **support** role is used for other agents.

Because we need all team members (except the goalkeeper) to be able to participate in the game properly, the designation of the agent into the position (attacker, defender) is not constant. It may be changed during the game. However, the distribution of power on the pitch must remain the same. Therefore, another agent (usually with the worst position for the actual action) takes its place instead.

It is the leader who chooses the most appropriate solution for his situation. Other players are acting accordingly. E.g. when our team is attacking, it is usually a good idea to have a player near to the opponent's goal. When the ball is near, only a swift move will score a goal. Other actions may be planned in a similar way.

9. Case Study – Goalkeeper's Behaviour Description

For explanation purposes and better understanding, a description of the goalkeeper's behaviour will be provided. The aspects of MDM described in the previous parts of this chapter will be shown on this example.

The goalkeeper's behaviour is not directly dependent on other team member's behaviour. Therefore, the case study is not unnecessarily complicated by the social aspect of the game. Discussion about these features would take a lot of time and for basic understanding it is not necessary to make things difficult.

The goalkeeper's main task it to protect the goal from the opponent's players. To do this, it is important to monitor the position of the ball. Because the ball may not move unpredictably (except the case of collision), but only in the direction of movement of the robot which has it under control, we use prediction described by formulas (24), (25), (26), (27), (28). We have to react to the next position of the ball, not to the actual one. The problem is schematically described on the Fig. 14.

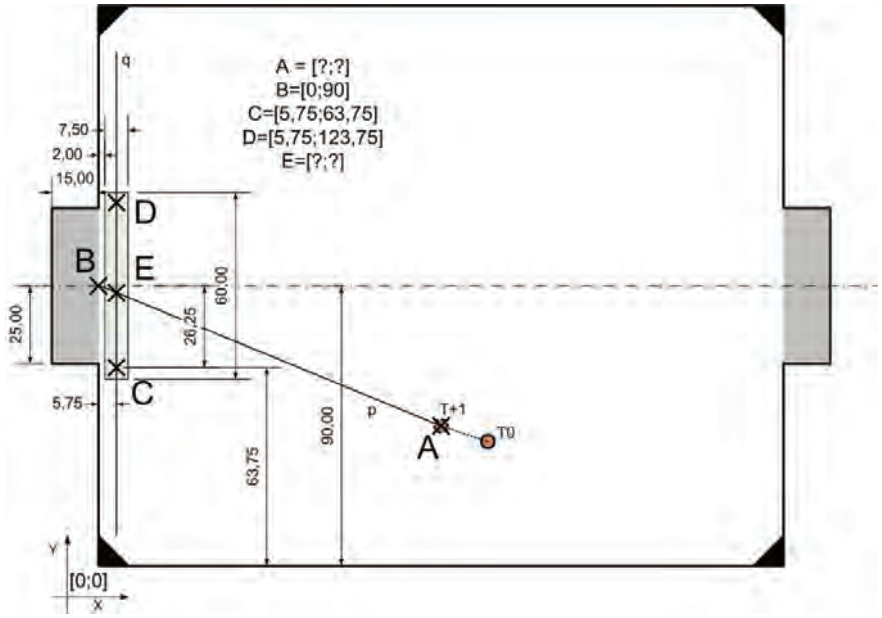


Fig. 14. Protecting the goal

The activity of the goalkeeper is following: (1) remain on the abscissa CD; (2) set the coordinates (point E) to intersection between point B and the predicted position of the ball (point A), (3) the front must remain turned concurrently with the line q (90° or 270°). By following this simple set of rules, the goalkeeper is able to defend the goal. The time difference between the moment T0 and the point A may be adapted as needed (according to the speed of game and capabilities of the opponent's players). However, this simple behaviour is furthermore complicated by handling exceptional situations which may never be entirely avoided in the real world. The most common problem is moving away from the line of movement.

Using the terminology introduced in the beginning of this chapter, the stable state of the goalkeeper agent is defending the goal. Two deviations from the ideal state may occur: the agent may be away from the designated line (CD) or he may be turned in an incorrect angle. If both exceptions occur, it is obvious in what order should be exceptions handled. First will be positioning, second will be turning in the proper angle. Therefore, there are monitored two corresponding attributes – the position (it is compared with the optimal state (see Fig. 14)) and the rotation. The positioning and turning are two temporary states; each of them is a simple algorithm solving the part of the problem. When the exceptional situation occurs, the decision-making procedure is performed and appropriate action selected.

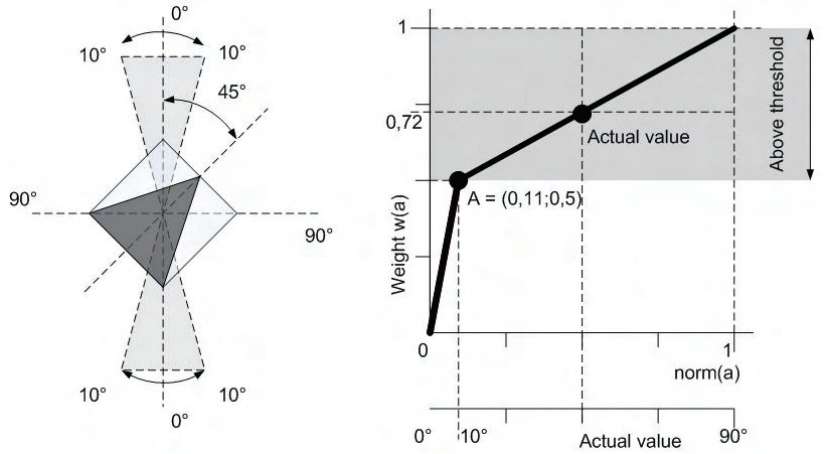


Fig. 15. Temporary state – turning. There is an exemplary "actual value" present

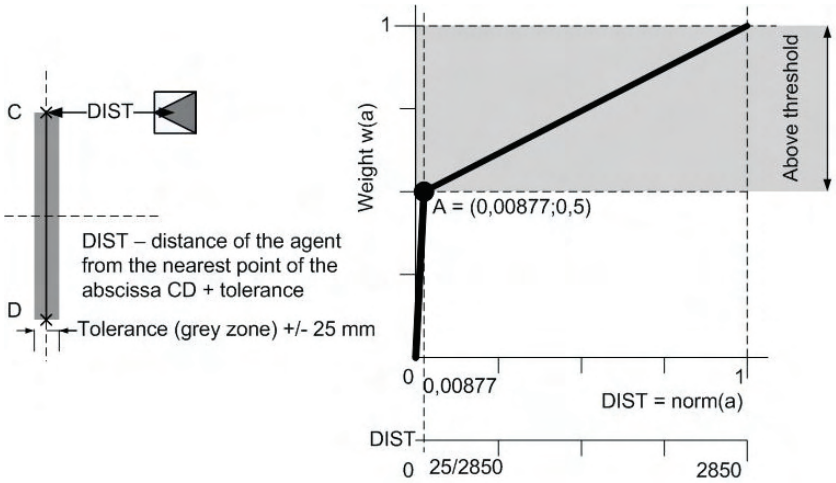


Fig. 16. Temporary state – positioning. The maximum value of the attribute DIST is given by dimensions of the pitch (its diagonal is 2842,53 mm)

It is important that there is a threshold value assigned to the each state. The purpose of the stable state is to prevent treating of every negligible difference from the ideal state. Certain level of imprecision has to be tolerated always. For the stable state, the threshold value is used to recognize when the algorithm should stop. See the Fig. 15 and 16 for details.
The behaviour of the goalkeeper agent is therefore defined by the behavioural pattern shown at the Fig. 17.

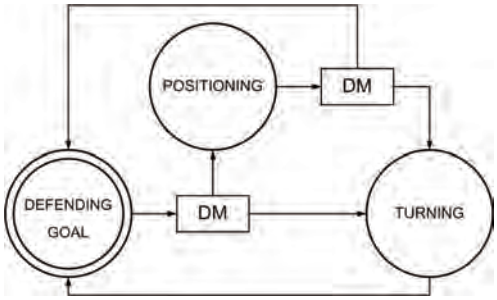


Fig. 17. Behavioural pattern of goalkeeper agent

The "DM" symbols at the Fig. 17 are representing decision-making moments. Until the threshold value is exceeded, the agent remains in the stable state "Defending goal". If the agent is turned in a wrong angle or has a wrong position, the stimulus' power will overcome the threshold and action with highest convenience value is selected and done.
Generally, there are two ways how to continue. The first way requires the agent to make prediction of development more than one step ahead. In this case, when deciding what to do next, the whole sequence of activities is considered (i.e. sequence of actions POSITIONING and TURNING, in our case). In this case, a maximum number of possible solutions will be matching the formula (16). This approach is making the planning and co-ordination more effective.
The other way how to proceed is to select only one next action at the time. This gives the agent an opportunity to quickly react to problems. However, the maximum effectiveness of the agent is obtained when it is staying in the stable state.
The goalkeeper's behaviour may be further complicated by the incorporation of the defender agent into our case. To be effective, the agents need to adapt their behaviour to each other. However, the leading agent is goalkeeper. He continues to perform his task as it was described above, the defender is trying to cover goal where it is impossible for the goalkeeper.

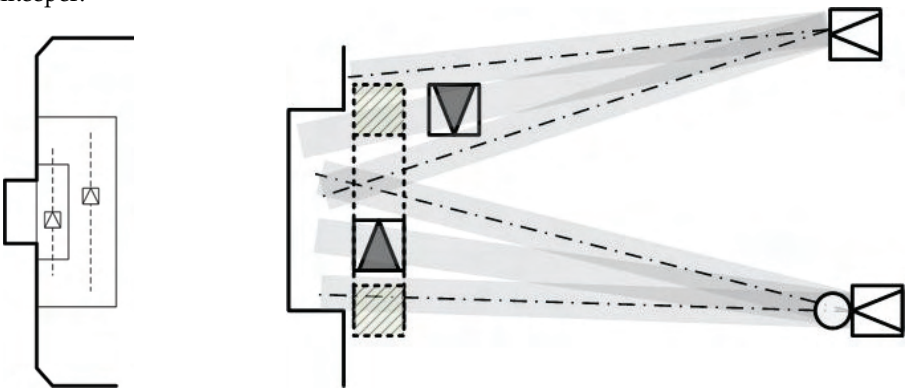


Fig. 18. The co-ordinated behaviour of the goalkeeper and defender against opponent's attack (white agents on the right side)

This description of the goalkeeper's behaviour is not meant to represent a complete behaviour of an agent. Its purpose is to illustrate the application of the MDM principle on the exemplary situation.

10. Conclusion and Future Work

The MDM method shows a great potential in the means of adaptability and speed. From this point of view it is suitable for the purposes of the robot soccer game. It is able to react quickly enough in the strict time-frame limitation of the image processing speed. Its modular framework (implementation of the actions is made separately) allows an easy modification of behavioural control. The behaviour of the individual agent or the team as a whole may be easily changed by the human operator or some form of heuristic control. This allows the team to easily change the tactics and strategy and surprise the opponent.
However, there are some serious setbacks present. The control mechanism development is difficult mainly in the analysis and testing phases. Implementation of isolated tasks is not a problem; this work has to be done no matter what control method is used. What is difficult is the initial design of weight functions and transition-between-states debugging. Both issues require extensive testing and optimization, which takes a lot of time and effort. Our future work will be focused on these problems.

11. Acknowledgements

The work and the contribution were supported by the project from Grant Agency of Czech Academy of Science – Strategic control of the systems with multiagents, No. 1ET101940418 (2004-2008) and by the project IPSV3/07 of University of Hradec Kralove.

12. References

Aguilera, M. K.; Chen, W. ; Toueg S. (1997). Heartbeat : A Timeout-free Failure Detector for Quiescent Reliable Communication. *Proceedings of 11th International Workshop on Distributed Algorithms*, pp. 126-140, ISBN 3-540-63575, Saarbrücken, Germany, September, 1997, Springer-Verlag, Berlin

Byrne, C.; Edwards, P. (1996) Refinement in Agent Groups. In: *Adaptation and Learning in Multi-agent Systems*, *Lecture Notes in Artificial Intelligence 1042* (Ed. Weiss, G.; Sen, S.), pp. 22-39, ISSN 0302-9743 (Print) 1611-3349 (Online)

Decker, K. S. (1995). Environment Centered Analysis and Design of Coordination Mechanisms, Ph.D. thesis, University of Massachusetts

Fiala, P.; Jablonsky, J. & Manas, M. (1997). *Vicekriterialni rozhodovani,* The University of Economics, Prague, ISBN 8070797487, Prague

Ferber, J. (1999). *Multi-agent Systems: An Introduction to Distributed Artificial Intelligence,* Addison-Wesley, ISBN 0-201-36048-9, New York

Genesereth, M. R., Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publ., ISBN 0-934613-31-1, Los Altos, Cal.

Kelemen, J. (2001). Reaktivni agenti, In: *Umela inteligence 3* (Ed. Marik, V., Stepankova, O., Lazansky, J. et al.), pp. 161-188, Academia, ISBN 80-200-0472-6, Prague

Kim, J.; Kim, D.; Kim, Y. & Seow, K. (2004). Soccer Robotics. *Springer Tracts in Advanced Robotics,* Vol. 11/2004, pp. 205-256, ISSN 1610-7438 (Print) 1610-742X (Online)

Kit, E. (1995). *Software Testing in Real World: Improving the Process*. Addison-Wesley, ISBN 0-201-87756-2

Kubik, A. (2004). *Inteligentni agenty – tvorba aplikacniho software na bazi mutiagentovych systemu,* Computer Press, ISBN 80-251-0323-4, Brno

Menzies, T. (1999). Knowledge Maintenance: The State of the Art. *The Knowledge Engineering Review,* Vol. 14, No. 1, (1999) 1-46, ISSN 0269-8889

Ramik, J. (1999). *Vicekriterialni rozhodovani – Analyticky hierarchicky proces (AHP),* Silesian University, ISBN 80-7248-047-2, Karvina

Tichy, P. (2003). *Social Knowledge in Multi-agent Systems,* Ph.D. thesis, CVUT, Prague

Tucnik, P.; Kozany, J.; Srovnal, V. (2006a). Multicriterial Decision-making in Multi-agent systems, In: *Lecture notes in Computer Science* (Ed. Alexandrov, V. N., Dick van Albada, G., Sloot, P. M. A., Dongarra, J.) pp. 711-718, ISBN 3-540-34383-0, ISSN 0302-9743, Springer, Berlin

Tucnik, P.; Kozany, J.; Srovnal, V., Pokorny, J., Lukas, D. (2006b). Software Structure of Control System for MiroSot Soccer Game, In: *Proceedings of FIRA Roboworld Congress 2006,* (Ed. Weiss, N., Jesse, N., Reusch, B.), pp. 183-188, University of Dortmund, ISBN 3-00-019061-9, Dortmund

Wooldridge, M. (2002). *An Introduction to Multi-agent Systems*, J. Wiley & Sons, ISBN 047149691X, London

23

Robust and Efficient Robot Vision Through Sampling

Alex North and William Uther
The University of New South Wales and National ICT Australia
Australia

1. Introduction

Vision is an extremely important sense for both humans and robots, providing detailed information about the environment. A robust vision system should be able to detect objects reliably and present an accurate representation of the world to higher-level processes, not only under ideal conditions but also under changing lighting intensity and colour balance; when fully or partially shadowed; with specular and other reflections; with uniform or non-uniform backgrounds of varying colour; when blurred or distorted by the object's or agent's motion; in spite of chromatic and geometric camera distortions; when partially occluded; and under many other uncommon and unpredictable conditions. Visual processing must also be extremely efficient, allowing a resource-limited agent to respond quickly to a changing environment. Each camera frame must be processed in a small, usually fixed, amount of time. Algorithmic complexity is therefore constrained, introducing a trade-off between processing time and the quality of information gained.

Within the domain of the RoboCup four-legged league, previous vision systems have relied heavily on the colour of objects since the ideal colour of most important objects is specified in the league rules: a green field with white lines, an orange ball, yellow and blue goals, and pink, blue and yellow navigational beacons. However, there is considerable scope for interpretation and variation allowed by the environmental specification, particularly with regard to lighting intensity and uniformity. Agents must be capable of performing under varying conditions, albeit with time allowed for detailed calibration procedures.

Typical systems group the continuous space of colours returned by the camera into a small set of discrete, symbolic, colours. They then attempt to form objects by grouping neighbouring similarly-classified pixels (Bruce et. al., 2000; TecRams, 2004). In implementation this usually results in a look-up table or decision tree that quickly maps the detected pixel value to a symbolic colour. Typical approaches to the generation of this table involve a supervised machine learning algorithm, where a human expert provides classification examples to a computer program, which generalises these to form the complete segmentation (Pham, 2004; Röfer et. al., 2004; Veloso et. al., 2004; Brusey & Padgham, 1999; Xu, 2004; Chen et. al., 2003).

Unfortunately, as lighting conditions change, the colours of real-world objects change and methods relying solely on colour become brittle and unreliable. In addition, these "blob of colour" based methods must process each image frame in its entirety, where there is frequently a large amount of redundant visual information. In contrast, this chapter presents a system that shifts the focus to recognising sparse visual features based on the relationships between neighbouring pixels, and detecting objects from a minimal number of such features.

Fig. 1. A Sony AIBO ERS-7 wearing the red team uniform and the orange ball on the RoboCup four-legged league field. Note the coloured goals and localisation beacons. The outer wall is optional

There have been a number of modifications attempted to address some of these shortcomings with "blob of colour" based vision systems in the RoboCup legged league. A dynamic classification approach, where the classification of a pixel value may change over time, is one such modification. Approaches based on multiple colour segmentations or relative classification have had some success, but such methods rely on multiple fine calibrations or overly simplified representations (Quinlan et. al., 2004; Sridharan & Stone, 2005; Jüngel et. al., 2003; Wasik & Saffiotti, 2002). The conclusion to be drawn from past attempts is that colour segmentation is a difficult problem. Local variations, temporal variation, complexity of segments and overlapping classifications all thwart creation of a perfect static classification, and dynamic classifications have so far been simplistic or drawn unacceptable side effects. The solution lies not in further improving colour classification methods, but in moving away from symbolic colour segmentation towards illumination-invariant vision.

Sub-sampled approaches are a recent development in RoboCup vision. Scan lines over the colour-segmented image are used in (Stone et. al., 2004) for detecting field lines and (Röfer et. al., 2004) and (Veloso et. al., 2004) for object detection. Boundaries are detected between regions of segmented colour lying on the scan lines and classified according to the adjacent symbolic colours. While still dependent on static colour segmentation this method is highly efficient. Along with dynamic colour segmentation, (Jüngel et. al., 2003) recognises that colour relationships are invariant under linear shifts in lighting, and detects edges as *contrast patterns* in the three image channels. Both of these approaches provided inspiration for the procedures presented in this chapter.

This chapter presents an image processing system for the four-legged league based on minimal sampling of the image frame. Rather than process each image in its entirety this approach makes an intelligent estimate of the information content of regions in the image and samples those areas likely to be of importance. Features are detected in the sampled areas of the image and these are combined to form objects at a higher level of processing. Instead of relying on brittle colour segmentations, this approach focuses on the relationships between neighbouring pixels, which remain more constant under variations in lighting.

This approach also presents solutions to, or implicitly avoids, some of the problems identified with purely colour-based approaches: colour segmentation need not be so tightly defined and calibration time is reduced; the calibrated colour relationship tests are environment independent; complex series of manually coded object validity tests are minimised; redundant information is avoided as only information-rich areas are sampled densely; and object recognition is robust to falsely detected features and unexpected background information. This system was successfully utilised by the UNSW/NICTA RoboCup four-legged league team, *rUNSWift*, at RoboCup 2005 and 2006.

2. Theory: Sub-sampled Object Recognition

Our approach is based on feature detection through the use of colour-gradient information gathered with a minimal sampling of each image, and object recognition from these relatively sparse features. It is based on two major theoretical differences from prior systems. Firstly, this approach moves away from absolute colour classification techniques and focuses instead on the relationships between neighbouring pixels. This reduces the dependency of the system on static colour classifications and precisely controlled lighting conditions. Secondly, this approach ceases to process the entirety of each image and instead samples only areas of the image likely to be of high informational value. This aids efficiency and reduces false positive errors caused by unexpected regions of colour.

2.1 Colour Relationships

Static colour segmentations are brittle and depend highly upon the exact lighting conditions under which the segmentation is made. As lighting conditions change the absolute colour values of pixels projected from a certain object change considerably. However, the colour-space differences between pixels representing distinct real-world colours change far less. While pixel values change under varying light, the values for all pixels will change in approximately the same way. Thus, while each colour may no longer be recognisable under a static classification, the difference between two colours remains discernible. Note that the

change in relative values is not exactly linear, depending on the particular characteristics of the light source, so while relative colour is more stable than absolute colour it is not so stable that it remains constant under excessively varying lighting.

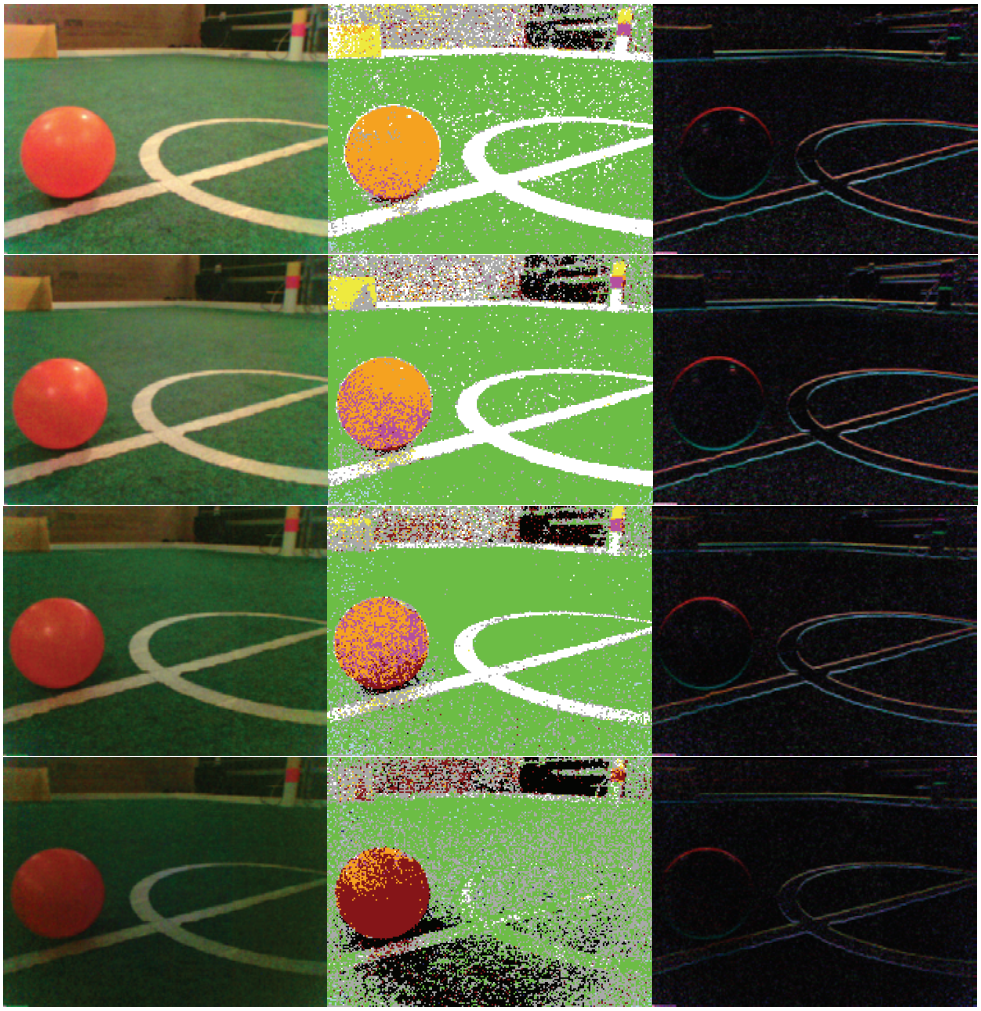
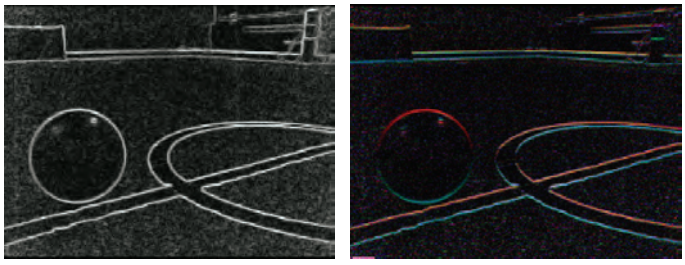


Fig. 2. A scene under progressively darker lighting conditions. The left-hand column shows images from the ERS-7 camera (after correction for chromatic ring distortion). The centre column shows the result of colour segmentation with a colour table optimised for bright lighting. The right-hand column shows the result of applying equation (1) to the images at the left, subtracting from each pixel the value of the pixel immediately above it. Note how this remains stable as the lighting intensity falls, while the colour segmented images deteriorate. The implementation described below successfully detected the ball in all images

$$\begin{aligned} Y_{x,y} &= 2\,|\,Y_{x,y} - Y_{x,y-1}\,| \\ Cb_{x,y} &= 2(Cb_{x,y} - Cb_{x,y-1}) - 128 \\ Cr_{x,y} &= 2(Cr_{x,y} - Cr_{x,y-1}) - 128 \end{aligned} \qquad (1)$$

Consider Figure 2; The left-hand column comprises images as detected by the ERS-7 camera, with ambient light intensity decreasing down the page. The centre column displays a static colour classification of the images in the left-hand column under a calibration tuned for bright lighting. Note that as the ambient light intensity is reduced the images become successively darker and the colours recognised by a static classification become increasingly less accurate. The images in the right-hand column show a graphic representation of the colour-space difference between each pixel and its vertically adjacent neighbour. Pixel values are calculated as given in equation (1). This difference precisely picks out the boundaries between the orange ball, white lines and green background. The white wall around the edge of the field is also apparent. The upper and lower boundaries of each object appear as different colours; the lower boundaries represent transitions away from green towards orange or white, and the upper boundaries represent corresponding transitions back into green. Searching for particular boundaries in the original images reduces to searching for these particular "colours" over the difference images. Note that these images remain relatively stable as the lighting intensity falls, in contrast to the rapid deterioration of the colour-segmented images in the centre column.

Detecting differences in neighbouring pixels is very similar to edge detection. General edge detection methods such as Roberts' and Sobel operators (Roberts, 1965; Sobel, 1970) (see Figure 3) are computationally too expensive to execute on every frame from the ERS-7. Instead, we may calculate a simple one-dimensional gradient. A crucial difference between this method and many standard edge detection algorithms is that the direction of the change in colour-space is explicitly captured, and "edges" can be easily classified by the colour difference they represent. Each difference-value depends on only two pixels, in this case a base pixel and the one immediately above it. Thus, to determine the colour-space transition direction at any particular point requires access to only two pixels. Determining the transition directions over a line of n pixels in any orientation requires access to only n+1 pixels, an important result for the sub-sampling approach discussed below.



(a) (b)

Fig. 3. (a) Roberts' operator applied to one of the images in Figure 2. The aggregate of the operator result over the three image planes is shown as brightness, normalised to a maximum value of 255. (b) A simple single-pixel difference as in (1) applied to the same image. Both would normally have a threshold applied before use to reduce noise

Since the direction of the vector does not depend on the actual pixel values detected these vectors will be independent of any linear shift of the colour-space. Although no real-world colour-space shift will be perfectly linear, many approximate linearity, being a compression or expansion of the colour-space along one dimension. For example, shadowing or reducing the intensity of ambient light intuitively compresses the set of observed pixel values towards black (i.e., makes them darker, but bright pixels experience greater change than dark ones). This reduction in intensity results in a much smaller change (depending on their magnitude) to the vectors representing object boundaries in the image. Thus, detecting these vectors, rather than specific colour values, is more tolerant of changes in lighting conditions.

2.2 Sub-sampling Images

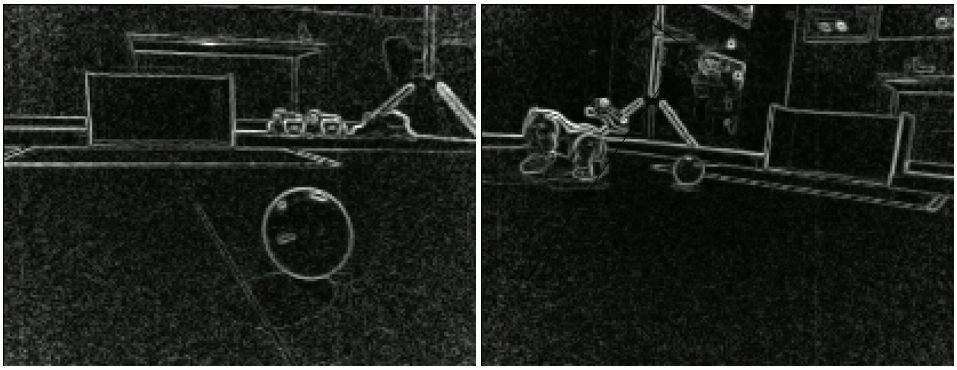
The information contained in any given image is distributed over its area. This distribution is non-uniform because neighbouring pixels are highly correlated, so the image carries redundant information (Burt & Adelson, 1983). Hence a vision system that processes every pixel in each image necessarily processes redundant information. But which pixels should we process?

In the four-legged league domain, important objects and landmarks are regions of uniform colour. Pixels in the centre of these regions of colour carry little information; their presence can be inferred from the surrounding similarly-coloured pixels. It is the edges of these objects that carry important information about position, size and orientation. Ideally, only pixels near information-bearing edges should be sampled frequently, while pixels in regions of relatively uniform colour should be sampled less often. In other domains the texture of objects may also provide useful information and samples in highly textured regions may provide additional information.

Knowledge of the environment and geometry of the robot allows first estimate for sampling the image. Figure 4 shows the results of applying Roberts' operator to two typical images. Regions of high information are displayed as edges. In general the lower part of an image carries less information (edges) than areas higher up, although the very top part of each frame usually captures background information from outside the playing field.

Objects close to the robot appear lower in the image and larger than objects far from the robot. An object such as the ball carries the same amount of information in either case, but when closer to the robot this information occupies a larger area in the image. Similarly, field lines close to the robot are sparse in the lower areas of the image. Lines further from the robot appear higher up and more closely spaced in the image.

The expected information density reflects this. Figure 5 shows a normalised sum of the results of Roberts' operator over a series of four hundred images captured from an ERS-7, holding its head level while turning its body on the spot at approximately one revolution each 4.5 seconds. The robot is positioned one quarter of the field length from one goal-line and laterally in line with the nearest goal box corner. The result of Roberts' operator on each individual image is thresholded at 40 (of a maximum 255) before being added to the aggregate in order to the reduce noise apparent in Figure 4.



(a) (b)

Fig. 4. The results of executing Roberts' operator over a number of typical in-game images. Note that the bottom part of each image, corresponding to objects close to the robot, typically contains sparse information. Objects farther from the robot appear smaller and higher up the image, leading to greater information density near the horizon

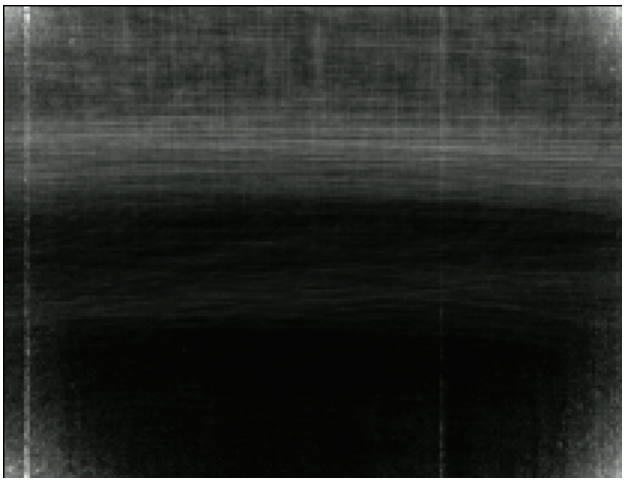


Fig. 5. The aggregate result of Roberts' operator over four hundred images. Note the dark region near the bottom of the image, an area of low average information density. Near the top of the image the brighter region represents an area of higher average information density. Noise due to chromatic distortion correction is visible in the image corners

From Figure 5 it can be seen that the average expected information density is low near the bottom of each image and increases to a peak about one third of the distance from the top of the image. The information density is approximately constant along horizontal lines.

This generalisation, that higher parts of an image are likely to carry more information, is only valid when the robot's head is held upright. If the position of the head changes then the

expected information density within the image changes too. This suggests that regions of the image close to the robot's visual horizon should be sampled with higher frequency than areas far from the horizon. We can calculate an artificial horizon from the geometry of the robot's limbs to provide a reference point invariant with its stance.

3. Implementation

We now describe in detail our sub-sampling robot vision system for the RoboCup four-legged league. This implementation is strongly biased towards working accurately, robustly and consistently in the RoboCup competition rather than towards any notion of elegance or mathematical correctness. Despite goals of domain independence implied above, advantage is taken of any reasonable domain-specific assumptions that may be made. This implementation was used with considerable success in the RoboCup 2005 and 2006 competitions.

3.1 Scan Lines

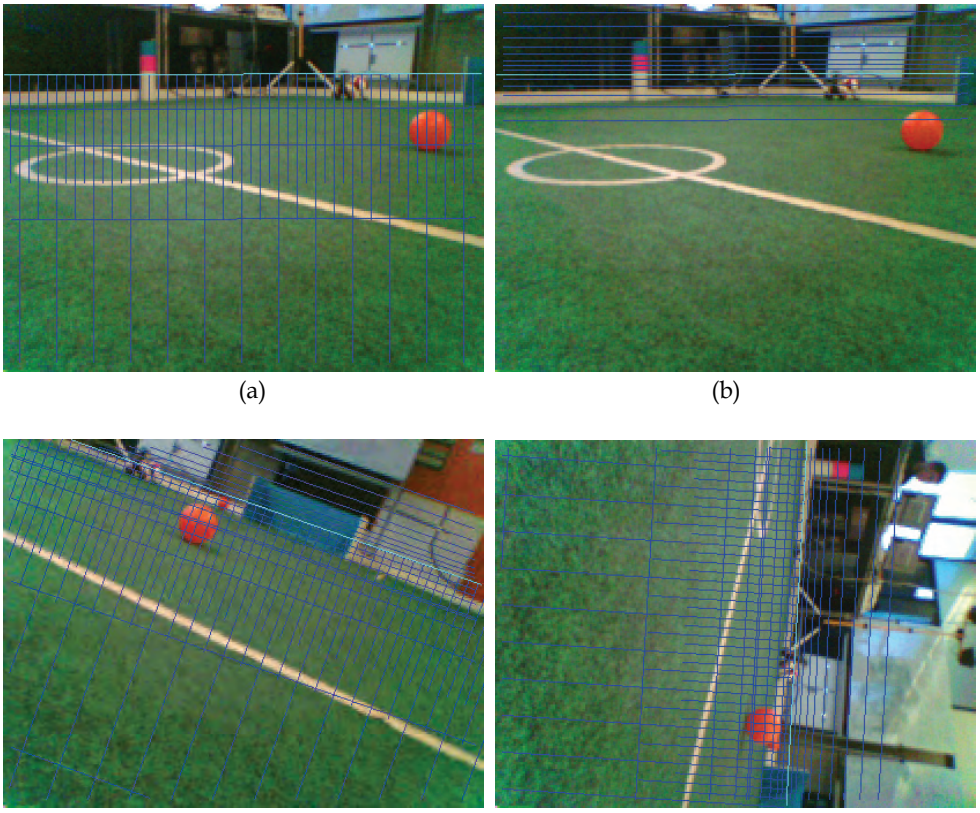
Selection of which pixels to process is by means of a horizon-aligned, variable-resolution grid placed over each image, similar to that proposed in (Röfer et. al., 2004). An artificial horizon is calculated from knowledge of the geometry of the robot's stance and camera. The horizon represents a line through the image with constant elevation equal to that of the camera. This horizon provides a reference point that is invariant with the stance of the robot and aligns the grid with the high-information areas of the image as described in section 2.2.

A scan-line is constructed perpendicular to the centre of the horizon line, continuing down to the lower edge of the image. Scan lines are then constructed parallel to this first line on either side of it at a fixed spacing of sixteen pixels; scan lines continue to be constructed at this fixed spacing until such lines lie entirely outside the image frame. Between each pair of these "full" scan lines a parallel line segment is constructed from the horizon line with a fixed length of 64 pixels. These "half" scan lines are thus eight pixels from each of their neighbouring full scan lines. Between each of these half scan lines and their adjacent full scan lines a "quarter" scan line is constructed with a fixed length of 48 pixels, spaced four pixels from the scan lines on either side. Finally, beginning sixteen pixels below the horizon and doubling this gap for each line, scan lines are constructed parallel to the horizon line until such lines lie entirely below the image frame.

This constructs a grid as shown in Figure 6 (a), with a greater density of scan lines closer to the horizon and more sparsely spaced lines further from it. This grid is used for detection of features related to the ball, field, lines and obstacles. The majority of the scan lines are perpendicular to the horizon, running from pixels that project close to the robot to pixels that project further away. The few scan lines parallel to the horizon are placed to capture features on field lines aligned with the camera axis that would otherwise lie entirely between scan lines. Since no features are expected to appear above the horizon, no scan lines are created above it.

A separate set of scan lines is constructed for detection of landmarks. Beginning just below the horizon line and continuing with an exponentially increasing gap above it, scan lines are constructed parallel to the horizon line as shown in Figure 6 (b). These lines are used for detection of the landmarks and goals around the field.

This same pattern of scan lines may be constructed regardless of the camera orientation, as in Figure 6 (c). In cases where the camera is rotated further than one quarter-turn about its axis the lower edge of the images becomes the upper edge, as seen in Figure 6 (d); in such cases scan lines continue to be processed in the same order and direction. The horizon may not appear within the image in situations where the camera is tilted up or down in the extreme. In these cases an approximation is made, drawing the horizon along the top or bottom edge of the image.



(a) (b)

(c) (d)

Figure 6. (a) The pattern of scan lines (shown in blue) used to find features on the ball, field, lines and obstacles. The horizon is shown as a pale blue line. (b) The pattern of scan lines used to search for landmarks. (c) The scan lines are constructed relative to the horizon. (d) When the camera rotates more than 90° about its axis the scan lines run up the image

This grid of scan lines defines the initial access pattern for pixels in the image. If no features of interest are detected then these are the only pixels processed in the image. The scan lines typically cover about 6,700 pixels, or twenty percent of an image. More pixels in the vicinity

of features and objects of interest are accessed during the image processing but the majority of pixels are not processed at all.

Some scan lines are also constructed dynamically. After processing of the initial scan line pattern described above, more information is available about the information distribution over an individual image. It is likely that additional processing in the vicinity of the previously detected features will lead to detection of further features, since there is significant spatial locality in edge information. Hence, if fewer than seven ball features (see section 3.3.1) are found in the initial pass then an extra fixed-size grid of scan lines is constructed around each of the detected features.

3.2 Colour Classification

Although this implementation attempts to move away from reliance on statically classified colour such classifications nonetheless provide useful information. Given the specification of the four-legged league domain in terms of objects of uniform colour, symbolic segmentation is used extensively in landmark detection and provides useful confirmation of features detected through other methods. However, only those pixels selected for examination by the sub-sampling approach are ever classified.

Because of the poor quality sensor on the ERS-7, correction is applied for chromatic distortion (Xu, 2004) and the particular robot used to capture images is treated as the reference robot for linear adjustment between robots as described in (Lam, 2004). These corrected images are then loaded into a classification application for classification.

A simple weighted Parzen kernel classifier provides segmentation. Our experience with this classifier suggests that incremental, online training is more useful than increased classification accuracy.

3.3 Edge Feature Detection

Feature detection takes place over the scan lines described in section 3.1. Features of the ball, field, lines and obstacles are detected in the vertical and low horizontal scan lines. These scan lines are processed from bottom to top (or left to right) and each pixel is compared with the previous pixel in the scan line. Rapid changes in the Y, Cb or Cr values indicate edges in the image, and particular edge vectors are recognised as being features of a particular environmental object.

3.3.1 Detecting Ball Features

Ball features represent transitions between the orange ball and the green field or white field lines. Occasionally transitions into the yellow or blue goal are also encountered and the majority are correctly detected by the procedure outlined below.

For each pixel in turn running up the scan line let (y, u, v) be the values of the Y, Cb and Cr channels respectively for this pixel, (dy, du, dv) be the difference between the channels of this pixel and the previous (i.e., next lower) pixel, and sum be the sum of the absolute values $|dy| + |du| + |dv|$. A ball feature satisfies:

1. $sum >= 20$ (a minimum gradient over all channels), and
2. $|du| > 15$ (a minimum gradient in the Cb channel), and

3. either:
    a. $dv = 0$, or
    b. $|du/dv| < 4$ (slope of Cb less than four times slope of Cr), or
    c. sign(du) = -sign(dv) (Cb changes in opposite direction to Cr).

In order to avoid confusion between strong specular reflection and white field lines a ball feature must also satisfy $y < 180$. Note that these rules capture transitions both towards and away from orange without any explicit notion of orangeness.

These features are then confirmed by consultation with the statically classified colours of nearby pixels. While processing a scan line, information about the direction of the scan line in the image is calculated. For these tests this is simplified to the nearest basis direction: up, down, left or right. The value of du calculated above gives the direction of change at this pixel: if $du > 0$ then the transition is towards orange; if $du < 0$ then the transition is away from orange. A line of five pixels beginning at the pixel under test and progressing in the direction of the centre of the ball (simplified to a basis direction) are examined. In order for the transition to be confirmed as a ball edge the classified colour of these pixels must satisfy:

1. at least three are classified orange, and
2. no more than one is classified red, and
3. no more than one is classified pink, and
4. no more than one is classified yellow.

A consistent run of orange is not required; any three from the five pixels may be orange. Thus we confirm that there are at least some orange classified pixels where they would be expected. Note that the pixel under test is not required to be classified orange; in fact it is quite often the case that the very edge pixels are somewhat blurred and take on a classification of white or yellow.

From the images in Figure 2 it can be seen that pixels near the upper edge of the ball maintain the correct classification under the widest range of lighting intensities. This test therefore favours transitions at the upper edge of the ball. The lower part of the ball deteriorates to pink or red quite quickly, and indeed this is a major problem with purely colour-based blobbing approaches. This test will continue to recognise edges so long as three out of five pixels are classified orange, but thereafter will reject the transitions as likely spurious. Heavily shadowed edges on the lower half of the ball are detected with a colour based approach outlined in section 3.4.3.

These colour-based tests are necessary because the transition tests are overly lax. The complexity of the transition tests here can be likened to the simple thresholding that was previously used for colour segmentation. The region satisfying the tests is not well fitted and includes many transitions outside the desired set. This method was chosen because it is simple enough to provide an easy implementation and demonstrate the validity of the concept without introducing many unnecessary complexities; future approaches could adopt more complex transition definitions to reduce reliance on colour segmentation even further.

3.3.2 Detecting Field Lines

Field line features represent transitions between the green field and white field lines or boundaries. For each pixel in turn (after the first) running up the scan line define (y, u, v), (dy, du, dv) and *sum* as for ball feature detection in the previous section. A field line feature satisfies:

1. $|y| > 32$ (a minimum value of Y), and
2. $|dy| > 15$ (a minimum gradient in Y), and
3. $|du| < 40$ (a maximum gradient in Cb), and
4. $|dv| < 40$ (a maximum gradient in Cr), and
5. $|du| < 4$ or \$sign(du) = sign(dv)\$ (a small gradient in Cb, or Cb and Cr slope in the same direction).

The direction of change may be calculated in a similar fashion to that for ball features, using the indicator dy rather than du. No symbolic colour tests are applied to field line edges. Note that this test is likely to misrecognise the boundary between robots and the green field as being a field line edge; no attempt is made to prevent this since in our implementation such noisy data is handled robustly in the localisation module (Sianty, 2005).

In addition to detecting field lines, a sparse sampling of each image is made for the purpose of detecting the green of the field, as an aid to higher-level localisation systems. Every 32 pixels along half of the full scan lines a check is made: four line segments of nine pixels with a mutual intersection at the centre of each line segment form an axis-aligned star shape containing 33 pixels. If at least two-thirds (22) of these pixels are classified as green then the centre pixel is marked as a *field-green* feature.

3.3.3 Detecting Obstacles

The method for detecting obstacles presented here differs from most other attempts. This approach specifically detects the shadows on the field caused by objects lying on it. While processing a scan line the Y value at each pixel is tested against a threshold value (we used 35). If the Y channel falls below this threshold the pixel is classified as an obstacle. A maximum of five pixels are classified as obstacles on any one scan line.

To avoid false obstacles caused by the ball, the robot itself or human referees a state machine keeps track of the classified colour of pixels on the scan line as they are processed. The twenty pixels immediately above a candidate obstacle are tested: if more than ten green classified pixels or five orange classified pixels are encountered then the obstacle candidate is discarded.

The results of ball, line and obstacle detection are shown in Figure 7.

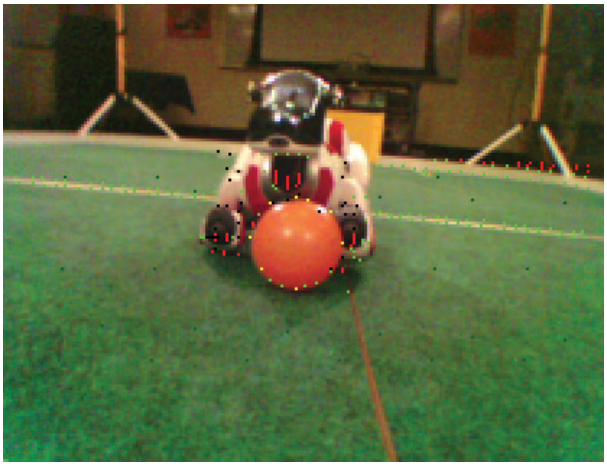


Fig. 7. The results of feature detection over a sample image. Ball edge features are displayed as yellow points, line edge features as light green points, obstacles as red points and field-green features as dark green points. Black points represent features that have been discarded due to checks applied after initial detection

3.4 Symbolic Colour Feature Detection

As noted earlier, colour remains an important part of the four-legged league domain, and this approach continues to use symbolic colour classification for detection of beacons and goals, using a sub-sampled approach rather than blobbing. Landmark detection takes place over the upper horizontal scan lines described in section 3.1. These scan lines are scanned left to right and each pixel in turn classified into one symbolic colour via a static colour segmentation as outlined in section 3.2. Colour is an appropriate indicator for the landmark objects as they are far less subject to variations in lighting during a four-legged league match than on-field objects such as the ball. Although the colour segmentation must be tuned for a specific environment, the perceived colour of beacons and goals changes little during the course of a match.

3.4.1 Detecting Landmarks

A state machine tracks the number of consecutive pixels found of each of pink, yellow and blue along horizontal scan lines, along with the start and end points of these runs of colour. Up to one pixel of "noise" (any other colour) is tolerated. Beacons are detected by the pink square that appears on all beacons. A run of five consecutive pink pixels (plus one pixel of noise) in a scan line creates a *beacon feature*. Goals are detected by their uniform colour of pale blue or yellow. A run of twelve blue or yellow pixels (plus one pixel of noise) creates a blue or yellow *goal feature*. These features are passed to the object recognition system for further processing.

A slight modification to the thresholds is made when the robot's head is held low and the horizon coincides with the top of the image, such as when the robot's head is controlling the ball. Pixels near the top border of the image are subject to significantly more noise (mainly

due to chromatic "ring" distortion and its correction) than those near the centre of the image, and are more likely to be misclassified. Thus, when the robot holds its head down over the ball and is searching for the goal an additional "noise" pixel is allowed in goal features.

3.4.2 Detecting the Wall

While the area outside the green carpeted field is undefined by the four-legged league rules there is often a low wall or region of uniform colour surrounding the field. It is advantageous to detect this to allow filtering of features and objects appearing off-field. A state machine tracks the number of green, grey or white, and other-coloured pixels encountered during the scanning of each vertical scan line. A wall feature is detected by a series of four green classified pixels (allowing one pixel of noise) followed by a series of at least five white or grey classified pixels, followed by a series of two pixels of other colours (allowing two pixels of noise). The requirement for non-green above the white pixels prevents close field lines being detected as walls. A wall feature is created midway between the start and end of the white/grey pixel series. A wall line is constructed using a random sample consensus, or RANSAC, algorithm (Fischler & Bolles, 1981).

3.4.3 Detecting Faint Edges

There are a number of cases where edges in the image become blurred so the ball and field line feature detection methods outlined above become less effective. The most common cause is motion blur: when either the camera or objects in the environment move quickly the result is a blurred image with indistinct edges. In such images the thresholds for change required for feature detection may not be met, since the transition is spread over many pixels. In these cases an alternative ball feature detection method is used, based on segmented colour. While symbolic colour classification is susceptible to changes in lighting it is fairly robust to blurring; edge detection exhibits the opposite tendencies.

A state machine keeps track of the number of orange, maybe-orange (i.e. pink, red and yellow) and non-orange (the remainder) pixels detected along a scan line. A transition is detected between non-orange pixels and orange pixels, possibly with a number of intervening maybe-orange pixels. Three consecutive orange pixels are required to satisfy as an orange region, although the number of maybe-orange pixels before this is unbounded. If the transition is into an orange region a feature is created at a point midway between the last detected non-orange pixel and the first orange or maybe-orange pixel. If the transition is away from orange a feature is created at a point midway between the last detected orange pixel and the first non-orange pixel, so long as these two points are within six pixels of each other. On transitions away from orange the maybe-orange pixels are ignored since such pixels usually occur on the lower half of a valid ball.

3.5 Object Recognition

Object recognition takes place over the features as recognised in the previous section. Object recognition involves grouping features related to the same real-world object and extracting the important attributes of these objects such as position in the image, position in the environment, heading, elevation, orientation and variances over these attributes. The results

of the early stages of object recognition are used to focus computational effort when few features are initially detected.

3.5.1 Beacon Recognition

Beacon recognition comprises grouping the detected pink features into candidate beacons then searching above and below the pink region for the other characteristic beacon colour (pale blue or yellow). Beacon features (which are horizon-parallel line segments) are grouped by merging adjacent "overlapping" features. Two features overlap if a line perpendicular to the horizon can be drawn that intersects with both features.

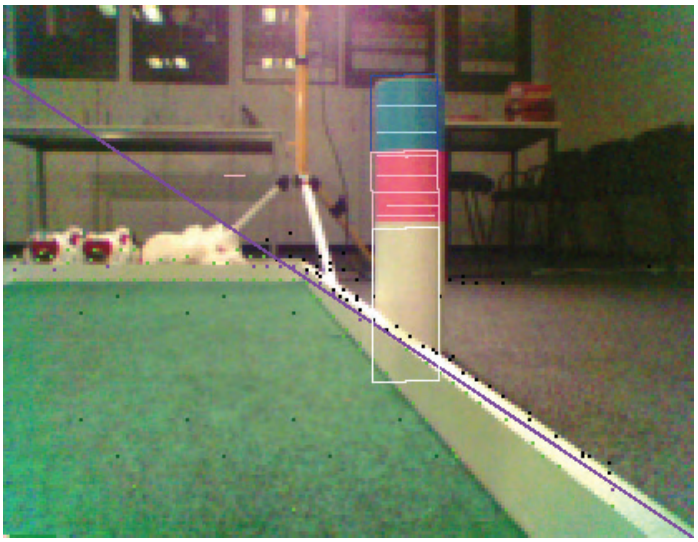


Fig. 8. A recognised beacon. The beacon features are displayed as horizontal pink lines. The white field wall has also been detected, displayed as a purple line

A local search is then performed to classify each group of beacon features as one of the four possible beacons, or as falsely detected features. Essential properties such as apparent height and heading are deduced from the beacon's geometry through simple methods which are not relevant here. Only a single check is performed to confirm the validity of a candidate beacon (c.f. the sixteen checks listed in (Lam, 2004)). The centroid of the beacon must not be below the horizon by more than 25 pixels. This check rules out some invalid beacon features that might be detected by excessive pink occurring in the ball or red team uniform.

3.5.2 Goal Recognition

Goal recognition comprises grouping the detected blue and yellow features into candidate goals. Goal features are grouped by merging adjacent "overlapping" features in the same way as beacon features, relaxed to allow up to one scan line separating features to be merged. However, it is possible for two distinct regions of the one goal to be visible as

shown in Figure 9. Thus goal feature groups are also merged if they contain features on the same scan line.

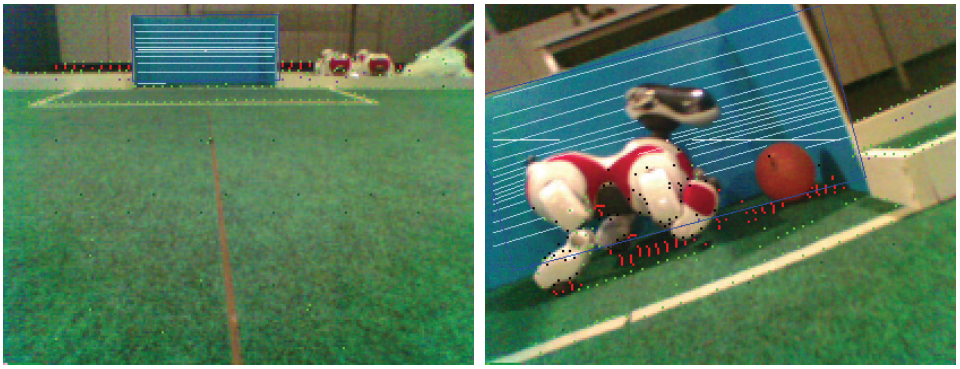


Fig. 9. (a) A recognised goal. (b) The goal is often occluded by a robot, but grouping of features leads to correct recognition, even if the goal is divided in half. The two possible gaps for shooting are indicated by horizontal white lines. The recognised goal is assumed to be aligned with the horizon, so no attempt is made to detect the goal outline

As for beacons, the essential properties of a goal are deduced from its apparent geometry through simple methods that are not relevant here. A few checks are made to confirm the validity of a candidate goal: the aspect ratio of the goal is checked to make sure it forms a sensible shape; a goal must not appear further than twenty pixels above the horizon; and a number of pixels underneath the candidate goal are tested for colour. If very few are found to be green the goal is rejected. The goal is also rejected if many are found to be white, as might occur in the blue or yellow patch of a beacon. The aspect ratio and colour checks are ignored when the robot holds its head down low while controlling the ball: both are likely to trigger falsely, and goal detection is of utmost importance in these cases.

3.5.3 Ball recognition

Ball recognition is performed after beacon and goal recognition has completed, allowing obviously spurious ball features to be ignored. If fewer than seven valid ball features have been detected additional scan lines are first created and scanned near existing ball features as outlined in section 3.1.

Ball recognition involves estimating the outline of the ball from the detected features. A circle is fitted to the ball edge features using a generalisation of Siegel's repeated median line fitting algorithm to circles, as described in (Mount & Netanyahu, 2001). Under the assumption that the majority of the points to fit lie on a circle, this algorithm claims robustness to up to 50% outlying data. Slight modifications are made to account for the fact that, due to motion blur, the ball frequently does not appear perfectly circular. This approach assumes that there is at most one ball in view.

Given the parameterised equation of a circle as $(x - a)^2 + (y - b)^2 = r^2$ all triplets of features (i, j, k) are considered, from a minimum of four features. Each triplet determines a circle by

the intersection point of perpendicular bisectors constructed to the chords formed by the triplet. The parameters (a, b) are calculated separately as in (2): for each pair (i, j) take the median of the parameter over all choices for the third point, k; for each i take the median parameter over all choices for the second point, j; and take the result as the median over i.

$$a = \text{med}[i]\ \text{med}[j \neq i]\ \text{med}[k \neq i,j]\ \ a_{i,j,k} \qquad \text{b} = \text{med}[i]\ \text{med}[j \neq i]\ \text{med}[k \neq i,j]\ \ b_{i,j,k} \tag{2}$$

In contrast to the algorithm presented in (Mount & Netanyahu, 2001) the radius is calculated from a single median after the positional parameters have been calculated, as given by (3). This aids stability in the presence of a large number of outliers. If at least seven features are present then the middle three are averaged to give the radius. This averaging helps to reduce jitter induced by image noise.

$$r = \text{med}[i]\ r_i \tag{3}$$

As for landmark features, the important properties of the ball may be derived from its position and size. Three checks are performed to ensure that the recognised ball is valid. Two geometry tests are applied: the ball is discarded if it appears above the horizon by more than ten pixels; and it is discarded if its radius is unreasonably large (two thousand pixels). Finally, a symbolic colour based test is applied: a valid ball should contain some orange pixels within its circumference. This test is only a validity check; the coloured pixels are not used for deriving the ball's properties.

If the ball is small (a radius of fewer than ten pixels) then a square of side length equal to the ball's radius is constructed around the centre of the ball, and all pixels in this square are colour-classified. Otherwise, features that lie within ten pixels of the circumference are randomly chosen and a line segment is constructed between the feature and the ball centroid. The pixels along this line segment are classified, up to a total of one hundred pixels over all features. In both cases counts are maintained of the number of orange, red, pink and green classified pixels encountered. If fewer than three orange pixels are encountered the ball is discarded. If the radius of the ball is greater than ten pixels and fewer than twelve orange pixels are found the ball is discarded. This colour checking is displayed as a cross shape of accessed pixels over the ball in Figure 11.

This approach allows accurate recognition of the ball under a range of conditions. While it is limited by an assumption that there is only one ball present in the image, the ball may be detected when blurred or skewed, occluded or only partially in frame. Figure 10 shows recognition in a number of these cases. The repeated median algorithm exhibits $\Theta(n^3)$ computational complexity in the number of features. Since n is usually small this remains appropriate; this implementation limits the number of features used to a maximum of seventeen, more than enough to achieve an accurate fit.

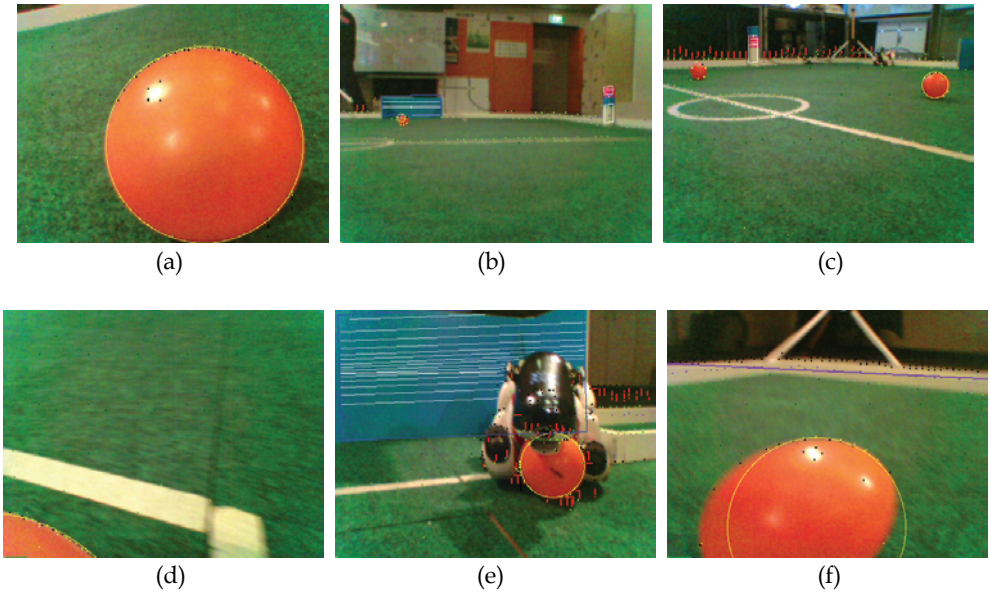


Fig. 10. Ball recognition in a number of different situations. Ample information available in (a) allows for a very tight fit. The repeated median estimator continues to be accurate at long range in (b), although at such distances precise information is less important. Noise features caused by a second ball in (c) are ignored. A ball lying mainly outside the image frame in (d) is still detected accurately, as is a ball partially occluded by a robot in (e). A combination of motion blur and skewing in (f) lead to a questionable fit

3.5.4 Mutual Consistency

Once object recognition is complete a small number of checks are made to ensure that the perceived objects are mutually consistent. These checks are outlined below, in order of application.

1. The two goals cannot be simultaneously perceived. If they are, the one comprising the fewest features is discarded. If they have the same number of features the goal with the highest elevation is discarded.
2. A goal centroid cannot appear inside a beacon. If it does, the goal is discarded.
3. A beacon centroid cannot appear inside a goal. If it does, the beacon is discarded.
4. A goal cannot appear above a beacon by more than ten degrees. If it does, the goal is discarded.
5. Diagonally opposite beacons cannot simultaneously be observed. If they are, both are discarded.
6. Beacons at the same end of the field cannot appear within thirty degrees of each other. If they are, both are discarded.
7. The ball cannot appear above a goal. If it is, the ball is discarded.

8. The ball cannot appear above a beacon. If it is, the ball is discarded.

These checks conclude visual processing for one frame. The information extracted from the recognised features and objects is passed to the localisation and behaviour modules.

4. Evaluation

In our domain of robot soccer, the accuracy and robustness of a vision system reflects strongly in the performance of a team of robots in a competitive soccer match. In a limited sense this is the most valuable method of evaluation. The goal of the vision system is to have robots play the best soccer and a vision system that results in a team consistently winning matches is better, in some sense, than a system that does not. However, it is difficult to hold other variables constant, so, while being the most important test of a system's quality, this test is also the most subject to random variation, noise and external influences. The performance of a team depends upon the performance of the opposing team and the environment both on and off the field. Further, this method of evaluation is highly non-repeatable; it is impossible to substitute an alternative vision system and have the same match play out with the exception of changes directly related to vision. Nevertheless, evaluation by playing matches remains an important measure of progress. If otherwise identical code is used in both teams over a number of competitive matches the influence of the vision systems may be observable in qualitative terms.

Behavioural evaluation of a single robot agent is another important method of evaluation. Agent provides information about the performance of its vision system in particular circumstances, with much of the interference caused by team-mates and opponents removed. For example, an agent's behaviour may provide clear indication of whether or not it can see a given object. It is possible to display informative indicators in the form of LEDs, or such information might be accessible via a remote data stream. Thus the quality of a single agent's visual information may be subjectively assessed. Alternatively, two independent agents might be active simultaneously, and the behaviour and data streams from each compared. Although each agent is processing different input, over time any significant systematic differences in visual processing will become apparent.

Single agent tests bear some semblance of repeatability: situations can be constructed and the performance of agents evaluated over a number of similar trials. Single agent tests are particularly useful for evaluating small modifications to a vision system. The RoboCup four-legged league also presents a number of technical challenges that are useful in evaluating an agent's vision system (RoboCup, 2005).

One of these was the *Variable Lighting Challenge* which explicitly tests an agent's vision system's robustness to changes in lighting conditions over time. In this challenge an agent must consistently recognise the standard RoboCup objects while the field lighting changes in different ways. While still heavily dependent on higher level behaviours this challenge tests robustness of image processing systems to shifts in lighting intensity, colour temperature and dynamic shadows. For an agent to perform well in this challenge it necessarily requires a highly robust vision system.

For yet more detail, an agent's image processing system may be compiled and executed in isolation on a standard PC ("offline"), where its performance for particular images or image sets may be qualitatively and quantitatively evaluated. This allows direct observation of the

performance of a system in precise circumstances, and in many cases provides insight as to why a system behaves as it does. A log file of the image data as sensed by the camera may be captured then played back, allowing the performance and internals of the system to be visualised and examined in detail.

Unlike the two previous methods, offline evaluation is fully repeatable and allows multiple image processing systems to process exactly the same data and the results to be compared with each other and a human-judged ground truth. This allows direct comparison of similar vision systems, and comparison against a subjective ideal interpretation. Alongside single agent evaluation this method is effective at comparing modifications to a vision system.

These three evaluation methods present trade-offs between their importance, accuracy and ease of administration. This approach, like many others, is designed to perform as well as possible in one general environment, here the competitive environment of the RoboCup four-legged league. In one sense this real-world performance is the most important evaluation measure.

The *rUNSWift* team using this system placed first in the RoboCup 2005 Australian Open and third in the RoboCup 2005 four-legged league World Championships. In 2006, *rUNSWift* were again Australian Champions and came second in the international RoboCup competition. The extremely successful GermanTeam also use elements of sub-sampling and colour relationships (Röfer et. al., 2004), so these methods are clearly a valid and successful approach to four-legged league vision. While it is extremely difficult to directly compare the accuracy and robustness of different vision systems, this chapter presents some results pertinent to this approach.

4.1 Colour Relationships

A focus on the relationships between neighbouring pixels, rather than symbolically classified colour, leads to a great improvement in robustness. As demonstrated in Figure 2, variations in lighting quickly lead to the degradation of static colour classifications, while colour-space relationships remain far more constant. This was confirmed by the ability of the system presented here to perceive the ball and field lines in a range of environments without re-calibration. In fact, the colour-difference vectors in this implementation were not modified after their initial calculation despite large changes in lighting intensity, colour temperature and even camera settings across four different field environments in which the implementation was tested.

This system was also used without significant modification for the 2005 four-legged league Variable Lighting Challenge, in which *rUNSWift* placed third in the world. The *rUNSWift* agent was able to perceive the ball for the majority of the challenge, especially as light intensity fell. However, when lighting was made significantly *brighter* than normal game conditions some red/orange confusion hampered performance. These performances demonstrate robustness to dynamically variable lighting.

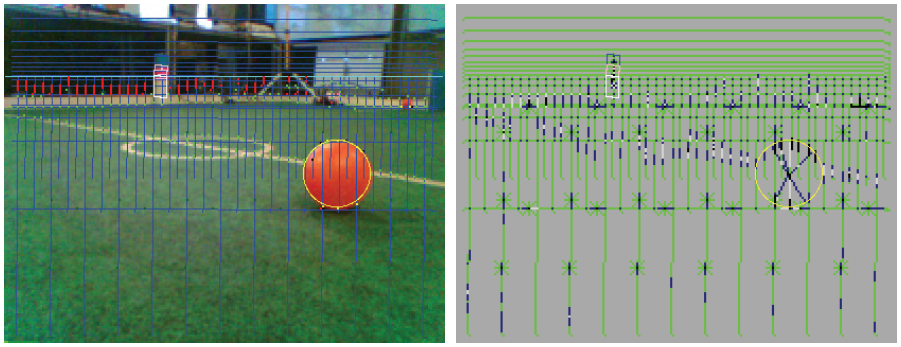
The reduced reliance on colour segmentation has also led to reduced complexity of and effort required for colour segmentation. Coupled with an interactive classification tool and the efficiency and flexibility of the kernel classifier, colour segmentation can be performed with more sample data and with less effort than previous approaches. The sub-sampling system is robust to a sloppy segmentation, although in a competitive environment effort should be made to achieve a stable classification. The final rounds of the RoboCup 2005

four-legged league competition were performed with a segmentation based on over one thousand images, trained in a few hours of human expert time and tested only in the few hours before the competition games.

4.2 Sub-sampling

Focussing image access on regions of high informational value leads to efficiency gains, as time is not spent processing redundant information within uniformly coloured regions. Domain knowledge allows targeting of high information areas for dense sampling, while regions of typically low information may be sampled more sparsely. A dynamic element to image sampling allows even more directed processing based on the information gained in earlier stages. Most potential invalid object candidates are implicitly rejected: only regions of an image likely to contain a particular feature are sampled for it, reducing the number of invalid objects it is possible to recognise falsely.

Pixel access to a typical image is shown in Figure 11, where it can be seen that areas containing useful information are sampled with higher density than less informative regions. The scan line pattern typically covers only twenty percent of the pixels in an image, with dynamic access processing a little more depending on the information gained in the initial pass.



(a) (b)

Fig. 11. Pixel access profiling for a typical image. (a) shows the original YCbCr image with scan lines, features and recognised objects displayed. A pixel access profile is shown in (b). Dark grey represents pixels that have not been accessed, green represents pixels that have been accessed once, and other colours represent pixels accessed multiple times. Note how access is clustered around high-information areas of the image such as the ball, field lines and beacon

Actual processor time consumed by this approach is in fact similar to previous methods, with complete visual processing typically ranging from 10ms – 15ms, depending on image content. There are a number of reasons why this is not significantly lower. Firstly, the scan line pattern describes essentially random access to the image. Approaches that sample the entire image may make good use of processor caching and similar optimisations, but these are lost when the image data is accessed in this pattern. Similarly, image correction and colour classification are applied on the fly to pixels as they are accessed, giving random

access to correction and classification tables. Disabling chromatic distortion correction leads to a small gain in efficiency. Removing all colour classification-based access leads to large gains but hampers the still colour-based beacon and goal detection. Finally, it is advantageous to use all available compute power to construct the best possible information, so optimisation efforts were not made beyond those necessary to reach this level of performance.

4.3 Object Recognition

Moving to a sub-sampled approach makes traditional blobbing approaches to object recognition impossible. Instead, more sophisticated object recognition procedures are required to form objects from sparsely detected features. A significant advantage to this approach is the reduction in complex, hand-coded validity tests for objects. (Lam, 2004) explicitly listed thirty such checks involved in blob-based detection of the beacons, goals and ball for *rUNSWift* in 2004, but the code used at the 2004 RoboCup competition contained many, many more and spanned thousands of lines of code. The fifteen validity tests outlined in section 3.5 represent the entirety of such checks in this implementation; in general this approach requires far fewer domain-specific tests. This leads to more efficient object recognition, allowing for more sophisticated and computationally expensive approaches to obtaining accurate information.

4.4 Shortcomings

As noted above, while the efficiency of the sub-sampling system as implemented is well within acceptable ranges for the four-legged league, improvements may be made by optimisations to the image access pattern or a reduction in colour-based tests. Random access to both the image data and correction and classification tables make poor use of processor caching features.

The beacon detection implemented in this approach leaves room for improvement. Being colour-based, it still has the deficiencies associated with reliance on a finely tuned static colour segmentation, but makes use of less information than other approaches. While the accuracy was adequate for our purposes there are gains to be made in more accurately fitting the beacon model to the available data. However, it is likely that the coloured beacons will be removed from the four-legged league field definition in the near future as the league attempts to move towards yet more realistic environments.

Edge-based methods in general respond poorly to blurred images, which are not uncommon for legged robots. Significantly blurred images, such as those obtained while contesting positions with robots from the opposing team, are poorly processed. Section 5.1 suggests some possible improvements.

5. Conclusion

The high level of detail and dependence upon environmental conditions makes creation of an accurate, robust and efficient robot vision system a complex task. Rather than a traditional focus on colour segmentation of entire images, the vision system outlined in this chapter moves towards detection of local relationships between a subset of the image pixels.

While systems based on colour segmentation are brittle and respond poorly to variations in lighting, the relationships between colours exhibit independence to lighting conditions, leading to a more robust system.

This chapter has demonstrated the stability of colour relationships under variations in lighting, particularly intensity. This system continues to provide reliable information under a range of environments and variations in lighting conditions, leading to a more robust feature detection system. A reduced dependence on colour segmentation also leads to reduced calibration time and eases the transition between different environments.

The effectiveness of sub-sampled image processing approaches has also been demonstrated. The information contained in a typical image is not uniformly distributed over its area; neighbouring pixels are highly correlated. In order to more effectively make use of constrained resources, regions of the image typically high in information content are sampled more densely than areas of typically low informational value. A dynamic element to the sampling allows an even tighter focus on useful regions in any given image. The sub-sampled approach leads to efficiency gains and implicit rejection of much unwanted data.

Given the information provided by a sub-sampled, edge-oriented approach, this chapter has also described robust recognition of objects for the four-legged league from relatively sparse features. Object recognition is performed over a discrete set of features corresponding to particular features in the sampled areas of each image. Domain knowledge allows accurate recognition under a range of conditions.

The successful results obtained by the approach presented in this report outline a path to more robust, lighting-independent robot vision. While there is still much work to be done, significant improvements in robustness have been gained by a shift of focus away from statically classified colour towards detection of colour relationships and transitions. Unlike approaches based on selection between multiple colour tables this approach gracefully caters for unexpected conditions without the need for additional calibration efforts. In contrast to existing dynamic classification approaches, this implementation allows for potentially arbitrary complexity in colour and colour-gradient classification without the need for adjustments calculated from past observations.

These changes in focus are likely to be applicable to other robotic vision domains where uniform colour is a primary differentiator for important objects. It is immediately applicable to other RoboCup leagues, and to other domains requiring robust object recognition under tight constraints on efficiency.

5.1 Future Work

While the system as implemented has been successful, a number of areas may provide fruitful future research. The image access patterns described in this report focus on areas of typically high information, but the concept can be taken further. A focus on dynamic processing, where image access is determined by information obtained by previous processing, could lead to even further gains in efficiency. The scan lines themselves could be sub-sampled, performing some variation of an interval search for transitions, sampling every pixel only around areas containing edges. In addition, temporal awareness might be used to further hone access patterns; areas of recently high information value might be sampled first and more densely than areas of little recent value.

Blurring remains a problem for transition-sensitive techniques. Consulting the relationships between pixels at greater intervals than immediate neighbours might allow detection of softer edges. A blurred transition between two colours will have a similar profile to a sharp transition if viewed over more widely spaced pixels. Conversely, on sharp transitions more detail is available from the ERS-7 camera than is currently used. The Y channel is sampled at twice the resolution of the chroma channels, and this information might be used to improve the accuracy of the location of detected features.

Perhaps the most clearly beneficial direction for future work is in generalisation of the colour gradient space regions used for feature classification. Strong parallels may be drawn between colour segmentation and the classification of transitions, the major difference being that the transitions are invariant under linear shifts in the colour-space. The classification procedure for colour-gradient vectors used in this implementation is of a similar complexity to early colour segmentation routines. Applying present-day colour segmentation methods to gradient classification would likely lead to a great improvement in the accuracy of detected features and further reduce reliance on colour segmentation.

Finally, there are some possibilities for improvement upon the object recognition methods presented in this report. The assumption that there is only one ball, and that it appears as a circle, limits both the flexibility and robustness of ball recognition. Some variation on feature clustering would serve the dual purposes of allowing recognition of multiple balls and rejection of gross outliers. The repeated median circle estimator is effective but its computational complexity prevents use of abundant features. The addition of an optimisation step such as least squares approximation may prove more efficient and allow fitting of more general ellipsoids.

National ICT Australia (NICTA) is funded by the Australian Government's Department of Communications, Information Technology, and the Arts (DICTA) and the Australian Research Council through Backing Australia's Ability and the ICT Research Center of Excellence programs.

6. References

Bruce, J.; Balch, T & Veloso, M. (2000). Fast and inexpensive color image segmentation for interactive robots, *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-00)*, volume 3, pp. 2061–2066, 0-7803-6348-5, Takamatsu, Japan, October 2000, IEEE Computer Society, Los Alamitos, CA, USA

Brusey, J. & Padgham, L. (1999). Techniques for obtaining robust, real-time, colour-based vision for robotics, *RoboCup-99:Robot Soccer World Cup III*, Veloso et. al. (Ed), 63-73, Springer, 3-540-41043-0, Berlin

Burt, P. J. & Adelson, E. H. (1983). The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, Vol. 31, No. 4 (April 1983), pp. 532–540, 0096-2244

Chen, J.; Chung, E.; Edwards, R.; Wong, N.; Hengst, B.; Sammut, C. & Uther, W. (2003). *Rise of the AIBOs III - AIBO revolutions*. Technical report, 2003, The University of New South Wales, Sydney, Australia

Fischler, M. A. & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, Vol. 24, No. 6 (June 1981), pp. 381–395, 0001-0782

Jüngel, M.; Hoffmann, J. & Lötzsch, M. (2003). A real-time auto-adjusting vision system for robotic soccer, *7th International Workshop on RoboCup*, Polani et. al. (Ed.), 214-225, Springer-Verlag, 978-3-540-22443-3, Berlin

Lam, D. (2004). Visual object recognition using Sony four-legged robots. Honours thesis, 2004, The University of New South Wales, Sydney, Australia

Mount, D. M. & Netanyahu, N. S. (2001). Efficient randomized algorithms for robust estimation of circular arcs and aligned ellipses. *Computational Geometry*, Vol. 19, No. 1 (June 2001), pp. 1–33, 0925-7721

Pham, K. (2004). Incremental Learning of Vision Recognition using Ripple-Down Rules. Honours Thesis, 2004, The University of New South Wales, Sydney, Australia

Quinlan, M. J.; Murch, C. L.; Moore, T. G.; Middleton, R. H.; Li, L. A. Y.; King, R. & Chalup, S. K. (2004). The 2004 NUbots team report. Technical report, 2004, The University of Newcastle, Newcastle, Australia

Roberts, L. G. (1965). Machine perception of three-dimensional solids. *Optical and Electro-Optical Information Processing*, J.T. Tippet et. al. (Ed.), pp. 159–197, M.I.T. Press, Cambridge, Massachusetts

Röfer, T.; Laue, T.; Burkhard, H.; Hoffmann, J.; Jüngel, M.; Göhring, D.; Lötzsch, M.; Düffert, U.; Spranger, M.; Altmeyer, B.; Goetzke, V.; Stryk, O.; Brunn, R.; Dassler, M.; Kunz, M.; Risler, M.; Stelzer, M.; Thomas, D.; Uhrig, S.; Schweigelshohn, U.; Dahm, I.; Hebbel, M.; Nistico, W.; Schumann, C. & Wachter, M. (2004). GermanTeam 2004. Technical report, 2004, Universität Bremen, Humboldt-Universität zu Berlin, Technische Universität Darmstadt, University of Dortmund

Shammay, J. (2005). Real-Time Shared Obstacle Probability Grid Mapping and Avoidance for Mobile Swarms. Honours thesis, 2005, The University of New South Wales, Sydney, Australia

Sianty, A. (2005). rUNSWift RoboCup Challenges 2005, Honours thesis, 2005, The University of New South Wales, Sydney, Australia

Sobel, I. E. (1970). Camera Models and Machine Perception. PhD Thesis, 1970, Stanford University, California, USA

Sridharan, M. & Stone, P. (2005). Towards illumination invariance in the legged league. *RoboCup 2004: Robot Soccer World Cup VIII*, Nardi et. al. (Ed.), pp. 196–208, Springer, 978-3-540-25046-3, Berlin

Stone, P.; Dresner, K.; Fidelman, P.; Jong, N. K.; Kohl, N.; Kuhlmann, G.; Sridharan, M. & Stronger, D. (2004). The UT Austin Villa 2004 RoboCup four-legged team: Coming of age. Technical report, 2004, The University of Texas, Austin, USA

Veloso, M.; Chernova, S.; Vail, D.; Lenser, S.; McMillen, C.; Bruce, J.; Tamburrino, F.; Fasola, J.; Carson, M. & Trevor, A. (2004). CMPack'04: Robust modelling through vision and learning. Technical report, 2004, Carnegie Mellon University, Pittsburgh, USA

Wasik, Z. & Saffiotti, A. (2002). Robust color segmentation for the RoboCup domain, *Proceedings of the 16th International Conference on Pattern Recognition (ICPR-02)*, pp. 651-654, 0-7695-1695-X, Quebec, Canada, August 2002, IEEE Computer Society, Los Alamitos, CA, USA

Xu, J. (2004). rUNSWift thesis – low level vision. Honours thesis, 2004, The University of New South Wales, Sydney, Australia

RoboCup (2005), Technical challenges for the RoboCup 2005 legged league competition. http://www.tzi.de/4legged/pub/Website/Downloads/Challenges2005.pdf

TecRams (2004), Team Report (2004). Technical Report, 2004, Tecnologico de Monterrey Campus Estado de Mexico, Mexico

24

Robot Localisation Using a Distributed Multi-Modal Kalman Filter

Oleg Sushkov and William Uther
University of New South Wales and National ICT Australia
Australia

1. Introduction

It is extremely important for any mobile robotics system to know where it is. If an agent does not know where it is or where the objects around it are, meaningful actions become very difficult to perform. The main obstacle to efficient and accurate robot localisation is noise. Noise is present in every part of a robotics system, both in the sensors as well as in the actuators. In a world without noise, with perfect sensors and actuators, localisation would be a relatively simple task.

It is also important to note that localisation is not necessarily restricted to determining the pose of a robot, but can also include tracking the state of other objects. Taking the Robocup domain as an example, localisation could include discovering the ball position and velocity and team-mate robot poses as well as the robot's own position.

The core concept of robot localisation is estimating the world state through sensor data. In most situations, the world state is not directly observable in its entirety – the world is only partially observable. In such cases the state of the world must be inferred from the given sensor data, and integrated over time. Different algorithms for robot localisation provide varying ways of incorporating the partial observations of the world state into an internal representation of the complete world state.

In this chapter we discuss one particular method of Robot localisation as applied to the 4-Legged League as part of Robocup, developed by the rUNSWift team for the 2006 competition. We base our system on Bayesian probability theory. The agent keeps a distribution over possible states of the world, and updates that distribution as it moves about and observes the world.

The Bayesian foundation for localisation is extremely general, and hence leaves many choices in implementation. What is the space over which the state is assumed to vary? How is the probability distribution over that space represented? Which observations and actions are used to update the distribution.

2. Bayesian Localisation

As an initial example of Bayesian localization we'll use a small robot in a world made up of a 5 by 5 grid of states. Initially we'll assume that the robot is completely uncertain about its

location, and so all states in the world are given equal probability. As the agent moves, we shift the probability distribution to account for the movement, and blur the distribution slightly as the movement is noisy. The exact amount of movement and blur is recorded by a motion model.

When the agent makes an observation, for example it notes that a wall appears two units north, that observation is processed through a sensor model. This sensor model records the probability of seeing each possible observation in each possible state $P(O \mid S)$. This model is discovered prior to attempting to localise by conducting experiments on the sensors.

With our prior distribution over states, and our sensor model, we can calculate a posterior distribution over states using Bayes' rule. For each state:

$$P(S \mid O) = \frac{P(S)P(O \mid S)}{P(O)} \tag{1}$$

We do not usually know the probability of a given observation, $P(O)$, but luckily that is a constant. As we know that the resulting probability distribution must be normalised, we can ignore $P(O)$ and simply renormalise the resulting distribution.

2. State of the Art

Representing probability distributions as tables of probabilities and doing these calculations individually for each state can be very slow. Luckily, there are common representations for probability distributions which allow efficient updates.

The currently preferred methods of localisation in the 4-legged Robocup league include Monte Carlo Particle Filters, and uni-modal Extended Kalman Filters. By far the most popular of these is the particle filter method. Particle filters approximate a probability distribution with a sample of points drawn from that distribution. Updates are then only required for the sampled points. Some reasons for the popularity of particle filters (Fox et al., 1999) include their ability to handle non-linear observations (where the mapping from observation space to state space is a non-linear function), their quick convergence, and their multi-modal nature (being able to track many different possible positions at the same time, termed Global Localization). The other popular method of robot localisation is the Kalman Filter (Kalman, 1960). This is a state estimation filter which uses a Gaussian probability distribution to approximate the current state of the world. The main advantage of this method is that it is very computationally efficient, being able to compute the updates algebraically in closed form.

In this chapter we describe a system for robot localisation which is a hybrid of the Extended Kalman Filter and the Monte-Carlo particle filter. Our representation for our probability distributions is a weighted sum of Gaussians – similar to having a small set of particles, where each particle is itself a Gaussian. We apply observation and motion updates to each Gaussian particle in the same way as for a standard Kalman Filter. Each Gaussian particle is then weighted according to how well the observation matched the hypothesis. This method combines the advantages of both the particle filter method and the Kalman Filter method. We are able to approximate arbitrary probability distributions (given enough Gaussian

particles), handle high dimensional state spaces, ambiguous observations, and at the same time keep computational costs reasonable.

3. Issues in the Localization Domain

The task of robot localization can be seen as calculating the belief distribution of the robot position with given observations and control updates. Measurement updates are observations of landmarks, in our case distance and heading measurements to the coloured beacons. Control updates are a prediction of the belief state after the robot performs some movement.

The task of robot localisation can vary in difficulty depending on various factors. Each of these factors will influence the choice of algorithm used for the particular situation, since some algorithms are adept at handling certain classes of problems but not others.

We can categorise localisation problems based on the type of measurements available and knowledge of the initial state of the system. Based on these we can classify the problem into either Local or Global localisation. In the case of Local localisation, the probability distribution function has only a single mode, meaning that the localisation algorithm is only required to deal with a small pose error, with the uncertainty confined to a small region around the robot's true pose. In the case of Global localisation, the probability distribution function needs to be able to handle multiple modes, and the algorithm used must be able to handle high uncertainty in the robot pose. This may be due to large errors in robot motion and measurements, or due to the presence of non-unique landmarks. For example, in the situation where there are two identical rooms connected by a corridor, the localisation algorithm must be able handle a probability distribution function which has a mode in each room.

The "kidnapped robot" problem is related to the problem of Global localisation. This arises if at some point in time the robot is taken from its current location and placed in a completely different location. In the case of the Robocup domain, this is especially prevalent, since robots are frequently taken off the field or placed back on the field in a new location. It is very important that the chosen algorithm can deal with "kidnapping" quickly and efficiently. This problem is magnified even further if the place where the robot is replaced has almost symmetrical landmarks when compared to its belief location.

Whether the environment is static or dynamic will also affect the choice of suitable algorithm. In the case of a static environment, the objects which are of concern for the robot are stationary during the operating cycle. A dynamic environment, however, may have moving objects which can affect the robot, or which the robot must interact with. In this case the algorithm must track not only the robot pose, but also the position of the moving objects. When applied to the Robocup domain, we can clearly see that it is a dynamic environment. Specifically, the orange ball is a moving object which we must track. The team-mate robot and opponent robots may also be considered as dynamic objects, but often, as in our case, they are ignored because they are very hard to observe.

The final distinguishing feature of robot localisation that we will consider is the issue of single-robot and multi-robot localisation. The simple case is single-robot localisation, in which case observations are all made by the one robot, only one pose needs to be tracked, and there is no need for communication between robots. Multi-robot localisation offers the advantage of the availability of a greater number of sensors and thus a greater number of

observations. We can use this increased observational power to improve the accuracy of the system as a whole. However, multi-robot localisation also brings with it many challenges. The localisation algorithm used must be able to incorporate the observations of other robots in the system, which in itself brings about issues such as communication lag. In our case, we can use the observations of other robots on the team to better track the position and velocity of the ball, and can also set up correlations in order to be able to use the ball as a beacon, thus improving not only the accuracy of the ball location, but also of the robots pose itself.

4. Kalman-Bucy Filter Algorithm

At its core the Kalman-Bucy Filter is a recursive solution to the discrete-data linear filtering problem. It allows us to estimate the state of a process minimising the squared error. Surprisingly, as already noted, this turns out to be equivalent to a Baysian tracking system when both prior and observation probability distributions are Gaussian.
In the case of Robot localisation, the process is the movement of the robot around the field. The process to be estimated is governed by the stochastic difference equation:

$$x_k = \mathrm{A}x_{k-1} + \mathrm{B}u_{k-1} + w_{k-1} \tag{2}$$

And measurement update:

$$z_k = Hx_k + v_k \tag{3}$$

In the equations above, A is an nxn matrix which relates the process state at the previous time step to the current state in the absence of control input. The nxm matrix B relates the control input u to the process state, and w is the process noise, which is assumed to behave as a Gaussian Distribution.

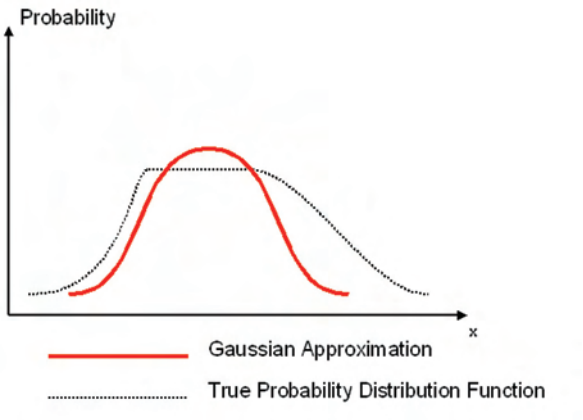


Fig. 1. We approximate a probability distribution function with a uni-modal Gaussian distribution

The Discrete Kalman Filter consists of two distinct steps, the time update and the measurement update. The time update uses the control input to update the belief state of the system, and the measurement update uses possibly noisy observations of the system state to improve the state belief estimate. In terms of the Robocup domain, the time update is derived from the odometry data from the actuators, and the measurement update is derived from the visual sighting of various landmarks around the field such as beacons. The variables which we must keep track of between time steps are the mean vector, and the covariance matrix. The mean vector is the best estimate of the world state, and the covariance matrix is the multi-dimensional measure of the uncertainty of the current estimate.

The time update equations:

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_{k-1} \\ P_k &= AP_{k-1}A^T + Q \end{aligned} \tag{4}$$

The Measurement update equations:

$$\begin{aligned} K_k &= P_k^- H^T (HP_k^- H^T + R)^{-1} \\ x_k &= x_k^- + K_k(z_k - Hx_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned} \tag{5}$$

In the time update step we must do two things, we must update the mean state estimate based on the control input, and we also need to update the covariance matrix P, that is, we need to increase our uncertainty estimate, due to the noise present in the control input.

The measurement update step is a more complicated process. Firstly we compute the Kalman Gain K, this is a measure of how much influence the observation z_k will have on the mean state estimate. For example, if we are very certain of our current estimate and we judge that the observation z_k is very unreliable, then the Kalman Gain K will be close to 0. However, if our uncertainty estimate is very high, that is, we consider the current mean to be very unreliable, and at the same time we consider the measurement to be very accurate, then the Kalman Gain will be close to 1.

After we have computed the Kalman Gain, we can adjust the mean state estimate x_k by moving it in the direction of the Innovation Vector $(z_k - Hx_k)$. The Innovation Vector can be seen as the direction in state space in which the mean vector needs to be shifted in order for it to more closely agree with the current state observation z_k. The more the currently observed state disagreed with the mean estimate, the greater will be the magnitude of the Innovation Vector, while if the mean estimate is in complete agreement with the observation then the Innovation Vector will be 0.

Finally, we must recompute the covariance matrix P, the uncertainty estimate. In general, observations tend to decrease the uncertainty estimate, while control updates tend to increase the uncertainty estimate. The derivation of the update of the covariance matrix is beyond the scope of this report. See the seminal paper by Rudolf E. Kalman (Kalman, 1960).

We can apply the Kalman filter to the problem of Robot Localization as follows. The time update step is triggered by the walking module when the robot makes a step. We can use the noisy odometry data to update the mean robot pose of the form (x, y, θ). Already we have violated the strict definition of a Kalman filter in that the motion of the robot is not a strictly linear change in state. The direction of the robot relates to the position of the robot through various non-linear trigonometric functions as shown in Fig. 2.
The measurement update is triggered when the vision system detects an object. Objects that can be detected include the four unique landmarks, or beacons, placed around the field, the ball, and the two goals. The vision system returns distance and angle estimates from the robot to the object detected. We could use the noisy distance and heading to the landmark to form an observed state estimate and then update the mean pose estimate. However, at this point the algorithm breaks down again. This is because the standard Kalman Filter assumes that the mapping between the state vector and any observation is linear, in the above case, represented by the matrix H. If we were able to observe directly, albeit with noise, the robot pose in (x, y, θ) form, then we would be fine. When observing a beacon, however, the information given implies that the robot pose can be anywhere on a helix in (x, y, θ) space. Knowing the distance to a landmark places you on a circle of a given radius around that landmark, and knowing the heading to it gives you a certain heading at every point on that circle, but they do not provide a single point. The helix is non-linear and cannot be represented by a Gaussian. In order to deal with this issue, we need to move to the Extended Kalman Filter, which can handle non-linear mappings between observations and state.

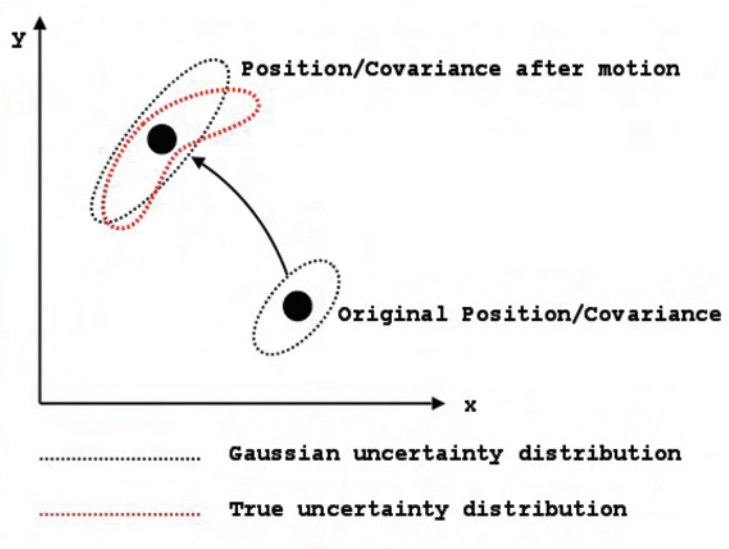


Fig. 2. Gaussian approximation to a non-linear motion update

5. Extended Kalman Filter

A Kalman Filter uses linear Gaussians over the state space to estimate the probability distribution function. However, as noted in the previous section, if a measurement or motion update has a non-linear nature, the classic Kalman Filter algorithm cannot handle this kind of situation. A solution to this problem is to linearise the function from state space to observation space around certain point such that we would now have a linear Gaussian approximation. We use the tangent line (or hyper-plane) which passes through the point x as the linear approximation. When applied to the Extended Kalman Filter, we must compute the multi-dimensional derivative of the non-linear function - the Jacobian Matrix. Take the function F which maps an n-dimensional state space onto an m-dimensional observation space:

$$F = \begin{pmatrix} f_1(x_1,\ldots,x_m) \\ \vdots \\ f_n(x_1,\ldots,x_m) \end{pmatrix} \tag{6}$$

The corresponding Jacobian Matrix would be:

$$J = \begin{pmatrix} \frac{\delta f_1}{\delta x_1} & \cdots & \frac{\delta f_1}{\delta x_m} \\ \vdots & \ddots & \vdots \\ \frac{\delta f_n}{\delta x_1} & \cdots & \frac{\delta f_n}{\delta x_m} \end{pmatrix} \tag{7}$$

Each of the elements of the Jacobian Matrix is a partial derivative of the non-linear function F. We can now use this linearisation to be able to incorporate non-linear observations and time updates into our Kalman Filter. The time update equations become:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k) \\ P_k &= AP_{k-1}A^T + Q \end{aligned} \tag{8}$$

In the above equation, f is the function which updates the mean state estimate position, it may be non-linear, and the matrix A is the Jacobian of this function. The measurement update equations become:

$$\begin{aligned} K_k &= P_k^- J^T (JP_k^- J^T + R)^{-1} \\ x_k &= x_k^- + K_k(z_k - Jx_k^-) \\ P_k &= (I - K_k J)P_k^- \end{aligned} \tag{9}$$

and we can now assume that the state to observation equation is no longer linear, becoming:

$$z_k = j(x_k) + u_k \tag{10}$$

If we consider beacon observations to be two dimensional observations, being heading and distance to the beacon, we can derive a Jacobian matrix which is a derivative of the mapping between robot pose state space and observation space. The mapping between state space and observation space is as follows:

$$\text{distance} = \sqrt{(x_{robot} - x_{beacon})^2 + (y_{robot} - y_{beacon})^2} \tag{11}$$

$$\text{heading} = \tan^{-1}\frac{(y_{beacon} - y_{robot})}{(x_{beacon} - x_{robot})} - \theta_{robot}$$

If we now compute the first derivative of this function from state space to observation space, we get the following Jacobian matrix:

$$J = \begin{pmatrix} -\dfrac{k*(y_{rbt} - y_{bcn})}{(x_{rbt} - x_{bcn})^2 + (y_{rbt} - y_{bcn})^2} & \dfrac{k*(x_{rbt} - x_{bcn})}{(x_{rbt} - x_{bcn})^2 + (y_{rbt} - y_{bcn})^2} & -1.0 \\ \dfrac{(x_{rbt} - x_{bcn})}{\sqrt{(x_{rbt} - x_{bcn})^2 + (y_{rbt} - y_{bcn})^2}} & \dfrac{(y_{rbt} - y_{bcn})}{\sqrt{(x_{rbt} - x_{bcn})^2 + (y_{rbt} - y_{bcn})^2}} & 0.0 \end{pmatrix} \tag{12}$$

A more thorough derivation and explanation of the Extended Kalman Filter can be found in other papers (Thrun et al., 2005) (Zarchan et al., 2005).

6. Multi-Modal Localization

The Extended Kalman-Bucy Filter is a powerful algorithm for robot localisation. However, one of its major downfalls is the fact that it can only approximate a uni-modal probability distribution function. So, for example, if the robot knew it was in one of two positions, say, it knew it was next to one of the two goals, the probability distribution function for the pose of the robot would have two local maxima, each centred near one of the goals. The Extended Kalman Filter, which uses a single Gaussian to represent the pose and uncertainty, would be unable to model this situation sufficiently well. This hypothetical situation would be much better modelled by the sum of two separate Gaussians, each of which is centred on one of the modes. In fact, it is possible to approximate any probability distribution function arbitrarily accurately using a weighted sum of an arbitrary number of Gaussians. To incorporate multiple modes into the localisation algorithm we use an array of uni-modal Gaussians, each with an associated weight. We can then view the full probability distribution over the world state as a weighted sum of Gaussians.

$$P(\underline{x}) = \sum_{i=0}^{N} \omega_i G_i(\underline{x}) \tag{13}$$

This weight ranges between 1.0 and 0.0 and is a measure of the probability that that particular Gaussian represents the state of the system. A Bayesian interpretation allows us to update the weights when an observation is made. In practice, the Gaussians which match the observation well have a higher weight than the Gaussians which disagree with the observation. Given an observation covariance R, Jacobian J, and the covariance C of the Gaussian prior, we can calculate the combined covariance E. Combining this with the innovation vector v we calculate the weight scalar S, which allows us to update the weight of the Gaussian distribution.

$$\begin{aligned} E &= R + J^T C J \\ S &= \exp(-\frac{1}{2} v^T E^{-1} v) \\ \omega_i &= S\omega_{i-1} \end{aligned} \tag{14}$$

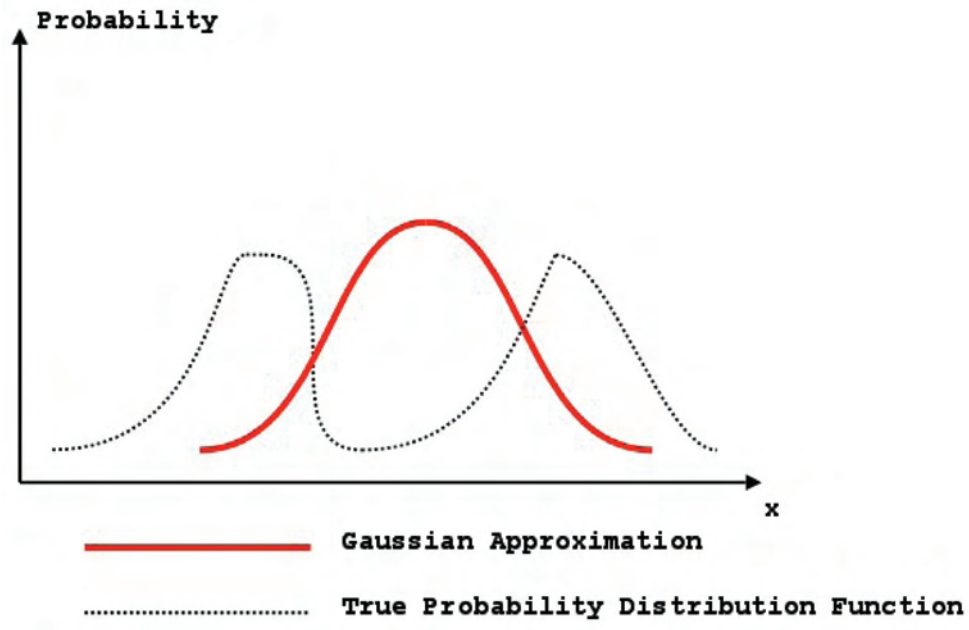


Fig. 3. A single Gaussian approximates a multi-modal probability distribution poorly

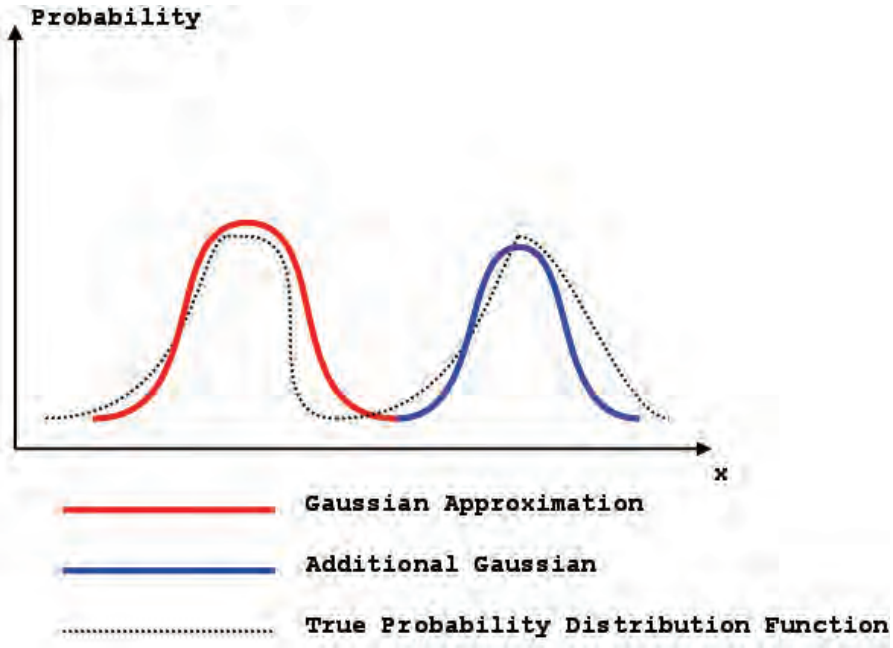


Fig. 4. A weighted sum of Gaussians provides a far closer approximation to the true distribution

The sum of the weights of all of the Gaussians in the distribution must sum to 1.0 in order to make the entire distribution a valid probability distribution (one which integrates to 1.0 from -1 to 1). In order to maintain this property a renormalisation must happen every time the weight of a Gaussian is modified. In addition to this, Gaussians which have a very low weight are removed from the distribution array. This is because they represent extremely unlikely modes, and for performance reasons we cannot keep track of unlikely modes. Another method used to reduce the number of Gaussians that form the distribution is merging similar Gaussians. If two Gaussians have a similar mean and similar covariance matrices, then one of them is removed and the other becomes the average of the two. To calculate the global maxima of the weighted sum of Gaussians distribution a simple approach is taken, the maxima is assumed to be the mean of the Gaussian of highest weight. This is not strictly correct, but is a good enough approximation, seeing as a correct solution for finding the global maximum of a sum of Gaussians is a lot more involved and does not provide enough of a benefit for it to be used in our system.

7. The State

An important part of building a tracking system is deciding which state to track. It is possible to track the pose of a single robot as a three dimensional system (x, y, θ). This is very effective, but it is possible to do better.
The current rUNSWift system tracks a 16 dimensional state, rather than a 3 dimensional one.

$$MeanVector = \begin{pmatrix} robotXpos \\ robotYpos \\ robot\theta \\ ballXpos \\ ballYpos \\ ball\delta a \\ ball\delta a \\ teammate_1Xpos \\ teammate_1Ypos \\ teammate_1\theta \\ \vdots \\ teammate_3Xpos \\ teammate_3Ypos \\ teammate_3\theta \end{pmatrix} \quad (15)$$

This state incorporates tracking the ball as well as tracking all four robots on the team. This enables information from all four robots to be combined when determining the state. The mechanism for distributing this information is discussed below.

In addition to this, we use more complex motion update than the simple "shift-and-blur" model mentioned above. A Kalman Filter can handle any linear transform on the state as a motion update. If the robot is standing still, this update would normally be an identity matrix for a single robot – the robot stays where it is. With the addition of a ball that moves when the robot is not moving, a non-identity transition matrix is required: the velocity of the ball is added to the location of the ball at each time step. This simple change induces a correlation between the location of the ball and its velocity in the state distribution. When we see the ball a second time, the new information about its position will also give information about its velocity.

$$MotionMatrix = \begin{pmatrix} I_{3,3} & & & & & \\ & 1.0 & 0.0 & 1.0 & 0.0 & \\ & 0.0 & 1.0 & 0.0 & 1.0 & \\ & 0.0 & 0.0 & friction & 0.0 & \\ & 0.0 & 0.0 & 0.0 & friction & \\ & & & & & I_{9,9} \end{pmatrix} \quad (16)$$

8. Noise Filtering

Using the power of a multi-modal filter we can improve the robustness of the system to spurious observations. In the 4-legged league it is very likely that the vision system will make false classifications of objects which are part of the background, such as spectators wearing coloured t-shirts. These may be classified as beacons, or goals, or the ball. This is a different type of noise in the system to what a standard filter is designed to handle, that is, noise that is centred on the mean. We need a way to be able to reject spurious observations, otherwise they will significantly reduce the accuracy of the localisation system, more so than standard "noisy" observations. Take as an example a seeing a spectator wearing an orange t-shirt in the crowd, which the vision system classifies as a very large (and thus close) ball. The variance of this observation will be small because the closer the ball is the more certain we are about its observed distance. This results in our estimate of the ball position shifting significantly towards the observed "phantom" ball position, despite the observation being false.

Our solution to this problem is to allow that the observation may or may not be correct, and as such when we apply the observation to every Gaussian in the weighted sum distribution we also make a copy of the Gaussians which do not have the observation applied, but we scale the associated weights down by a constant factor. This constant factor can be seen as the probability of a false observation. This means that for every observation, the system doubles the number of Gaussians which make up the distribution. These are later culled if there are too many or their weights are too small. An observation is said to be a phantom observation if the weight of the Gaussian with it applied is lower than the weight of the Gaussian without the observation applied.

This technique works because the more an observation disagrees with the current state, the lower the weight will be of the resulting Gaussian after applying the observation. So if an observation is made which is extremely unlikely, and so is probably a false one, the resulting Gaussian will have a lower weight than the Gaussian without the observation applied.

The effectiveness of this approach was demonstrated to us when we accidentally swapped two of the beacons around and asked the robot to position itself at a kick-off position. This results in 4 valid landmarks (2 goals and 2 beacons), and 2 invalid landmarks (the 2 swapped beacons). Despite expecting the robot to localise poorly due to the contradictory observations, the robot localised extremely well, rejecting almost all observations which were of the switched beacons. This robustness to false observations was extremely important at the competition due to the fact that there were spectators close to the field at "eye level" who were wearing coloured shirts. Without this noise filtering in the localisation system, the performance of the system would have been far lower. It is important to note that unlike some previous work (Browning et al., 2002), out system does not reject "bad" observations, but rather tracks all possibilities as different Gaussians, rejecting them only when they become extremely unlikely.

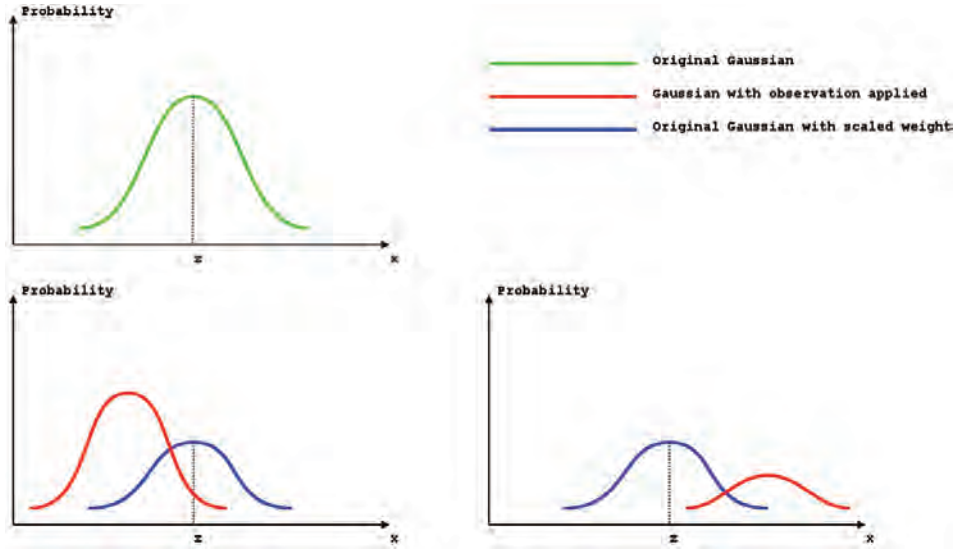


Fig. 5. shows two possible scenarios. One is where the original Gaussian (top left) is split into two, the resulting distribution is such that the Gaussian with an observation applied has a higher weight than the scaled down Gaussian with no observation applied (bottom left). The other is that the observation is spurious and thus the Gaussian with the observation applied has a lower weight than the scaled down Gaussian (bottom right)

9. Multiple Linearisation Points

One of the sources of error in an Extended Kalman Filter comes from the fact that we are approximating a possibly non-linear probability distribution with a linear Gaussian function. This is one of the advantages of a particle filter approach to localisation, particle filters do not have to linearise a non-linear distribution, since they can approximate any probability distribution function. However, using a sum of multiple weighted Gaussians to represent our function, we can reduce the error from the linearisation process by better approximating non-linear function. In effect we have a simple, and efficient, unscented Kalman Filter or Rao-Blackwellised particle filter. Figure 6 is a representation of the linearisation process for the standard Extended Kalman Filter approach. The true probability distribution function is not a linear Gaussian one, so in order to approximate it, a linear Gaussian distribution is used which is tangential to the true probability distribution at the current mean point.

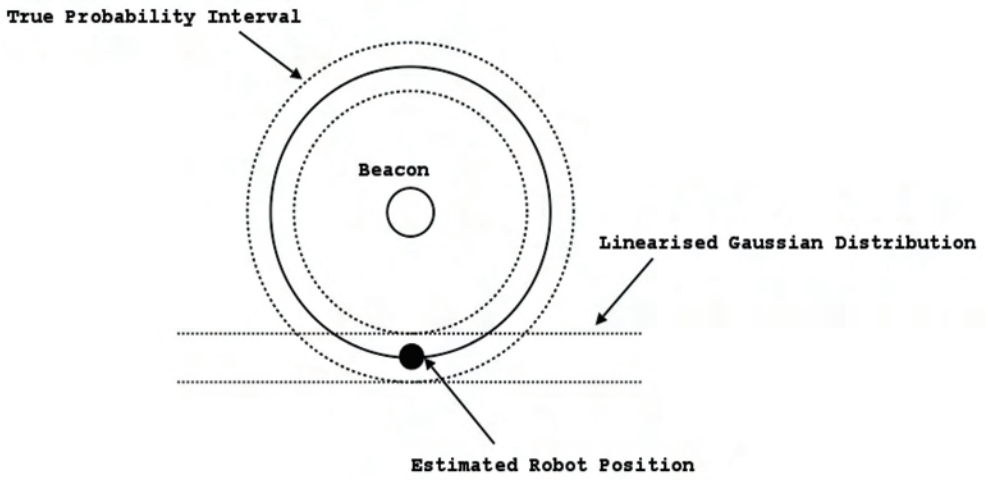


Fig. 6. The standard Extended Kalman Filter linearises around only one point, the estimated mean of the world state

The way in which we do this is every time a single beacon (note that for the purposes on this section, the ball should be considered as being a beacon) is observed, a copy is made of every Gaussian in the distribution, with the mean of each copy being offset such that it is the same distance from the observed beacon, but is rotated around by a given angle. The weights of these new displaced Gaussians are also scaled down. It is important to note that the displaced Gaussians can only be generated when only one landmark is observed. This is because it is not consistent with the observations to rotate a Gaussian around a beacon if there are multiple observed beacons. Every time there is a single beacon observation, only one displaced copy is made per existing Gaussian, whereas we would need 2 to maintain symmetry. This is done with the aim of improving the speed of the localisation module, spawning two additional Gaussians would have been too expensive, so instead we alternate whether to rotate the added displaced Gaussian clockwise or anti-clockwise around the beacon. In our implementation we chose to rotate the displaced Gaussian by 16 degrees, and the weights are multiplied by 0.1.

The end result of this is a better approximation of the probability distribution function through a reduction in the inherent error introduced by the linearisation process. Figure 7 shows how the two additional linearisation points are placed in relation to the mean, and how the multiple linear Gaussians are a closer approximation to the true probability distribution function.

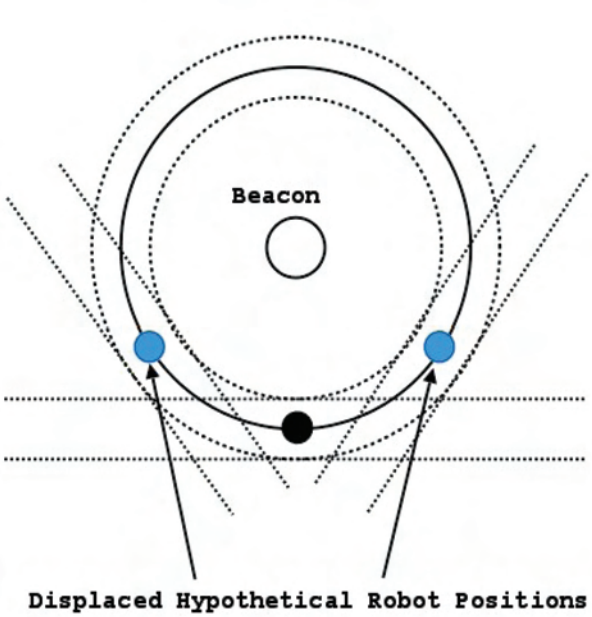


Fig. 7. Our system adds low weight Gaussians rotated around the beacon, meaning we linearise around multiple points

10. Incorporating Teammate Observations

The previous sections have all described ways in which the increased representational power of a sum of Gaussians representation allows us to more accurately model a probability distribution, and hence more accurately localise. In this section we discuss the distribution of information between robots on a team rather than the representation of information on a single robot.

This incorporation of distributed observations of the world state into Kalman Filter localisation is one of the most important improvements of our system over a standard extended Kalman Filter. Our technique involves each robot keeping track of a separate traditional Kalman Filter, which we update as per normal, but does not form part of the main weighted sum of Gaussians probability distribution function. We refer to the data stored in this Kalman Filter as the "shared Gaussian". Periodically, this is sent to all team-mates. After being sent, the shared Gaussian is reset to a 'uniform' state, with high variance and a mean equal to our best estimation of our pose. This avoids any observations being incorporated into a team-mate's state estimate multiple times. In addition to sending the Shared Gaussian mean vector and covariance matrix, we also send the cumulative odometry information, which we also reset every time a wireless packet is broadcast.

When a wireless packet is received, we can incorporate the team-mate observation into the main probability distribution as a direct observation. There is no need to linearise the function mapping observation to state space, since it is already a linear one - all observations have already been linearised by the sending robot. The data sent is of the form of a 7

dimensional mean vector and a covariance matrix. This reduced form is used to save communications bandwidth. The receiving robot must have a matrix which maps the team-mate pose and ball position/velocity estimates into its own world estimate.

$$A = \begin{pmatrix} 0_{7,3} \\ 0 \\ I_{3,3} \\ 0 \end{pmatrix} \tag{17}$$

$$B = \begin{pmatrix} 0_{3,4} \\ I_{4,4} \\ 0_{9,4} \end{pmatrix}$$

$$H = (A \mid B)$$

The matrix H is the mapping between the wireless team-mate observation and our world state estimate. The matrix B is constructed such that the placement of the 3x3 identity matrix corresponds to the index of the robot id from which the packet was received, such that our idea of where that team-mate robot is positioned is updated accordingly inside the mean vector.

The inclusion of team-mate observations greatly increases the accuracy of ball position and velocity tracking. With this change it is possible for a robot to grab a ball under its chin with its own local distance observations turned off, and relying on team-mate robots on the field to obtain the distance to the ball.

In addition, the accuracy of the robot pose itself is greatly improved by this communication. Our scheme allows for the ball to act as a moving landmark from which the robots can localise. For example, if a very well localised robot is looking at the ball, it can transmit a very accurate and certain position of the ball to its team-mates. Following this, a poorly localised team-mate robot can look at the ball, and knowing the ball's position, gain information about its own location.

Before introducing this information sharing, our robots required an active localisation behaviour. If a robot is chasing the ball for a prolonged period of time it will have seen few, if any, landmarks and would previously become severely mis-localised. The active localisation behaviour involves a robot looking away from the ball to glance at a landmark to re-localize itself. The downside of active localisation is that while the robot is looking away from the ball, there is a chance the ball could be moved by an opponent and our robot will lose track of it. With information sharing in place, the ball itself helps the robot localise, allowing it to focus more on the ball and less on looking around for beacons.

11. Results

We evaluated the effectiveness of our system by setting up various configurations of landmarks, balls and team-mates, and then by running a robot between a set of waypoints

in order. The robot would stop when it considered itself to be close to a waypoint, and then the distance in centimetres between the robots position and the true waypoint position was recorded. We performed this test with several different field layouts and between several different version of the localisation module. The field layouts for the various tests are shown in Figure 8. The results are presented in tabulated form, listing the average measured distance between the robot position and the waypoint position for every waypoint. The total average distance error is also recorded.

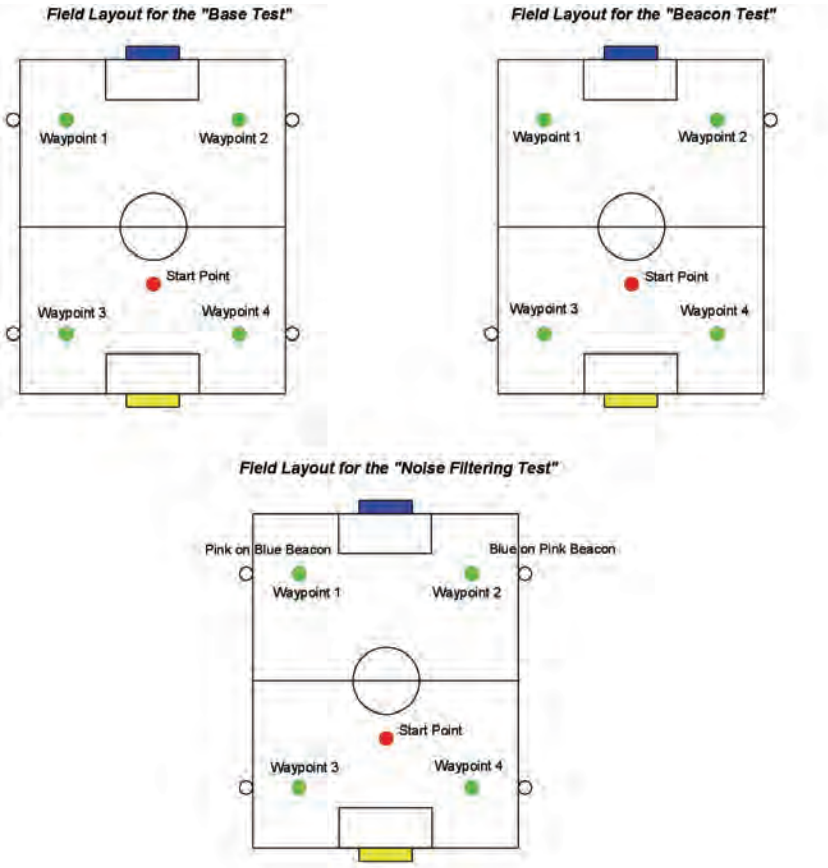


Fig. 8. This figure shows the respective field layouts for the tests that were run in order to evaluate the performance of the system

The first test which we ran is a base test. This involves 4 waypoints, each at a quadrant of the field, and all of the beacons and goals. The robot starts near the centre circle and runs between the waypoints in increasing numbered order. We use this test to compare the accuracy of the previous localisation system, and the new localisation system which has all of the enhancements mentioned in this report.

Full 2005 Localisation System		
	Mean Error (cm)	Standard Deviation
Waypoint 1	32.0	8.86
Waypoint 2	18.8	5.15
Waypoint 3	31.4	23.6
Waypoint 4	62.8	64.74
Average	28.0	25.5

Full 2006 Localisation System		
	Mean Error (cm)	Standard Deviation
Waypoint 1	37.2	9.32
Waypoint 2	14.0	5.3
Waypoint 3	19.0	11.2
Waypoint 4	17.0	7.7
Average	21.8	8.3

11.1 Beacon Test

This test is very similar to the Base Test, except for the removal of 2 beacons. This change makes it much harder for the robot to localise as it receives far fewer observation updates. This shows a larger disparity between the old and new localisation systems.

Multiple Linearisations Disabled		
	Mean Error (cm)	Standard Deviation
Waypoint 1	58	30.5
Waypoint 2	34.2	12.1
Waypoint 3	20.8	10.3
Waypoint 4	28.4	8
Average	35.35	15.22

Multiple Linearisations Enabled		
	Mean Error (cm)	Standard Deviation
Waypoint 1	27.0	12.4
Waypoint 2	13.0	7.0
Waypoint 3	37.0	5.1
Waypoint 4	15.6	2.3
Average	23.15	6.7

11.2 Noise Filtering Test

Our system is multi-modal in nature, allowing it to consider observations as possible false and possibly true. This results in spurious observations having a far lower effect on the accuracy of the system. If the robot observes a landmark which does not make sense for the current estimated world state then it is possible for the system to deal with this by using the Gaussian without the observation applied as the mean Gaussian instead. The setup for this

test is similar to that of the Base Test, except that we swap around the Yellow on Pink and Pink on Blue beacons. This result in any observation of either of these beacons as being "false", and hopefully the localisation system will cull them.

Noise Filtering Disabled		
	Mean Error (cm)	Standard Deviation
Waypoint 1	137.0	31.0
Waypoint 2	191.0	153.0
Waypoint 3	189.0	8.2
Waypoint 4	123	13.5
Average	160.0	51.42

Noise Filtering Enabled		
	Mean Error (cm)	Standard Deviation
Waypoint 1	31.0	8.6
Waypoint 2	27.3	9.8
Waypoint 3	34.0	8.6
Waypoint 4	12.6	1.0
Average	26.22	7.0

We used a one sided Mann-Whitney U test to test the significance of these results. The Beacon and noise filter tests were significant (the beacon test at $p = 0.05$ and the noise filter test at $p = 0.01$). The base test is not significant with this small sample, but the changes do not degrade performance.

In the end, the best test is the performance of the entire system. Our experience is that each of the changes presented in this chapter lead to small, but important, improvements in the level of play of the team as a whole.

Our current experiments demonstrate a small but not statistically significant improvement in accuracy due to the ball tracking. Our more subjective tests with the whole team in a game suggest that this is important for complete game behaviour.

12. Conclusion

The above results show that the documented improvements have had a great effect on the overall accuracy of the localisation system. The effectiveness of the Noise Filtering part of the system is staggering. It should be also noted that these experiments only measured the (x, y) pose error, whereas the localisation system tracks much more data than those 2 dimensions, including the ball position and velocity. All of the mentioned improvements to the localisation system allowed us to great flexibility and power in terms of higher level behaviours. This is shown by the fact that our team came 2nd in the World Open in 2006, and in the same year were the Australian Champions.

National ICT Australia (NICTA) is funded by the Australian Government's Department of Communications, Information Technology, and the Arts (DICTA) and the Australian Research Council through Backing Australia's Ability and the ICT Research Center of Excellence programs.

13. References

Browning, B; Bowling, M & Veloso, M. (2002). Improbability Filtering for Rejecting False Positives. *Robotics and Automation,* Vol. 3, (November 2002) 3038-3043, 0-780-37272-7

Fox, D; Burgard, W & Dellaert, F. (1999). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. *Proceedings of AAAI,* pp. 343-349, 0-262-51106-1, Orlando Florida, July 1999, MIT Press, Cambridge

Kalman, R. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering,* Vol. 82, (1960) 35-45

Thrun, S; Burgard, W & Fox, D. (2005). *Probabilistic Robotics*, MIT Press, 0-262-20162-3, Cambridge

Zarchan, P & Musoff, H. (2005). *Fundamentals of Kalman Filtering: A Practical Approach SE,* AIAA, 1-56347-694-0, Reston, VA.

25

Impossibles: A Fully Autonomous Four-legged Robot Soccer Team

Hamid Reza Vaezi Joze, Jafar Habibi and Nima Asadi
Sharif University of Technology
Iran

1. Introduction

The goal of the international RoboCup soccer initiative is to develop a team of humanoid robots that is able win against the official human World Soccer Champion team until 2050. Currently, there exist a number of different RoboCup soccer leagues that focus on different aspects of this challenge. The Four-Legged League is one of them. In the league teams consisting of four Sony Aibo robots each play on a field of 6 m x 4 m. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers.
In this chapter we are going to present the Impossibles main architecture and its modules to create a fully autonomous team of 4-legged robots (Sony Aibo) for playing soccer according to RoboCup 4-legged Soccer League's rule. This architecture includes different modules such as World Model, Vision, Decision Making, Motion Controller, Communication, and Localization. This chapter presents the integration of our researches in different fields which came together to create fully autonomous robots for specific purpose that is playing soccer as humans do. And this could be a primitive attempt to develop intelligent robots.
In the first section we briefly describe the specifications of Aibo Robots, the history and rules of RoboCup 4-legged Soccer. In Section 2 we discuss previous works which consists of our team's architecture. Fig. 1 demonstrates our team (*Impossibles)* architecture. In next sections each module will be discussed separately. It includes the tasks of the modules, the corresponding task in other teams and the scientific methods which were used before or those that are presented by us.
The Vision module consists of chromatic distortion, color classification, line and object detection which is mainly concerned on colored image processing. Our Localization module applies piecewise linear probabilistic localization method which is based on Markov localization. In the case of distributed agents (against centralized agents) these raw data should be shared between all teammates via wireless communication. Decision making module is responsible for high-level decision and it consists of soccer strategies besides fuzzy rules. Motion skills, parameters of walking and movement estimation are the main parts of Motion module. In the Tools & Debugging Section we concentrate on debugging tools which simplify the process of debugging on the robots and also an interpreter which is used for facilitation of determining team strategies and player's roles.

1.1 Why Four-Legged Robot Soccer?

As an internationally recognized team game, soccer is a perfect standard project for studying how multi-robots perform and react autonomously in uncertain, team-oriented scenarios. The sport also provides some entertainment value adding good spirits and public interest to a concrete test bed event for researchers. These reasons could be the motivation of existence of RoboCup Soccer Competitions since 1996.
After years of competition in soccer with the Small Size and Middle Size robots, which moves with wheels, besides Simulation Leagues, 4-Legged Soccer League was added to RoboCup. The ambition of 4-Legged soccer league is to simulate the way human beings play soccer, with the use of legs. One of the other advantages of 4-Legged robots is that the platform is common among all the teams and this establishes a fair test-bed for Artificial Intelligence achievements.

1.2 Aibo Robot Specifications

AIBO robots' first generation was developed by SONY Corporation in 1999 for entertainment purposes besides its use in research laboratories. The Sony Aibo robot is currently a very interesting platform to conduct research in Robotics and Artificial Intelligence. Aside the numerous captors and actuators, the most important element is that Aibo is programmable. The Aibo programming language, built on top of C++, is provided by Sony as the OPEN-R SDK. The latest product of Sony Aibos is its third generation Aibo robot, ERS-7, with a great tool for wireless communication.
These robots have 20 degrees of freedom and are equipped with a 576 MHz processor and 16 MB of RAM. The most important sensor of this robot is a CMOS camera with 56.9° horizontal and 45.2° vertical vision angles.

2. Impossibles Architecture

Our previous experience in Multi-Agent System (MAS) architecture design in Simulation league environment led us to World Model Based Architecture (WMBA). We employ it as our basic designed architecture for concurrently-running objects of Open-R SDK. WMBA contains four major tasks which are done independently in following subsystems:

1. Sensing Subsystem
2. Communication Subsystem
3. Action Subsystem
4. Debugging Subsystem

These subsystems are to run repeatedly with different frequencies. They are managed in such a way that objectives are achieved and constraints are convinced. The main constraint of the AIBO robots is the limited resources such as CPU. The last subsystem which is new to previous architecture is in charge of gathering appropriate information from other subsystems and sending them to Aibo Controller software which will be described later.
Fig. 1 demonstrates the World Model based architecture. Subsystems are denoted by dotted rectangles, data flow is shown as arrows, and processes are shown by circles.
To make Decision Making module completely separate from other parts, we use a non Open-R abstract class called *AbstractPlayer*. There are two pure virtual functions in this

abstract class. The first one is *sense* which is called every time robot gets some new information. The second virtual function is *decisionMaking* which is called in a specific period of time to decide about new actions. The result of this function could be any of the actions such as walking, shooting, looking, etc. Some actions such as standing back to normal position in the case the robot falls down and also Blocking Skills (which will be discussed later in detail) are done completely without interrupts or preemptions, so *decisionMaking* is not called during these skills.

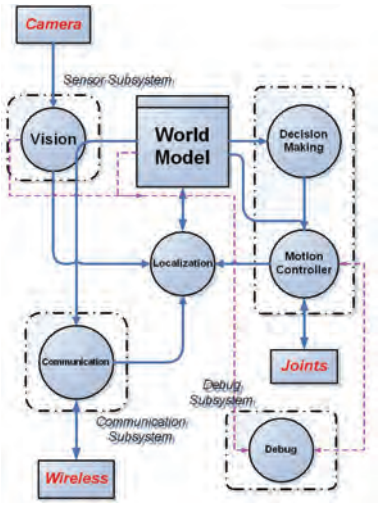


Fig. 1. Impossibles World Model base Architecture

3. Vision

Undoubtedly, vision is the most powerful sense of human being providing a great deal of information for interaction with environment without any physical contact. In this section, we concentrate on providing a method for real-time vision in a robot with low computational power and limited memory. Real-time vision means processing image frames with the speed of robot's camera. The most important sensor of this Aibo robot is a CMOS camera with 56.9° horizontal and 45.2° vertical vision angle.

Vision problem for these robots which refer to recognize type and location of surrounding objects is described as follows:

- Input: (i) The output of robot camera in the form of 30 pictures per second, with the size f 208x160 pixels and in the YCrCb color space consisting color distortion and noise. (ii) The value of robot's joints through which the value of camera's pan, tilt and roll can be obtained.
- Output: (i) Robot's distances and angles in relation to the ground's fix location by which the robot estimates its position in the field. (ii) Robot's distances and angles in relation to moving objects which determine their position relative to the robot.

The first work carried out on the image received from robot's camera is the correction of distortion existed in the color of image pixel far from the center of image. Then the color of

each pixel in the corrected image is attributed to one of the predefined color class. Then, existing blobs of each color are formed for the color of existing objects in the image, and based on the color, size and density of existing objects in the image is recognized. Also, by using a method provided for obtaining the three dimensional coordinates of each pixel in the image in the outside space relative to itself, position of each object relative to the robot is calculated. Specially, ground's lines are among of important objects in the soccer field which are distinguished in a separate manner with seeking green-white edges. Finally, having determined the position of objects relative to the robot, the position of robot in the field is calculated through the objects having a fixed place in the environment.

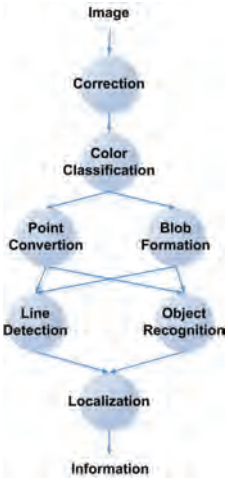


Fig. 2. Architecture of real-time vision system of robot

Fig. 2 displays the architecture of robot's real-time vision system (Mokhtarian et al., 2007). Also this system require offline processing regulating some provided algorithms' parameters for various setting prior to the application of software on the robot.

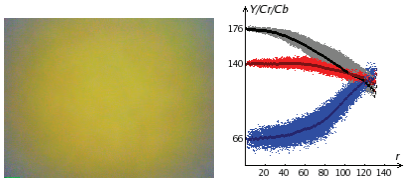
3.1 Correction of Chromatic Distortion

Aibo robot's camera causes a considerable chromatic distortion in the color of pixels at the corners of images. Fig. 3 (a) displays an image taken by such camera from a uniformly colored yellow page. Variations of the three color components (*Y*, *Cr*, and *Cb*) of this image's pixels are shown in Fig. 3 (b) in terms of pixels' distance from image's center (*r*).

The thick and fading curves of Fig. 3(a) illustrate variations of the observed values of the *Cr* component (channel) of pixels, in terms of *r*, in a number of images taken from uniformly colored pages. We define the *actual value* of each component for these colors, as that component's value in the pixels around the center of the corresponding image, where there is negligible distortion.

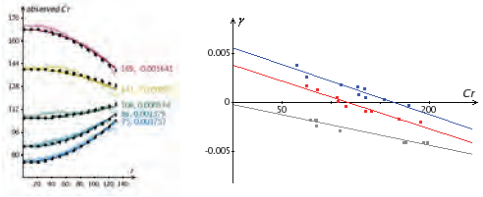
By shifting the horizontal axis to the width of *actual value* of the *Cr* component, for each curve (e.g. 141 for yellow) in Fig. 4(b), the diagram of $Cr_{observed} - Cr_{actual}$ is obtained for each color, which we approximate by a curve of second degree in the form of $\gamma \times r^2$. These

curves are shown in Fig. 4(a) by thin continuous curves. Beside each curve, the *actual value* of the *Cr* component and the corresponding γ value are displayed as well. The coefficient γ for each curve depends on the *actual value* related to the curve, as shown in Fig. 4 (b).



(a) (b)

Fig. 3. (a) Image taken from a uniformly colored yellow page. (b) Values of color components of Figure 2-a's pixels in terms of distance from the center (*Y*, *Cr*, and *Cb* are shown by gray, red, and blue colors respectively)



(a) (b)

Fig. 4. (a) Variations of the component Cr in terms of r in a number of uniformly colored pages. (b) γ values in terms of the actual value of Cr

We approximate the values of γ by a linear equation in terms of Cr_{actual} in the form of $\gamma' = \alpha \times Cr_{actual} + \beta$. Coefficients α and β for each color component, are parameters independent of images' colors and are determined for the camera with specified parameters (shutter speed, white balance, and gain).

Curves approximating $Cr_{observed} - Cr_{actual}$ using the new coefficient (γ' which is on turn obtained by a linear approximation from Cr_{actual}) are displayed in Fig. 4(a) as thick dotted curves. The maximum error of this two-level approximation in our experiments on uniformly colored images –Fig. 4(a) depicts variations of the *Cr* component of a number of them– has been acquired as an inaccuracy of at most 10 units for a component, which seems appropriate with respect to the interval of components' values (0..255). After determining coefficients α and β for the component *Cr*, Equation (1) holds for each pixel of an image.

$$Cr_{observed} - Cr_{actual} = (\alpha \times Cr_{actual} + \beta) \times r^2 \Rightarrow Cr_{actual} = \frac{Cr_{observed} - \beta \times r^2}{\alpha \times r^2 + 1} \quad (1)$$

Therefore, having $Cr_{observed}$ for each pixel, pixel's distance from the center of the image, and coefficients α and β, the actual value of the *Cr* component of each pixel's color is obtained using Equation (1). Similarly, the actual values of two other components of pixels' colors are

obtained and thus the chromatic distortion of the image is corrected. Fig. 5(a) and (b) depict two sample images captured by robot's camera and Fig. 5(c) and (d) depict the result of their correction. For speeding up the process of chromatic distortion correction of images in real-time visioning, we use pre-calculated tables for each process.

(a) (b) (c) (d)

Fig. 5. Two samples images taken by the robot's camera, before ((a) and (b)) and after ((c) and (d)) correction of chromatic distortion

3.2 Color Classification

In order to recognize objects existing in an image, firstly it should be segmented into color regions with specified colors and the rest of processes are carried out on the color-classified image. We use a three-dimensional $(128 \times 128 \times 128)$ color classification lookup table mapping points of the YCrCb color space to corresponding color classes. The number of colors that should be distinguished from one another (n) is 8. Moreover, we consider an additional class named *unknown* for colors similar to more than one of our specified classes.

In order to construct this table, we take a large number of images from the environment and objects with which the robot is dealing, and after correction of chromatic distortion, we specify relevant color segments in each image for the learning tool.

We conduct color learning and color classification based on the HSL color space since it resulted in best outcomes in our experiments regarding colors existing in our environment and its lightning conditions. Therefore, having collected the samples of each color class from captured images, the averages of *H*, *S*, and *L* components of each color class (C_i) is designated the standard point (h_i, s_i, l_i) for that color class in the HSL space.

In order to determine the class of each cell of the three-dimensional color classification table, first we obtain its corresponding point in the HSL space, and then calculate its similarity to each color class using a heuristic function in the form of $h : \{c_i \mid 1 \le i \le n\} \longrightarrow R$. The value of this function for a (h,s,l) point is calculated using Equation (2).

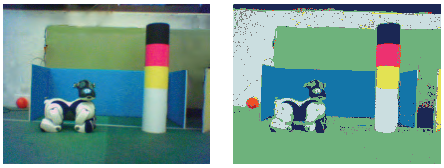
$$h(c_i) = \frac{w_i}{d((h,s,l),(h_i,s_i,l_i))} \tag{2}$$

In Equation (2), function d stands for the Euclidean distance between two points in the three-dimensional representation of the HSL color space, and w_i is the weight of each color class which somehow indicates its dispersion in the color space. Therefore, the standard deviation of positions of each class's sample points in the HSL space can be thought as an appropriate statistical criterion for the weight of that class. In addition, this criterion can be used as an initial state for obtaining the optimum set of weights which results in the best

outcome (i.e. minimum difference between automatically and manually color-classified images) using manual tuning tools or intelligent searching algorithms.
Having determined point in the HSL space corresponding to each cell of our lookup table, and calculated its similarity to each of our color classes, the most similar class is designated the color class of that cell. In addition, those points of the HSL space which are almost equally similar to more than one color class (i.e. the difference of their similarity to the most and the second most similar color classes is lower than a certain value) are placed in the *unknown* color class. Fig. 6 illustrates the color classification table obtained for our Aibo laboratory environment. This figure is a cube of side 256 from whose front a cube of side 192 is taken out. Gray areas of this figure stand for the unknown color class.

Fig. 6. Color classes in the three-dimensional space HSL

A sample of color classification (after correction of chromatic distortion) on the image shown in Fig. 7(a) is depicted in Fig. 7(b).



(a) (b)

Fig. 7. (a) Image taken by the robot's camera. (b) The result of color classification of (a)

3.3 Transforming a pixel in an image into a point in the space

A pixel in an image can simply be represented by (m,n), its coordinates in the image, where the coordinates axes of image are chosen as Fig. 8(a) for facilitating the calculations. On the other hand, a point in the space can be represented by (x,y,z), where the coordinates axes of the actual space are relative to the robot as shown in Fig. 8(a) (the floor is the plane $z=0$).

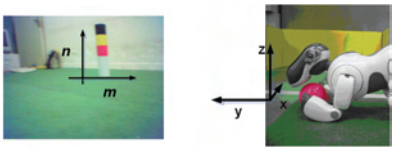


Fig. 8. coordinates axes in the image and in the actual space

Required parameters of the problem are:

- α : Camera's tilt (Angle made by camera and the horizon).

- β : Camera's pan (The angle of camera's deviation to left or right).
- γ : The angle between the oblique image and its horizontal representation.
- h_c : Height of the camera from the floor.
- ψ_{hor} , ψ_{ver} : Camera's horizontal and vertical angle of view.
- $width_{image}$, $height_{image}$: Image's width and height.

These parameters for Aibo robots are functions of the positions of rear and front legs and the three neck angles.

As a case in point, at each distance from the robot's camera, distances between objects in an image are assumed constant, i.e. if we know two objects are at distance d from the camera and at distance l pixels from one another in the image, then the actual distance between them in the space is regardless of whether they are seen at the center or at the corner of the image. If the given pixel has a value of m in the horizontal axis, then the actual point is located on a plane in the space crossing the point $(0,0,0)$, thus Equation (3) holds.

$$x / y = 2m / width_{image} \times \tan(\psi_{hor} / 2) \tag{3}$$

$$width_{image} \times x + 2m \times \tan(\psi_{hor} / 2) \times y = 0 \tag{4}$$

Therefore, the point is located on the plane represented by Equation (4). Similarly, if the pixel has a value of n in the vertical axis, then this point is located on a plane in the actual space represented by Equation (5).

$$height_{image} \times z + 2n \times \tan(\psi_{ver} / 2) \times y = 0 \tag{5}$$

The resulted line of crossing these two planes is a locus whose all points are seen at the pixel (m,n) in the image. Since the robot mainly interacts with objects located on the floor, we can posit the relevant point on the floor, thus the desired point is obtained by crossing the line mentioned above and the floor plane. After shift and rotation of coordinates axes relative to camera to axes relative to the robot itself using α , β , and γ , the results can be calculated using Equations (6) and (7).

$$\theta_x = a\tan(\tan(\psi_{hor} / 2) \times \frac{m}{width_{image}}) \qquad \theta_y = a\tan(\tan(\psi_{ver} / 2) \times \frac{n}{height_{image}}) + \alpha$$

$$x = -\frac{\cos(\theta_x)\sin(\theta_y)\sin(\gamma) + \cos(\theta_y)\sin(\theta_x)\cos(\gamma)}{\cos(\theta_y)\cos(\theta_x)\sin(\gamma) - \cos(\theta_x)\cos(\theta_y)\cos(\gamma)} \times h_c \tag{6}$$

$$y = \frac{\cos(\theta_y)\sin(\gamma) \times x + \cos(\theta_y)\cos(\gamma)}{\sin(\theta_y)} \times h_c \tag{7}$$

The above calculations have been performed assuming that $\beta = 0$, otherwise, the actual point $(x, y, 0)$ should be rotated by an amount of β around the origin.

3.4 Object Recognition

Object recognition in robot's vision consists of detecting objects existed in an image, assigning actual objects of the environment to them, and locating the recognized object regarding the coordinates axes relative to the robot.

The robot's working environment can be assumed closed, i.e. there is a set of specified objects to which no new one will be added. However, it is in general possible for Aibo footballer robots to see unspecified objects, i.e. other than known objects of the robot's environment (ball, goals, players, and landmarks). In this subsection, recognition of known objects based on blob formation is described first, and then recognition of unspecified objects.

A) Blob Formation

In our robot's real-time visioning sub-system, specified objects are recognized based on their color, size, and density and position of their corresponding blob in the image. Therefore, the next task after color-classification of an image is to form blobs existing in it.

In order to form such blobs in a color-classified image, connected pieces of relevant colors are obtained by a scan on the image. The density of a blob is equal to the number of points of the connected piece divided by the surface of the rectangle circumscribing that blob. Obviously, small blobs and those having low densities are ignored as noise. Fig. 9 shows a color-classified image and its relevant blobs.

Fig. 9. A color-classified image and its relevant blobs

B) Specific Object Recognition

The noteworthy characteristic of specified objects is that their shape and size are known for us. Therefore, types of these objects can be recognized knowing positions of relevant blobs, and their locations can be determined by geometrical calculations depending on their shape. Ball recognition is presented here as a sample of specified objects recognition.

In order to recognize the ball, which is an orange sphere, the relevant orange blob is considered the blob candidate to be the ball. Two parameters, circle's radius R, and coordinates of the circle's center (m_c, n_c) in the image, should be extracted from this blob. They can be calculated by averaging the center and the radius obtained for each three arbitrary border pixels of the observed ball. Regarding the parameters and coordinates, the ball's actual (x, y) relative to the robot can be calculated using Equations (9) and (8).

$$y = (\frac{33}{2R} \times 50 + 8.14) \times \cos(\beta) \tag{8}$$

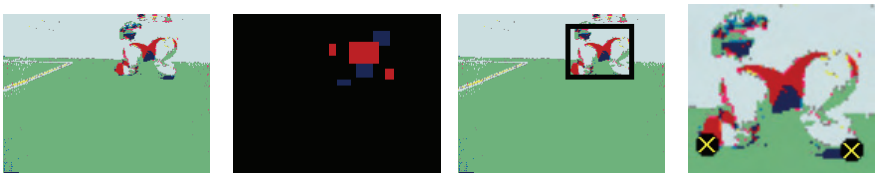
$$x = -R_{Ball} sign(\gamma) \times (n_c - \frac{m_c}{\tan(\gamma)}) \Big/ R\sqrt{1 + \frac{1}{\tan(\gamma)^2}} \quad (9)$$

White lines in the green fields are benefical information especially for determining the position of robot. For more information about our Line Detection methods refer to (Vaezi et al., 2007).

C) Recognition of Unspecific Objects

Unspecified objects' position can not be determined using their geometrical properties (i.e. shape). We consider unspecified objects just as some obstacles. Since an Aibo robot has a complicated shape whose recognition is not practical in our robot's real-time vision, the players in the field are viewed as unspecified objects by our robots.

Our algorithm is that existing blobs in the image which do not constitute any specified object and are located on the floor, are considered unspecified (unknown) objects. The assumption that they are placed on the floor allows us to locate their points on the floor using their blobs' lowest pixels and the method presented in Section 3.3 for transforming image's pixels into points in the actual space. At this location, there is merely an obstacle, and nothing about this object is determined but this obstacle's front edge placed on the ground. Stages of conducting this procedure for an unspecified object are shown in Fig. 10.



(a) (b) (c) (d)

Fig. 10. Stages of recognition of an unspecified object

The color-classified image and its relevant blobs are depicted in Fig. 10(a) and (b) respectively. Having checked these blobs and understood that they do not belong to any known object, the composition of these blobs, which is depicted in Fig. 10(c), is recognized as an unspecified object. Two points located at the bottom of this object (bottom-right and bottom-left corners of the object) leading us to calculation of this object's whereabouts are shown in Fig. 10(d).

3.5 Experimental Result

Since the presented methods and algorithms are to be used for Aibo robots' real-time visioning, the running time and the accuracy of them are two main parameters for assessment of these methods' appropriateness. Table 1 displays the running time of our visioning stages (accuracies are noted in relevant subsections). According to methods described above, the time required for correction of chromatic distortion and color classification is constant for all images. The time needed to form colored blobs, which is indicated in the third row of the table, is calculated for an almost crowded image including all colored objects existing in the environment of Aibos football field. Objects recognition

consists of a few calculations and its running time is negligible in comparison with other visioning algorithms.

Table 1. Running time of different process in Impossibles Vision subsystem

Visioning Stage	Running Time
Correction of chromatic distortion	4.8 *ms*
Color classification	4.1 *ms*
Blob formation	3.9 *ms*
Object recognition	Less than 1 *ms*
Line detection	3.8 *ms*
Total	16.7 *ms*

The total amount of time consumed by the visioning process is about 17 *ms* per image, thus almost the half of the time interval between two frames is left available for the rest of processes (e.g. decision making, communication, etc.).

4. Communication

Sony AIBO ERS-7 model have a wireless LAN module (wi-fi certificated) which made it able to communicate with other teammates to share useful information perceived from the environment to improve the quality of each agent's world model eventually resulting in more accurate localization and object detection.
Furthermore, Due to RoboCup 4-legged rules, each team has an upper band limit of 500 Kbps for communication among the agents which includes also game manager commands but every team has some special UDP port to broadcast. So ideally we can count on about 100 Kbps bandwidth for each AIBO robot.
In what follows we will discuss various aspects of communication and will explain the way that our communication module is working.
Impossibles AIBO communication module, as an independent module, works in parallel with other modules such as vision, and decision making. It is also in charge of sharing essential data amongst all players. For instance, knowing accurate positions of the players are only possible by having each player report his information such as its own position to the others.
The communication module is to be reliable and eventually be aware of the packet loss if any exists. It will repeatedly choose entities from World Model (WM) objects based on their last report time, their reliability measure and also importance of data for other teammates.

4.1 Information Level

There are two general strategies for communication in Multi-Agent Systems (MAS) that a team can employ depending on system's general architecture and also on what kind of data the agents intend to share.
High Level Commands: This strategy is best applicable in centralized system architecture where a center commands its agents; therefore, in this method, all critical and high level processes and decision makings are made in center. Consequently, only high level commands are sent to agents in order to make them aware of their behavior.

Low Level Commands: In this method communication system is trying to share all raw data that each agent have and every thing could and should be distributed.

4.2 Centralized vs. Distributed Architecture

Generally, we consider the communications amongst players distributed, but due to the large amount of transmitted data and hence time-consuming processes, agents themselves accomplish their own jobs and broadcast the results, i.e. processes data.
If there wasn't any broadcast feature in our access media, having centralized communication may also reduce number of messages which are needed to share all information among agents.

m = number of messages needed to have all information shared between agents

- With broadcast message:
  - Centralized approach $m = n + 1$
  - Distributed approach $m = n$
- Without broadcast message:
  - Centralized approach $m = n + n = 2n$
  - Distributed approach $m = n \times (n - 1)$

When we are considering our access media properties including its broadcast ability and limited bandwidth and also the fact that defining an agent as center might be unreliable we decide to use distributed communication by broadcasting messages.
The messages contain low level data sensed and acquired by agents from the surroundings such as ball, teammates, and opponent players which are used in localization and updating word model in with each agents self awareness.

5. Localization

Mobile robots must know where there are to operate their tasks properly and this is the first step to having autonomous mobile robot**Error! Reference source not found.** (Kortenkamp et al., 1998). Mobile robot Localization is the process of determining and tracking the location of a mobile robot in global coordinate frame. Localization problem is occasionally referred to as the most fundamental problem to providing a mobile robot with autonomous capabilities. A number of techniques have been used for this, including grid-based approaches or sample-based approaches such as Monde-Carlo. Grid-based approaches require computation of the probability even in the area where the probability is negligible and in the vast environments grids are either too many or big. The former makes communication expensive and the latter makes the results low-resolution. On the other hand, sample based approaches which are mostly based on sampling-importance re-sampling (SIR) algorithms (Rubin, et al., 1988) require the computation of a significant number of samples to support high-performance especially for large areas.
We use a localization algorithm to localize soccer player robots in the field which is called piecewise Linear Probability Distribution Localization (PLPDL) (Vaezi Joze, et al., 2007) that

besides designing for low-performance and low-memory machine can localize robots in the uncertain and dynamic situation. The approach is based on Markov localization (Fox, 1998) which localize robot probabilistically. Our approach inherits from Markov localization the ability to localize a robot under global uncertainty. Using piecewise linear functions to approximate probability distribution functions of these random variables would help our approach to be fast and inexpensive which is suitable for real-time processing of our robots.
The key idea of Markov localization is to compute a probability distribution over all possible position in the environment. Let $l=(x,y,\theta)$ denote a position in the state space of the robot, which x and y are the robot's coordination in the soccer field frame, and θ is the robot's orientation. The distribution $Bel(l)$ expresses the robot's belief for being at position l. Markov localization applied two different probabilistic models to update belief function, an action model to incorporate movement of the robot and a perception model to update the belief upon sensory input.

5.1 Separate Probability Density Functions

If we suppose that probability of $x=x_0$ is independent from probability of $y=y_0$ and also other independencies for other coordinate variables of robot location, we could conclude from independency rule of probability theory that:

$$\begin{aligned} &Bel(l_0=(x_0,y_0,\theta_0))=P(x=x_0\wedge y=y_0\wedge\theta=\theta_0)\\ &=P(x=x_0)P(y=y_0)P(\theta=\theta_0)\end{aligned} \tag{10}$$

This could convince us to use separate probability for each of the coordinates. In contrast with Markov Localization assumption or other grid-base localization coordinate variables such as x are continuous in real environment, so we should consider them as continuous random variables and their density function could be define in the following formula: (notice that we assume these random variables are independent, otherwise we have a three variable density function)

$$P(a<x<b)=\int_a^b f_X(x)dx \tag{11}$$

Equation (11) concludes $f_X(x)$ is non-negative function and $\int_{-\infty}^{+\infty} f_X(x)dx=1$.

Using Equation (10) and definition of density function for coordinate random variables:

$$\begin{aligned} &Bel(l_1=(x_1,y_1,\theta_1)<l<l_2=(x_2,y_2,\theta_2))=P(x_1<x<x_2\wedge y_1<y<y_2\wedge\theta_1<\theta<\theta_2)=\\ &P(x_1<x<x_2)P(y_1<y<y_2)P(\theta_1<\theta<\theta_2)=\int_{x_1}^{x_2} f_X(x)dx\int_{y_1}^{y_2} f_Y(y)dy\int_{\theta_1}^{\theta_2} f_\theta(\theta)d\theta\end{aligned} \tag{12}$$

The product of these functions obtains belief of robot to be at this position, namely $Bel(l)$. So our new localization method considers a separate probability density function for each variable (such as x, y and θ for mobile robot in 2 dimension environments). In Markov Localization0, for each position in the area $l = (x_0, y_0, \theta_0)$ there is $Bel(l)$ which means the robot's belief for being at position l. In contrast, in Probability Distribution Localization (PDL), we have three probability density functions to express our belief for location of robot. This model could be used for current location, new observation and also differential motion, i.e. $(\Delta x, \Delta y, \Delta\theta)$. We need to update current location of robot in the case of motion and reading sensors. In PDL, Motion Update is corresponding to robot motion of Markov Localization and Sensor Update is corresponding to sensor reading in Markov Localization.

- ***Movement Update:*** We consider X a random variable and its probability density related to x position of robot and ΔX as a random variable of movement of robot in x dimension and its probability density. So the new value for X will be $X + \Delta X$. In this way the corresponding density function for X is obtained. We know from probability theory that if X,Y,Z are random variables and Z=X+Y so probability density of Z could be conclude by convolving probability density of X and Y. And also other random variables will be updated independently using motion data that should be in the form of different probability density function for each random variable.
- ***Sensor Update:*** Sensor is usually return to the vision module in robot. However it could be any other sensor for localizing mobile robot. If we suppose sensor data return a probability density function for each variable such as X and p that is the belief of correctness of these data. Now we should create a new probability density for X by the following formula using previous density of X and sensor data in the form of new density function and our belief of its correctness:

$$Fx_{New} = Fx_{old} \times (1 - p) + Fx_{Sensor} \times p \tag{13}$$

Our probability density functions may become worthless after too many movements or sensor updates with small p. So we use the idea of "Sensor Resetting Localization" (Lenser & Veloso, 2000) that considers a threshold for average of p. Some new sensor updates must replace when it becomes lesser than the assigned threshold. This could be translated to threshold for distribution of our density functions. In such cases, more sensor data must be fed by sensor module. The case should happened when the result density function is so distributed (a precise parameter needed for determining threshold that could be variance)
As explained before, Probabilistic Distribution Localization (PDL)'s main output is a probability density function and not a crisp value, but PDL is required to provide more suitable results for other modules such as motion module. So a kind of clustering algorithm can be employed to prepare crisp data as output if it is needed.

5.2 Piecewise Linear Probabilistic Distribution Localization

In this section we are going to change PDL approach in such a way that it becomes simple and suitable for real-time applications for mobile robots. In order to simplify the PDL

process, we employ piecewise linear probability densities in order to make our calculation and storage system much simpler. In piecewise linear probability densities, functions are limited to be made by linear pieces. For storing these functions it is enough to store points which are disjunction of linear pieces. For instance, function shown in Fig. 11, could be determined by set of following points: { (0,0) , (1,.5) , (2,.5) , (3,0) }

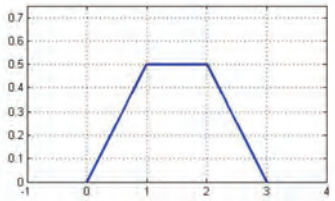


Fig. 11. A sample piecewise linear probability density function

Using piecewise linear function for expressing probability density of location parameter in PDL approximate the main probability density to obtain speed and decrease in memory usage. Storing set of points instead of storing complicated function could help to decrease memory which used for localization purpose. On the other hand applying linear limitation over probability density function could imply appearance of faster algorithms for Movement update and sensor update that are express bellow.

- ***Movement Update:*** With us considering both probability density of random variables X and Y piecewise linear, probability density X+Y is paranoid-segment function. To simplify the process, we approximate such functions to be piecewise linear function (we should also suggest such a converter algorithm). It could be done using convolution of these function as it is discuss before. As an example Fig. 12(a) is probability density function of a random variable before movement update. Fig. 12 (b) shows probability density of movement and Fig. 12 (c) is the final result of movement update. Fig. 12 (d) shows linear approximation of result via PLDL algorithm.

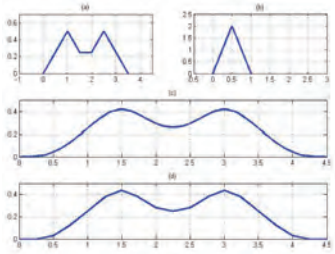


Fig. 12. An example of Movement Update process

- ***Sensor Update:*** This step is straightforward in piecewise linear density function. Using Equation 5-4 we should compute weighted sum of two density function such as Fig. 13 (a) and (b). These functions have stored by set of points so for calculation result we should construct result set using union of x points of both sets with corresponding y that could be calculated by sum of corresponding y. Fig. 13(c) illustrates result of Sensor

Update with p=0.7. Fig. 13(a) is previous density function for a distinct variable and Fig. 13 (b) is sensor data of that distinct variable.

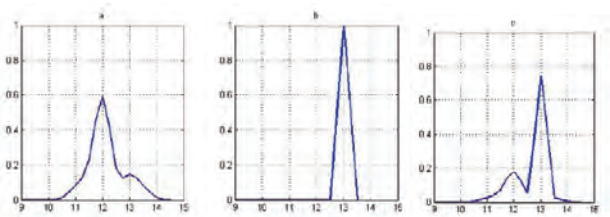


Fig. 13. Sensor Update example. (a) Previous density function (b) sensor data (c) result by p=.7

Also to decrease required memory for storing set of points for probability density functions we omit points that express small amount of information. Strictly speaking, each three consecutive point construct a triangle, we omit central point when the area of this triangle is smaller than a threshold. Additionally, the function is scaled in a way that integral of the function becomes one. We could also describe a parameter as maximum number of points in the set so we could control the computational time needed for PLPDL.

5.3 PLPDL Application

The mentioned algorithm used in the software of controlling Aibo robots for playing soccer as a self-localization sub-system. Vision used as a sensor and movement update support from Motion Controller sub-system. Fig. 14 demonstrates Self-Localization flowchart using PLPDL method. We explained sensor data which is supplied by vision in Section 3 and movement data will discussed in Section 7. As it presents before probability density function for each coordinate variable is stored by set of points. The next sub module is *PDF Filtering* which filters probability density functions in order to omit small values and also non-reliable ones. Then, it is determined if some extra samples are required from Vision sub-System. This decision is made using a threshold over probability density functions' variance after clustering them.

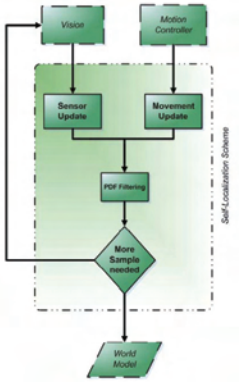


Fig. 14. Self-Localization Flowchart

When it needs more sensor data it sends a signal to vision module to provide localization module with some extra sensor data. This signal also contains information about the accuracy of such a data that may cause executing time-consuming routine in Vision sub-system for more accurate data. Although we do not use information about positions of Aibo robots from external source such as other Aibos using wireless communication, it can be employed as extra sensor data with smaller belief.

6. Decision Making

As explained in our architecture (Section 2), Decision Making (DM) module plays the key role in logical decisions made by agents in a multi-agent environment such as AIBO soccer. DM is done in a completely distributed manner in *Impossibles* AIBO robots; however, communication is employed in order to propagate the information obtained from vision, sensors, and communication modules. Consequently, information is propagated by communication and decisions are made by agents themselves. This section is organized as follows: The intra-DM architecture is explained in details in subsection 1. Team behavior is given in subsection 2. Subsection 3 is devoted to individual behaviors.

6.1 Architecture

Intra-DM module in *Impossibles* AIBO robots have a hierarchical layered architecture (shown in Fig. 15). In fact, DM module consists of two major layers. Team Behavior (TB), i.e. tactics, layer is the highest one which determines the tactics of the soccer team. Secondly, Individual Behavior (IB) layer is the techniques employed by individual players. As demonstrated in Fig. 15, Decision Making (DM) module gets its input from the system's world model including opponent players', teammates', and ball's locations accompanied by some degrees of belief which is due to existence of uncertainty in real system environment such as soccer. Having gotten the inputs from its world model, the AIBO robot will analyze the input in a two-step procedure. Team behavior (TB) sub-module gets DM inputs from world-model and then resolves the whole team behavior, e.g. tactics stored in a database. Finally, TB passes the team behavior and world model information to the lower layer that is Individual Behavior (IB).

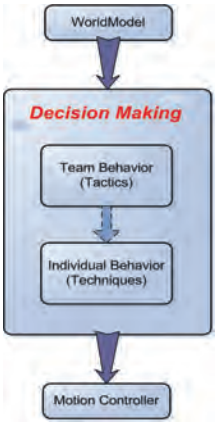


Fig. 15. Decision Making Module Architecture

As the second step, the Individual behavior (IB) sub-module obtains the whole team behavior and world model information form upper layer sub module (TB); then, analyzing its inputs, IB sub-module decides one of the possible actions to do. As a matter of fact, these actions are the outputs of the IB sub-module and hence the outputs of the whole Decision Making (DM) module. This actions set includes (1) shooting in a specified direction with a particular power, (2) blocking the way in a special direction, (3) walking through a path determined by an array of points, (4) looking in one direction, and (5) grabbing the ball.

6.2 Team Behavior

As explained above, *Impossibles* AIBO team tactics is resolved in Team Behavior (TB) sub-module. The final tactics of the team will be selected from tactics database.

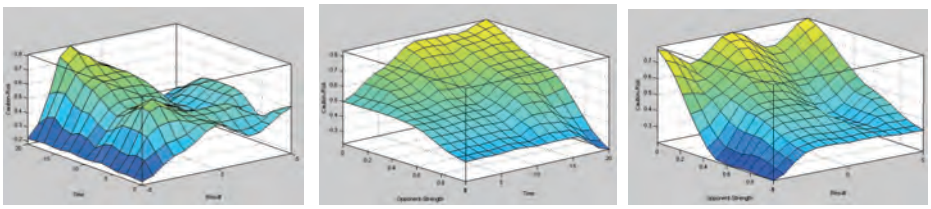
Determining Factors

Tactics selection step needs two parameters. First, fuzzy membership degree in offense set is to be determined, i.e. Defense-Offense (DO). Teammates' and Players' sites is defined to be the second parameter

A) Defense vs. Offense

Tactics of the team is defined by a fuzzy membership degree, i.e. DO, in offense set. Between complete defense condition, i.e. 0, and complete offense condition, i.e. 1. The following three parameters are calculated and then employed to obtain the membership degree.

1. *Caution and Risk:* The first parameter which contributes to obtain DO is Caution-Risk (CR) fuzzy membership degree. We have employed a fuzzy logic controller which outputs CR degree. This fuzzy logic controller gets three inputs: The result of the game, time, and opponent's strength.



(a) (b) (c)

Fig. 16. Caution and Risk which is the output of fuzzy controller for (a) a fixed result (b) a fixed Opponent-Strength (c) a fixed Time

2. *Ball Ownership:* Ball ownership is a critical factor which contributes in producing the final selected team strategy of an AIBO soccer team. In real world soccer environment we can define ball ownership as a crisp value which at least last long enough to determine team strategy. In fact it can be represented via digital magnitudes such as a Boolean variable. Ball ownership in AIBO cannot be defined in such a way. Because in AIBO soccer game robots intermittent lose the ball; therefore, selected team strategies will be changed so irregularly that it becomes impossible for a team either to defend or attack. Here we define a fuzzy membership degree in a Ball Ownership (BO) set. In

Impossibles robots, the following formula is employed to calculate the BO of the team in order to evaluate the team membership degree in the complete ownership set.

$$BO(Team_i) = \sum_{m_j \in Team_i} \frac{1}{d_j^{\alpha}} \tag{14}$$

The final Ball Ownership (BO) factor is obtained by division of our team's BO and opponent team's BO.

3. *Hyperbolic Danger Safety Degree:* As explained above, Ball Ownership (BO) is a factor due to players' rational locations to the ball; however, players' absolute locations are also important. In order to accomplish the job a Hyperbolic Factor (HF) is defined. A hyperbola (Gray, 1997) is a conic section defined as the locus of all points P in the plane the difference of whose distances $r_1 = F_1P$ and $r_2 = F_2P$ from two fixed points (the foci F_1 and F_2) separated by a distance $2c$ is a given positive constant k, $r_2 - r_1 = k$. Letting P fall on the left x-intercept requires that $k = (c+a)-(c-a) = 2a$. So the constant is given by $k = 2a$, i.e., twice the distance between the x-intercepts (left figure below).

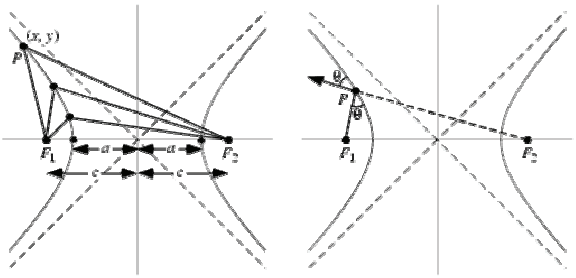


Fig. 17. Hyperbola used to calculate Hyperbolic Factor (HF)

We think of the AIBO soccer field to be locus of hyperbolas with variable positive 'a' and goals to be the focuses of theses hyperbolas; therefore, 'c' is defined to be $0.5 \times Field_Length$, i.e. distance of the goals from the center of the soccer field. Each point in the field is defined to have a danger degree (DD) if an opponent team member is located in this point. On the other hand, Safety Degree (SD) is defined if one of our team members is located in that point.

$$DD(P_i) = \frac{\text{distance}(P_i, \text{opponent Goal}) - \text{distance}(P_i, \text{our Goal})}{Field_Length} \tag{15}$$

$$SD(P_i) = \frac{\text{distance}(P_i, \text{our Goal}) - \text{distance}(P_i, \text{opponent Goal})}{Field_Length} \tag{16}$$

According to above equations, points on a hyperbolic locus with a constant 'a' will have equal Danger Degrees (DD) in the case of having an opponent player in the point. Similarly, the points have equal Safety Degree (SD) if one of our team members is situated in one of those points. The final Hyperbolic Danger-Safety Degree (HDSD) factor is calculated employing players individual Danger Degree (DD), or Safety Degree (SD).

$$HDSD = \frac{f_1(SD(\text{our team members}))}{f_2(DD(\text{opponent team players}))} \tag{17}$$

As a significant factor, $f(x)$ is selected according to coach basic idea of either defensive or offensive strategies. We have employed '*Averaging*' function. Therefore, *HDSD* factor is evaluated:

$$HDSD = \frac{\sum_{m_i \in \text{our team}} SD(m_i)}{\sum_{m_i \in \text{opponent team}} DD(m_i)} \tag{18}$$

B) Team Fear-Relax Emotional Degree

As explained before, team behavior is determined according to Defense-Offense Degree (DOD) which lies in the range of 0 to 1. In above subsections, three determining factors were defined: Caution-Risk (CR), Ball Ownership (BO), and Hyperbolic Safety Danger Degree (HDSD). Now these three parameters are to be combined to represent the final Team Behavior Defense Offense Degree.

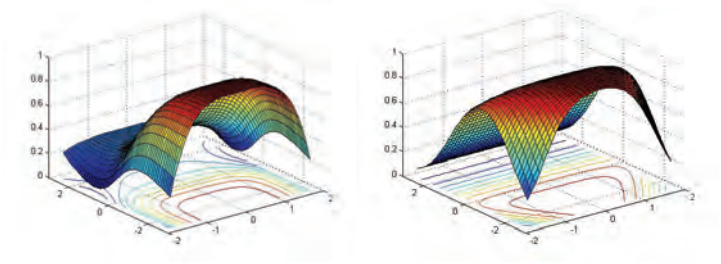
Team Strategy Database

In order to avoid having CPU over usage problems, we save predefined team strategies in a Team Strategy Database (TSD). In each moment of the game, a linear combination of the proper strategies is computed based on TBDOD and players' locations. Selecting from TSD offers two priorities over computing dynamic team strategies. First it supports to have a more flexible team behavior, because further team strategies can be added to TSD later. Second, this approach helps to decrease the CPU usage.

Generally, team strategies are categorized into three groups. Defensive strategies, midfield strategies, and offensive strategies are the mentioned groups. Two independent defensive team strategies (DTS) of *Impossibles* AIBO robots are presented. **Error! Reference source not found.** demonstrates the first DTS in which our players try to defend opponent's forward players reaching the goal along a line from the center of the field to our goal. It is the most defensive strategy of the team employed in critical circumstances.

Note that the following surfaces represent the values of points on the soccer field according to the team strategy. For instance, the dark red points in the diagrams indicate the most important regions of the field. On the other hand, the blues ones denote the regions which are not considered as significant regions. To clarify the problem it may be useful to declare that (0, 2.7) is the center of our goal. Fig. 18(b) demonstrates the general defensive (GD)

strategy of the *Impossibles* AIBO robots employed to some objectives such as preventing the game result being changed.



(a) (b)

Fig. 18. (a) the maximum defensive team strategy. (b) General Defensive (GD) team strategy

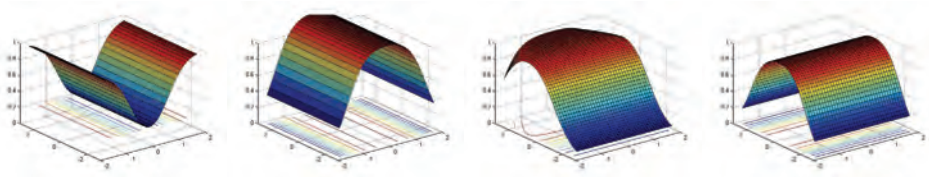


Fig. 19. Four basic strategies

Midfield and offensive team strategies of *Impossibles* are in fact generated easily as a combination of the following basic strategies (Fig. 19).
In Fig. 20 denote midfield and offensive strategies of the team. These strategies are generated using the above basic strategies. Here we have employed the simplest operation, i.e. multiplication, to produce midfield and offensive team behaviors.

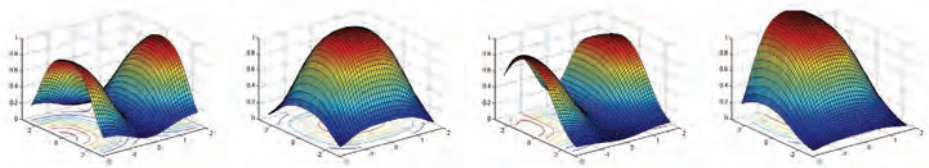


Fig. 20. Four strategies which generated using combination basic strategic

Emotional Tactics Selection
One of the most important aspects of human decision making is the role of emotions in its behavior and reactions. Humans choose their actions and make their decisions due to several internal variables called emotions. Emotional decision making is employed by humankind to avoid time-consuming and computationally-expensive approaches, e.g. using mathematical equations in decision making, to optimize the final result in critical circumstances such as danger (Locus et al., 2002).
In order to reduce the computation burden of the team behavior generation, we exploit Emotional Tactics Selection (ETS) approach. As presented in, agents' Emotional Decision Making (EDM) is based on their different states, called emotions. Up to now the robots have

calculated the fuzzy emotional Fear-Relax (FR) degree of the whole team. Given this degree, robots are to compute the team strategy, i.e. a linear combination of Team Strategy Database (TSD) elements. Fundamentally, in, transitions from one emotional state to the other are thought of being a gradual change, i.e. not suddenly, as it is done in real world animals and human. Therefore, we avoid quantization of the given FR degree. In contrast, based on the given FR degree, a linear combination of the strategies in TSD is computed as the whole team behavior, i.e. team strategy.

6.2 Individual Behaviors (Techniques)

Impossibles AIBO robots make use of a priority-based selection approach to choose the most proper action in various situations. In other words, after calculating some mathematical equations, each possible action is assigned to have a score; then, the most appropriate action is chosen. For instance, a player, who has the ball ownership, can select one of the following actions: (1) Moving with ball, (2) Passing to a teammate player, (3) Shooting toward the opponent team's goal, or (4) Looking around. In this section, different individual behaviors are explained logically. A lower level design is provided in Motion Controller (MC) section.

Predefined Dynamic Assigned Regions

Generally, in multi-agent systems, decision making can be accomplished using one of these four solutions (Habibi & Nayeri, 2006): (1) No Sharing Decision Making, (2) Information Sharing Decision Making, (3) Centralized Decision Making, (4) Fully Centralized Decision making. *Impossibles* AIBO robots use the second approach. In this method, communication is employed just for transporting the information; therefore, neither commands nor decisions made by center are transmitted.

With us using Information Sharing decision making solution, the most critical problem was similar behaviors of the robots in the same situations. So robots are assigned predefined roles as in real world soccer.

Players are assumed to have tendency toward their dynamically assigned regions. This tendency is represented by a simple spring; hence, there will be a linear dependency ($\alpha = 1$) between the player's tendency and the distance from its current location to its assigned region.

$$\text{Tendency}(P_i) \;=\; K \times \text{distance}\big(\text{Location}(P_i),\ \text{Assigned_Region}(P_i)\big)^{\alpha} \tag{19}$$

Where 'K' is a positive constant which can be learned. Experiences show that setting 'α' to be 1.3 results in the best known outcome.

Outputs as Decision Making-Motion Engine Interface

Last of all, having logically produced Individual Behaviors (IB) of the players, Decision Making (DM) module passes IBs to the lower level module, i.e. Motion Controller (MC); therefore, IBs are considered to the interfaces between the DM and MC modules. In this section the employed Individual Behaviors are explained logically.

1. Looking: The objects stored in World Model (WM) own a saved parameter called Update Time (UT). It denotes the last time when a particular object has been seen by an

agent. So it may be necessary for agents to refresh their knowledge about their surroundings limited to their vision capability, i.e. approximately 1.5 meters.
2. Running: Walking and Running as two basic categorizations of motion should be implemented efficiently because of their importance. A lot of learning algorithms on AIBOs have been presented during the past few years (Kohl & Stone, 2004).
3. Ball Grabbing: In order to get the ball ownership, Decision Making (DM) module of a player passes 'GRAB' to the lower level module, i.e. Motion Controller (MC).
4. Shooting & Passing
5. Blocking

7. Motion

Different approaches can be considered as the way to make AIBOs move in a proper order, all accompanied by number of advantages and disadvantages. Enhancing robot's movement by restricting it to mathematical formulas and geometrical models, dividing motion skills into some predefined and atomic consecutive series of actions, using simulation results for realistic environment, eliciting models for each of the skills based on experimental data and learning algorithms, and so on, are instances of how to obtain methods for robot's motion. Studies conducted on all approaches, led us to a hybrid, innovative method with the most consistency to other modules, chiefly Decision Making.
We divided motion skills into two categories based on their usage: Blocking Skills and Non-Blocking Skills. While performing a non-blocking skill, Decision Making (DM) module can make new decisions if necessary, and send an interrupt to Motion Controller module. Thus, Motion Controller will preempt or halt the previous action and start performing new commands. Walk and rotate are basic actions included in non-blocking skills. On the other hand, Blocking Skills are smallest consecutive segments done atomically. Different kinds of shoots as well as ball blocking actions are examples of blocking skills.
We mapped the movement of robot's joints, while running, to a geometrical space so to have a complete set of parameters for Machine Learning techniques. The geometrical shape to which we modeled robot's movement was an ellipse. As a result, the main task of this module is done through some offline processes to improve ellipse's parameters in a way to reach a better speed and increase the accuracy of walks.

7.1 Architecture

This module, Motion Controller (MC), provides an interface of high level commands for DM module, such as *walk, look, localSearch, ballGrab, defend, block,* etc. When these high-level commands are received from DM module, a planning algorithm is used to break them into a series of low-level commands to satisfy the robot's conditions and team strategy. Finally, all generated data are converted to physical joint values inside the Motion Maker layer. Based on the attributes of each skill, the final data will be put in Motion Queues with the special characteristics of Blocking or Non-Blocking actions. Data inside the queue are considered as segments.
Segments are treated as atomic actions. These include some points in a 3D coordinating system, for example the coordinate of paws, or the camera in relation to body's center. So, in order to convert these points to joint values, we used Inverse Kinematics functions for both head and legs.

Based on aforementioned architecture, design and implementation of the MC in layers can avoid complexity and will increase simplicity of defining new skills. Fig. 21 shows the architecture of MC module in brief.

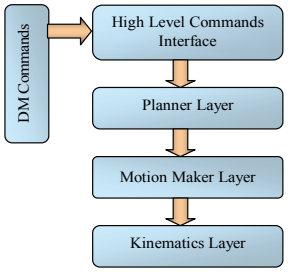


Fig. 21 Architecture of Motion Controller Module

7.2 Blocking Action

Blocking Skills are smallest consecutive segments which can not be preempted or halted while being processed. So Decision Making should use them in suitable situations. These actions contain consecutive value of joints which can be utilized from a static look up table that does not change during the game. We use our structure to store these date in files; therefore we have separate configuration files for each blocking skill. These actions contain all types of shoots, blockings by goalie and stopping the ball.

7.3 Non-Blocking Action

None-Blocking skills are those who accept interruptions. While Aibo is processing and running non-blocking commands, DM module can decide new actions and send them to the robot. In this case, the robot will halt the current job and start new task. These actions are necessary due to uncertain conditions of a soccer game. Walking and ball grabbing are simple non-blocking actions.

7.4 Movement Parameter

We modeled the movement of a robot to the movement of AIBO's paws on the perimeter of an ellipse. Although the Aibo leg's degrees of freedom are less than natural animals, this model resembles the natural walk of them more.

To form the ellipse based on which the robot plans to move, walk needs number of parameters. These parameters are as follows:

- Semi-major axis of the ellipse, with the symbolic name of a, for both front and rear legs
- Semi-minor axis of the ellipse, with the symbolic name of b, for both front and rear legs
- Coordination (x_0, y_0, z_0) of the center of the ellipses for all legs in relation to body's coordinating system.
- Angles alpha, beta and theta which gives the ellipse all degrees of freedom. So we can rotate the ellipse around each of the 3D coordinating system's axis.

Formulating movements of joints has additional advantages affecting the performance of other modules, especially Localization. By the use of geometrical models an increase in

accuracy of estimations is gained, although most of estimations are based on experimental data. This estimation helps Localization module to localize the robots without the use of camera or other sensors in an average period.
Computation of new walk parameters and constructing new motion skills by combining the previous motion skills is also another advantage of formulation. Based on some predefined parameters for basic skills in addition to mathematical models for skill combinations, there would be a chance to gain new motions. For instance, by merging the parameters for a simple forward walk and anticlockwise rotation, a new motion can be constructed to circulate around a circle with specific radius; this skill, with the radius as an input parameter, can be used in ball grabbling action.

7.5 Estimation

Since repetition of intricate geometrical solutions based on image processing algorithms may appear expensive in many cases, motion approximations for movements may be used to increase the accuracy of total estimations, and to lessen the cost of localization methods. The mechanism for motion estimations is based on robot's movement, in any of the directions, rotations and mixture of all couples of skills.
For each of the basic skills, X movement (d_x), Y movement (d_y) and angular movement (d_t) are estimated and written as attitudes for their own parameters. Whenever a skill is being performed by the robot, these movements will be calculated for each time slice first and finally will be calculated for robot's total walk. A simple way of calculating the estimations based on d_x, d_y, d_t is shown in Equation (20). Then all these factors are reported to Localization module to estimate the location of the robot.

$$D_x(S) = d_x(S) * t_p(S) \ , \ D_y(S) = d_y(S) * t_p(S) \ , \ D_t(S) = d_t(S) * t_p(S) \qquad (20)$$

These estimations cause some problems in some cases. Disadvantages of this method appears when each skill is performed for short time slices –a result of quick changes in decisions- or when there is a great change –unsmooth field for example- in the soccer field. Therefore deciding when to use these estimations can be critical and they can cause failures or inaccurate localization. So, to prevent these problems, we can avoid estimating quick actions (performed less than a specific amount of time) and to make robots learn the parameters of the skills when the condition is changed.
Estimation, in the case of great reliability, can be considered as a helpful component for localization and it is worthwhile working on motion estimations in the future.

8. Tools & Debugging

Running compiled code on AIBO is time-consuming and inefficient in many cases. Thus, lack of debugging software and simulator for testing and debugging purposes is felt and the need is obvious.
Therefore, Impossibles spent time on writing tools which let us debug the codes running on Aibo platform and simulate some geometrical codes without having robots available. These tools consist of AIBO Controller and AIBO Geometrical Simulator.

8.1 Aibo Controller

This software is divided into two units cooperating with each other. A *DEBUG* module which is written into memory stick (which can be accompanied by other soccer modules and run in parallel with them) and a client application which runs on the PC.
DEBUG module sends collected data to the client program. Data consist of all internal states of other soccer or non-soccer modules (provided for other challenges) in addition to other internal representations of the robot (including joint data, images, body sensor data, world states, perceptions, etc.)
On the other side, the client application receives the data and makes it possible to visualize and analyze them via different internal implemented algorithms and other user defined ones. This will give us opportunity to run vast of algorithms and methods in different fields (Image Processing, Motion, etc.) based on data collected.
On the other direction, since reaching to the state which revealed bugs in a nondeterministic and real environment is somehow impossible, setting parameters to return back to the desired state, can be found in our controller application.
Channel between *DEBUG* module and the client program is a wireless connection. Results from experimental statistical analysis of protocols in a wireless communication system led us to choose packet sizes and protocols in a way to achieve higher throughput.
To conclude, a list of features provided by *AIBO Controller* is followed:

- *Visualization* and *analysis* of data, especially intermediate representations of other modules running in the robot.
- Turning *on* or *off* different algorithms and parts of the code for debugging purposes.
- *Modification* of robot's state and parameters to algorithms.
- Run different algorithms solely on the collected data on the client side instead of the robot itself.
- Designing new blocking skills by reaching to desired and discrete states.

8.2 Aibo Geometrical Simulation

Writing codes to memory stick after each change and waiting for the robot to load modules, all occur for several times and all slow down the process of code development. Thus, a simulator was developed by *Impossibles* to simulate some special purpose codes on the PC instead of the Aibo itself.
One of the most time-consuming processes is the geometrical challenges of robot. Geometrical Simulator makes it possible to simulate motion's geometrical space on the client side instead of the robot itself. This will give us features to develop forward and inverse kinematics methods for any kind of robots, simulating and collecting the results of any kind of movements, and, designing in companion with testing the movement of robot's joints during blocking actions.

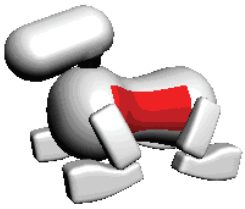


Fig. 22. Aibo model in Geometrical Simulator

This software is fed by the codes generating motion steps as inputs and the output will be the graphical movements of our Aibo model's legs and head. The Aibo model in our simulator is shown in Fig. 22.

8.3 Code Interpreter

Most of the codes in a soccer module are dependant on the condition in which the robots are playing. These include game strategies, player's roles and some complex decision making commands. Therefore in order to make the code flexible and prevent changes in the hardcode each time one of the above alters, converting hardcode into scripts which can be interpreted by the AIBO is an essence.

Scripts facilitate programming Aibos when accompanied by a high-level interpreter module, which runs inside the robots. When the fundamental parts of a soccer software is developed, the codes for decision making methods, determination of player's roles and game strategies are written in a simple scripting language and all are stored in memory stick as a raw text file. An interpreter module is in charge of translating the scripts into standard codes for Aibos and running them logically and consecutively.

These scripts use a high level interface provided by the main modules in our architecture, as mentioned earlier in this chapter, to fulfill the need of a potent access to hardcode. In addition to the interface of the main modules, our interpreter supports structural statements such as loops and conditional statements. There are some data structures provided for better manipulation of collected data from camera, joints and other sensors.

Players change roles dynamically, so putting them in hardcode would be bothersome. Some predefined functions in our interpreter make it feasible to distribute roles among players dynamically and prevent any sort of conflictions. This way even if the team strategy is changing dynamically in a specific game, responsibilities are assigned to players in a correct manner. Taking absence of players into account –when penalized- is another specification of those predefined functions.

Complex decisions can be implemented based on a State Diagram Machine (SDM) in our script. All decisions are first converted to states for simplicity, and then implemented by the features available in our scripts. SDM makes it possible to easily change decision steps and its contents.

9. Conclusion

In this chapter we presented the Impossibles main architecture and its modules to create a fully autonomous team of 4-legged robots for playing soccer. This architecture includes different modules such as World Model, Vision, Decision Making, Motion Controller, Communication, and Localization which are all independent of the robot platform. This chapter presented the integration of our researches in different fields which came together to create fully autonomous robots for specific purpose that is playing soccer as humans do. And this could be a primitive attempt to developing intelligent robots.

Participated in two years of RoboCup competitions in Bremen and Atlanta in RoboCup 2006 and 2007, the team gained some valuable experiences which led to designs of low cost algorithms. The most probable restriction in this soccer module is the limited resources such as CPU and memory, therefore developing optimized algorithms is the main target of team achievements.

10. References

Gonzalez, R. C. & Woods, R. E. (2001). *Digital Image Processing,* Prentice-Hall, 2nd edition.

Gray, A. (1997) Modern Differential Geometry of Curves and Surfaces with Mathematica, Boca Raton, CRC Press, 1997.

Fox, D. (1998). *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation.* Institute of Computer Science III, University of Bonn, Germany, Doctoral Thesis, December 1998.

Habibi, J. & Vaezi Joze, H. R. (2005). A new Architecture for Multi Agent System in Rescue Simulation Environment. *the CSI Journal on Computer Science and Engineering,* Vol. 3, No. 2&4, pp. 1-7.

Habibi, J.; Vaezi Joze, H. R.; Aliari Zounoz, S.; Rahbar, S.; Valipour, M. & Fathi, A. (2006) Impossibles Aibo Four-Legged Team Description Paper RoboCup 2006, *RoboCup 2006*, Bremen, June 2006.

Habibi, J. & Nayeri P. (2006) Centralized vs. Non-Centralized Decision-Making in Multi-Agent Environments, *11th CSI computer Conference*, January, 2006.

Jain, R.; Kasturi, R. & Schunck, B. G. (1995). *Machine Vision*, McGraw-Hill.

Kohl, N. & Stone, S. (2004) Machine Learning for Fast Quadrupedal Locomotion, *In Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*, San Jose, CA, July 2004.

Kortenkamp, D.; Bonasso, R. P. & Murphy, R. (1998). *AI-based Mobile Robots: Case studies of successful robot systems*, MIT Press, Cambridge.

Lenser, S. & Veloso., M. (2000). Sensor resetting localization for poorly modelled mobile robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, 2000.

Locus, C.; Shahmirzadi, D. & Sheikholeslami, N. (2002), Introducing a Controller Based on Brain Emotional Learning Algorithm: BELBIC (Brain Emotional Learning Based Intelligent Controller), *International Journal of Intelligent Automation and Soft Computing (AutoSoft)*, USA, August 2002.

Mokhtarian, K.; Vaezi Joze, H. R. & Habibi, J. (2007). An Inexpensive Approach for Real-Time Vision on Four-legged Footballer Robots. *Proceedings of 12th International CSI Computer Conference*, pp. 1856-1861, February 2007.

Rubin, D.B.; DeGroot, K. M.; Lindley, D. V. & Smith, A. F. M. (1988). Using the SIR algorithm to simulate posterior distributions. In M.H. Bernardo, *Bayesian Statistics 3.* Oxford University Press, Oxford, UK.

Sridharan, M. & Stone, P. (2005) Real-Time Vision on a Mobile Robot Platform. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2148-2153, August 2005.

Vaezi Joze, H. R.; Habibi, J. & Rahbar S. (2007). Piecewise Linear Probability Distribution Localization: Fast and Inexpensive Approach for Mobile Robot Localization. *Proceedings of 4th TAROS Conference*, wells, September 2007.

Vaezi Joze, H. R.; Mokhtarian, K.; Asadi, N.; Kamali, A.; Zolghadr, N. & Kaffash, S. H. (2007). Impossibles Aibo Four-Legged Team Description Paper RoboCup 2007, *RoboCup 2007*, Atlanta, July 2007.

26

RoboCup@Home: Creating and Benchmarking Tomorrows Service Robot Applications

Tijn van der Zant
University of Groningen
The Netherlands
Thomas Wisspeintner
Fraunhofer Institute IAIS
Germany

1. Introduction

RoboCup and robot soccer have seen an enormous growth in the past decade, constantly widening the range of used technologies and scenarios. @Home is a new league in RoboCup with the aim to foster the development of applications in the domains of service and assistance robotics, ambient intelligence and human-robot interaction. In this book chapter, the underlying concepts of this new league are introduced. There are strong relations between the @Home league and the soccer competitions. An example is 'Natural Interaction with robotic systems'. If soccer robots are to play against humans, they should be able to understand the human environment, the 'umwelt' that humans are living in. After the introduction, we motivate the foundation of this new league by giving some philosophical background.

Then, the structure and rules of the competition are explained. The league uses a set of independent benchmarks that test certain abilities in the domains of human robot interaction, manipulation, navigation, localization, and human and object recognition. Adapting, enhancing and integrating these benchmarks consequently over the years offer the opportunity to guide research and development toward robust, useful and applicable solutions.

Not all benchmarks are predefined. An Open Challenge is provided as a platform to generate new, innovative and possibly unconventional ideas. Promising ideas are then used to adjust the road map of the league. The scope of necessary technologies is wide. Teams have to integrate many different technologies, communicate, exchange knowledge and develop multi-purpose components that can be transferred between different leagues to be successful in the competition. A central goal of @Home is to provide platforms for exchange and standardization of science and technology in the robotics domain.

Section 4 points out how the @Home league is actually building on top of the technologies developed by doing robotic soccer for the past decade. The @Home league is not only using technologies of the soccer leagues, but it is also generating new technologies. By comparing robot vs. robot soccer with @Home, and regarding the technologies we probably need to win in 2050 from humans, it becomes clear that @Home is researching technologies that can

be transferred back to the soccer leagues once the robots start playing against humans. At the RoboCup world championships in Atlanta 2007 already the first action in this direction was demonstrated where the RoboCup trustee board played against the robot world champions of the Mid-size league. It is needless to say that the humans still played a lot better with the robots being fooled most of the time. Though it was an important step towards the big goal, it showed that the robot have no understanding of the world.

2. Robotics: Philosophy of Mind Using a Screwdriver

The title of this section is borrowed from I. Harvey (Harvey, 2000). He describes how the philosophical stance of the designer is reflected in the design choices being made during the creation of a robotic system. The background ideas of the designer are an important aspect of the design process. The @Home league addresses these philosophical stances of the robot designers by going into the real world where it just might be that the ideas and constructs that hold in an artificial environment (such as a soccer game) are not true and/or do not work anymore. Robotic systems that act in the real world are no longer shielded from unwanted influences. This implies also that the designers of these systems cannot hide from the problems of the real world. The idea behind robots playing soccer is to go from a static environment into a dynamic environment (Kitano et al., 1997). The idea behind robots assisting humans in a real world environment is to go from a dynamic and structured environment into a dynamic and unstructured environment that includes humans (van der Zant & Wisspeintner, 2005).

Some of the advantages of going to the real world are that researchers can work on robots that might be able to do something useful in human society and that it is easier to demonstrate that working with robots is not only about making a great toy. These two aspects are covered in section 3.3.

One point of view is that the RoboCup is a practical investigation of the mind-body problem. Basically there are two sides on this topic. Some claim that the mind and body can be treated as separate entities, often referred to as GOFAI systems (Good Old Fashioned Artificial Intelligence) or classical Artificial Intelligence. The New Artificial Intelligence or embodied cognitive science paradigm on the other hand states that the mind, body and environment are all connected. They should not be treated as separate entities that can be studied apart from each other. For an excellent introduction on these topics, see (Pfeifer & Scheier, 1999). If this is true then it is very important to pay close attention to the environment where the robots are functioning in. Having the real world as the environment of our robotic systems could mean that a shift in theories is necessary for the creation of artificial devices that exhibit general intelligence.

Usually the goal of the RoboCup federation is captured in the "Winning Soccer in 2050" statement. This is not the only thing that RoboCup is about. It is stated very clearly RoboCup website that "RoboCup is an international joint project to promote AI, robotics, and related fields. It is an attempt to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined."

Although it is clear that robotic soccer includes very important research topics, it might not address al the issues involved in the creation of robots that exhibit general intelligence. Even the best human soccer players do not always excell in general intelligence. It is therefore a

legitimate question to ask if robotic soccer will lead us towards robots that show intelligence on a broad level.

3. @Home in a Nutshell

3.1 The General Idea

RoboCup@Home is a new league inside the RoboCup competitions that focuses on real-world applications and human-machine interaction with autonomous robots. The aim is to foster the development of useful, general and robust robotic technologies and applications that can assist humans in everyday life. The competition consists of a series of predefined tests, an Open Challenge and the finals where the teams are free to demonstrate new abilities and robot applications. While in the beginning necessary base abilities are being developed, tests will focus more and more on real application scenarios with a rising level of uncertainty, requiring a high grade of system integration.

3.2 The Scenario

At the moment the competition takes place in is a constructed living room scenario. To foster general solutions, the scenario is not standardized or pre-defined So shape, walls, floor or furniture change every time. Within a few years some of the tests should be held in the real world environment like a real supermarket where the robots have to assist humans with shopping. To foster advance in technology and to keep the competition interesting, the scenario and the tests will steadily increase in complexity.

3.3 Social Relevance

The applications that are being tested for should have social relevance. Through showing what robots can already do, people can see for themselves what robots could do for them. It is important to create public awareness. Robots that assist humans in every day situation are easy to relate to. If robots are to be accepted by humans they should also be appealing

There are several implications with the 'social relevance' argument. One of them is that the robots should behave appropriately if they ever are to be accepted. This is called 'robotiquette' in (Dautenhahn, 2007). She argues that it is important that not only the robot is part of the research about robots, but also how humans react to them.

Another aspect is that robots should be social. Humans interact differently with each other than with non-humans, such as animals, robots and objects. Social intelligence predates object intelligence. Primate intelligence evolved in social situations, where it was important to be smarter than the other primates of the group in order to create the best chance to procreate. This is called the social brain hypothesis (Dunbar R., 2003). Even animals that do not have the possibility to manipulate objects show social intelligence. This might imply that in order to create robots that show general intelligence, it is important that researchers create robots that exhibit social intelligence, also called sociable robots (Breazeal, 2002).

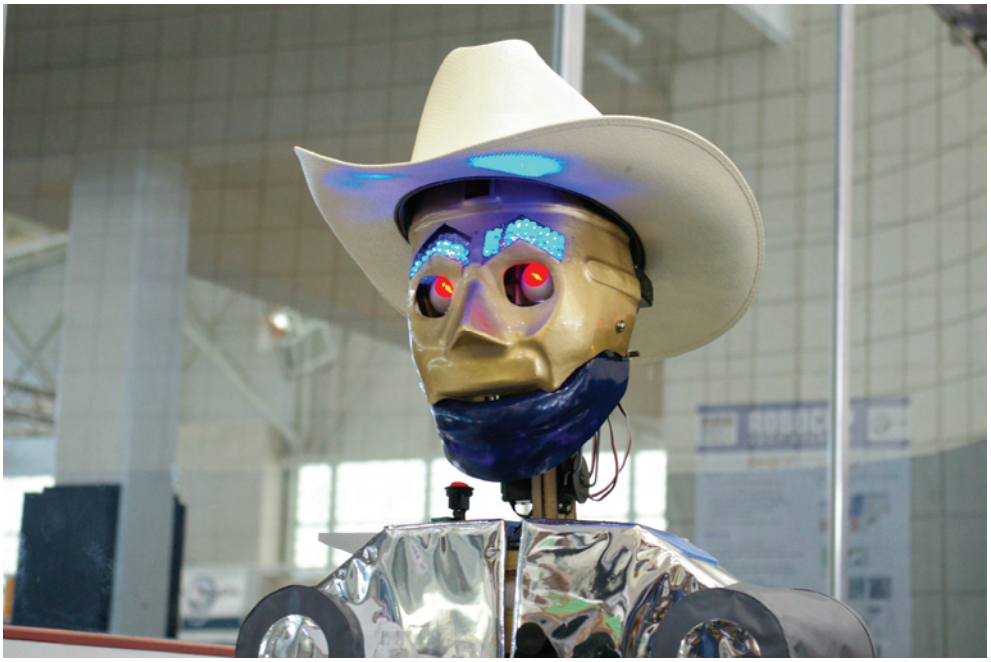


Figure 1. The @Home robot from the Pumas team (Mexico), lead by Jesus Savage, trying to convey some emotions. The first steps towards sociable robots?

3.4 Tests

Tests evaluate predefined base abilities which form the core of the league. Each test consists of a 'proof of concept' and a 'general applicability' phase. During the proof of concept uncertainty and complexity is limited by allowing the team modify the environment, doing the set up and the execution of a test. This ensures a low entry level for new teams. In the second phase the setting up and execution is done by external people with restrictions on the use of aiding technologies (like markers or external devices). In both cases set up time is limited to a few minutes to foster effective and simple calibration and set up procedures. For example, in the 'Follow and Guide' test, the robot has to follow a person and guide him back to the start location. The setup time is limited to a maximum of one minute including calibration on the person. The automatic calibration procedure can be started by a single push of a button or by using a voice command. In other tests the following questions are currently being addressed:

- Can a robot recognize and identify persons?
- Can it open a door?
- Can it navigate robustly in natural environments ?
- Is the robot save to handle?
- Does it have an appealing appearance?
- Does it allow for cooperation with humans
- Can it communicate with other intelligent systems and humans
- Can the robot manipulate random objects?

- Does it have a basic understanding of the world?

The investigated capabilities will change over time and highly depend of the already achieved skills. The combination of predefined, independent tests and the Open Challenge, where these and new abilities should be integrated to form new application scenarios, support a gradual and iterative development. On purpose, the league does not have an ultimate goal like the soccer leagues do. Instead it functions more as a roadmap generator, with the league responding flexibly to new improvements in technology and new upcoming application.

4. Winning in 2050?

To win with soccer playing robots in 2050 implies that the robots should be humanoids and preferably androids (MacDorman & Ishiguro, 2006), with the same or similar physical capabilities that humans posses. The needed capabilities are still far away from what is common in the present day. It is the firm believe of the authors that the @Home league is investigating also the technologies that are needed by the soccer playing robots of the future. It is very likely that a lot of the science and technology from the @Home league will be transferred to the soccer leagues once they start to play against humans.

Technology/environment	**Robot-robot soccer**	**@Home**	**Human-robot soccer**
Object recognition	+	+	+
Navigation	+	+	+
Dynamic environment	+	+	+
Autonomy	+	+	+
Reliability	+	+	+
Specialized AI	+	+	+
General AI	-	+	0
Language understanding	-	+	+
Natural human-robot interaction	-	+	+
Reasoning capabilities	0	+	0
Adaptive vision	0	+	+
Adaptive behaviour	0	+	+
Recognition of human intentions	-	+	+
Understanding of human emotions	-	+	+
Recognition of human behaviour	-	+	+
Recognition of individuals	-	+	+
Localization in unstructured environments	-	+	+
Learning by example	-	+	+
Manipulation of random objects	-	+	0

Table 1. Overview of research aspects for robot vs. robot games, @Home league and humans vs. robot games. '-' means unlikely, '0' means probably and '+' means very likely

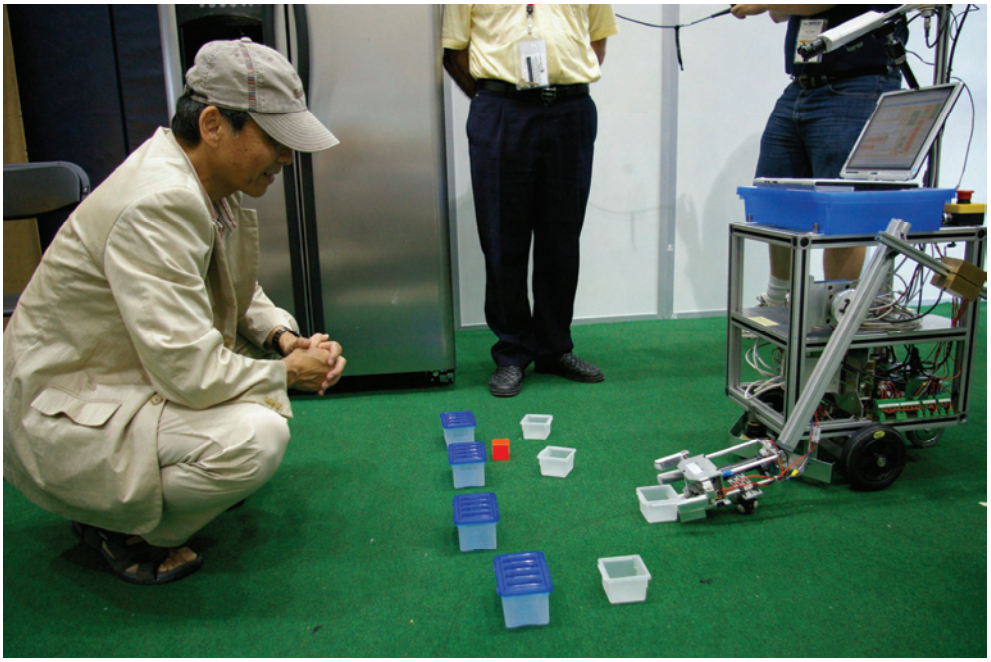


Figure 2. Human-robot interaction. In this picture the president of the RoboCup federation, Minoru Asada, is playing a game with the @Home robot from the RH2-Y team (Switzerland), lead by Jean-daniel Dessimoz

Table 1 tries to give an overview of relevant research topics and requirements for playing soccer with robots playing against robots, for the @Home league and those that are probably required by androids winning a soccer game in 2050.

At the moment issues such as the understanding of language or recognizing certain humans are not tackled. Issues such as the recognition of emotion are even further down the road. It is already difficult enough to recognize one's own team mates and the analysis of the behaviour of the opponent is so difficult that hardly any team is trying it at all. Also localization is not easy, especially not on large soccer fields. Could this have something to do with the environment? Localization is usually about recognizing the corner poles, the white lines and the goals. But when I, as a human, am on a soccer field then there is usually one tribune (always empty in my case) and on the other side there are often trees or a clearing. I do not have to recognize the cornet poles, or recognize the goalkeeper's face from a distance, or do complex mathematics on white lines in the grass… Instead I look if I see the tribune or the trees and I know exactly which way to dribble with the ball. Maybe it is also an idea, when one is working on a soccer robot, to take the rest of the world into account.

In the first two years of the @Home competitions it became very clear how important soccer research has been and still is to be able to perform decently in the @Home league. For example, the Allemaniacs' team (Schiffer et al., 2006), the @Home world champion in 2006 and 2007 has been participating in the mid-size league for years, and they even used a modified mid-size league robot.

5. Future Work

What can be expected in the near future of the @Home league? In the next five years the plan is to introduce cooperative tasks between humans and robot. An example would be cooking together, where the robot can get a recipe from the internet and stir a spoon around in the dough. Ambient intelligence will also come into focus. The robot should be able to interact with house hold devices and can function as the ultimate interface.
The grasping of unknown object is a task that people in the league are already working on. In 2009 we plan to have the robots go shopping in a real supermarket (not an artificial scenario). Humans can have simple conversations with robots by use of natural language and gestures. One idea is to actually involve a developmental psychologist who assesses the robot to get an estimation of the age it would have were it a human. An example is occlusion; it takes a few years before the human infant realizes that an object is not gone when it is out of sight. Quite often the robotic systems are not much better than toddlers. Since it is measurable, we will probably introduce it. It requires natural interaction and would give an indication of the amount of human intelligence it possesses. Perhaps this could be a 21st century version of the all-too-famous Turing test.

6. Conclusions

RoboCup@Home is a league that has its foundations in robotic soccer. Technologies developed in soccer are being transferred to the @Home domain and on top op these technologies @Home might be able to give some more back. To reach the goal of winning in 2050 it will be necessary to use technologies developed in the @Home league. Since @Home focuses on general artificial intelligence in real world scenarios, theories will probably be developed that are not being developed in the soccer scenarios. Especially if the brain, body and environment are considered to be a holistic system where interactions go in all directions, the environment should also be paid attention to. Therefore the soccer leagues might also want to experiment with more than the artificial cues that are available in a soccer game.

Breazeal, C. L.(2002). *Designing sociable robots*, The MIT Press, ISBN 978-0262025102, Cambridge, USA, MA

Dautenhahn, K. (2007). Socially intelligent robots : dimensions of human-robot interaction, *Philosohical Trans. R. Soc. B(2007) 362,* pp. 679-704

Dunbar, R. I. M. (2003). The social brain : mind, language and society in evolutionary perspective. *Annu. Rev. Anthropol. 32,* 163-181

Harvey, I. (2000). *Evolutionary Robotics : From Intelligent Robots to Artificial Life,* Vol. III, Gomi, T. (Ed.), AAI Books, Ontario, Canada, 2000. pp. 207-230. ISBN 0-9698872-3-X.

Kitano H., Asada M., Kuniyoshi Y., Noda I., and Osawa E. (1995). RoboCup: The Robot World Cup Initiative, *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Alife*, Springer, Berlin

MacDorman, K. F. & Ishiguro H. (2006). The uncanny advantage of using androids in cognitive and social science research, *Interaction Studies 7:3*, 297-337, ISSN 1572-0373

Pfeifer, R. & Scheier C. (1999). *Understanding Intelligence,* The MIT Press, ISBN 0-262-16181-8, Cambridge, Massachusetts

Schiffer, S., Ferrein, A., Lakemayer, G.(2006). Football is coming Home, *Proceedings of the International Symposium of Practical Cognitive Agents and Robots*, November 2006, Australia

Van der Zant, T. & Wisspeintner, T. (2005), RoboCup X : A proposal for a league where RoboCup goes real world, *Proceedings of the RoboCup 2005: Robot Soccer World Cup IX*. Osaka, July 2005, Springer, Berlin

27

VolksBot - A Construction Kit for Multi-purpose Robot Prototyping

Thomas Wisspeintner and Walter Nowak
Fraunhofer Institute IAIS, Fachhochschule Bonn-Rhein-Sieg
Germany

1. Introduction

The development of mobile robotic systems is a demanding task regarding its complexity, required resources and skills in multiple fields such as software development, artificial intelligence, mechanical design, electrical engineering, signal processing, sensor technology or control theory. This holds true particularly for soccer playing robots, where additional aspects like high dynamics, cooperation and high physical stress have to be dealt with. In robot competitions such as RoboCup, additional skills in the domains of team, project and knowledge management are of importance.

Having participated in RoboCup Middle Size League since 1998, Fraunhofer IAIS and FH Bonn-Rhein-Sieg have developed six generations of different mobile robot platforms. Like many other teams we faced several difficulties. These systems were often monolithic, highly integrated prototypes that took a long time to develop, they were high in costs and hard to maintain. Also robots tend to grow old quite quickly when used frequently, put under high physical stress or when the robot's hardware simply gets outdated after a few years. Fluctuation of people combined with long training times for new team members and loss of knowledge are other difficulties we frequently experienced.

In this chapter, we present methodologies to cope with such diverse difficulties by using modular, component-oriented design approaches for mobile robot prototyping. The approach should enable developers to focus on their specific domain, still being able to have a clear understanding of the entire system by help of different levels of abstraction and well defined interfaces to hardware and software modules. Furthermore, (re-)usability should be maximized by having well documented system components of manageable size.

A concrete implementation of such an approach will be presented by way of the VolksBot concept (Wisspeintner et al. 2005). This project was started in September 2002 with the goal to create a multi-purpose, cost-effective and robust robot construction kit for advanced research, education as well as for application oriented prototyping.

Originally being applied to indoor scenarios like RoboCup Middle Size, since 2004 the concept was extended to fulfil the demands of more real-life applications like outdoor use, higher payload, velocity or scalability in morphology and hardware configuration of the platform.

Component-based prototyping concepts have been applied successfully in developing robots mainly for indoor applications or in the field of education some of them using a construction kit. The advantages of using a kit are quite obvious as this usually reduces development time and costs by fostering reuse of existing components. On the other hand, universal modules are not specialized, thus one loses in performance. There is always a trade off between the general applicability and the performance in modular approaches.
Significant work has been done in the field of rapid prototyping of robots in the past. (Won et al., 2000) have shown that rapid prototyping is a viable method to create articulated structures of robotic systems. (Reshko et al., 2002) have illustrated methods to quickly produce prototypes of desired quality in considerably small time by using ready-made components such as servo motors, sensors, plastics parts and Lego blocks.
Examples for robot construction kits mainly used for education and edutainment are Lego Mindstorms (Ferrari et al., 2001), Fischertechnik Mobile Robots (Fischertechnik), Tetrixx (Enderle et al., 2000) or the CubeSystem (Birk, 2004). Though aspects of modularity are addressed well by these systems, they are limited in onboard computational power and focus on miniaturization and low-cost hardware. As a consequence the aspect of application-oriented rapid prototyping of fully autonomous robots is hardly provided in these approaches and on-board perception is limited. On the other side, several robot platforms with higher complexity in sensors, actuators and higher processing power are usually specialized for a certain field of application or a certain scenario (Evolution Robotics) (K-Team) (ActiveMedia). Besides, many of such systems are specific in their morphology, their mechanics and hardware does not follow a construction kit approach. One exception is presented in the MoRob project with a focus on educational robotics (Gerecke et al., 2004).
This chapter structures as follows. After having motivated the use of such component-based construction-kit approaches, we summarize the resulting design goals. Then we derive concrete design criteria from these goals and show the interrelation between them. In the next section we show how applying these criteria have lead to a component pool in hardware, software and mechanics forming the VolksBot robot construction kit. In the following we illustrate how we used these components for prototyping of various robot platforms for different applications ranging from robot soccer to autonomous transportation, robot rescue and service robotics. The chapter ends with a conclusion and an outlook on future work.

2. Modular Design

With the aim to develop an approach for multi-purpose robot prototyping, several design goals in hardware and software can be defined which are illustrated in the following. The goals are labeled in brackets (G1-G12) for later reference.
One of the major goals is to reduce costs, time and resources needed to conduct mobile robotic projects (G1). This should motivate more research groups from various backgrounds to start or continue activities related to mobile robotics in education and research. Also it should help to generate interest and open the market for new robot applications with more companies being willing to invest in robot technology and prototyping projects.
The complexity of recent robotic systems grows constantly with the complexity of the applications they are designed for. Modern mobile robots usually require a variety of

different sensors, actuators and controllers but also algorithms and methods for signal processing, sensor data fusion, planning, localization, navigation and control of the robot, especially when being used in real world environments. The approach therefore should support the developers to manage this constantly growing system complexity (G2).
The system should allow the exchange and reuse of existing components in hardware and software (G3). For example it should not be necessary to start a system development from scratch every time a new robot needs to be built.
Also, an already existing robot platform should be easily reconfigurable and extendable by use of these hardware and software components (G4).
Reconfiguration and maintenance of the platform should be efficient and should not require special tools or machinery (G5). This way, developers are independent from having access to special facilities and experts and have more time to spend on research and development.
Often, groups already have worked in the domain of mobile robotics in the past. Therefore they should be able to efficiently integrate already existing technology into the system (G6).
The approach should help to foster the exchange and distribution of knowledge (G7). The design of such systems usually requires the interplay of many different individual skills which are distributed over a group or multiple groups of people.
The mechanics of the kit should be robust and scalable and allow for high payloads and high dynamics (G8).
To offer a wide range of possible applications, the kit should allow for diverse robot variants for different scenarios (G9).
The training periods for new users should be short (G10), so that they can produce results more quickly.
Synergies should be achieved by help of standardization (G11). Setting standards in mobile robotics projects to foster synergy between different research groups is an active research topic (OMG). Recently, Microsoft has introduced Robotics Studio (Microsoft) to foster exchange and synergy on the software level. The RoboCup 4-legged league has given excellent examples how using a standardized platform in combination with consequent code sharing can accelerate research and development in this domain.

Here is a summarization of the design goals:

G1. Reduce costs, time and resources in mobile robotics projects
G2. Be able to manage system complexity
G3. Allow exchange and reuse of existing components
G4. Allow easy reconfiguration and extension of the systems
G5. Allow simple and efficient maintenance
G6. Allow efficient integration of existing technology
G7. Foster exchange of knowledge
G8. Robust and scalable mechanical design
G9. Allow for a wide range of robot variants and applications
G10. Allow for short training periods of new users
G11. Achieve synergies by standardization

From these goals, we derived various design criteria for the construction kit in hardware, software and mechanics.

To reduce the costs and efforts for manufacturing and design of special components, standardized, available industrial components should be used if applicable (D1).
To keep the system complexity low and to be able to maintain the construction kit, the amount of components should be kept minimal, yet offering a high grade of reconfigurability. (D2)
Components should posses a fine granularity and should be universal to ensure reuse (D3).
A comprehensive mechanical component library should be built up using standard CAD software tools (D4). Before actually building the robot, a complete design and simulation should be done in CAD avoiding major design errors and allowing fast iterations during the design phase.
The same holds true for software development, where a software library should be built up using state-of-the-art software development standards regarding architecture, documentation and coding conventions (D5).
Besides development of own software, existing software and frameworks should be used and integrated into the approach (D6).
When developing a component in hardware or software, documentation standards for developers and users should be applied (D7).
Different layers of abstraction should be provided during system integration and development in hardware and software (D8). This should help to reduce training times and allow a wide range of people from different technical background to work with the system.
Clear interface definitions for hardware and software components have to be defined and maintained (D9).
Furthermore to keep the number of possible variants high and the system complexity low, dependencies between components should be avoided (D10).

Here is a summarization of the design criteria:

D1. Extensive use of standardized, industrial components
D2. Small number of different components with high reconfigurability
D3. Fine granularity of modules to ensure reuse
D4. Build up mechanical component library in CAD
D5. Build up software library with documentation and coding standards
D6. Use and integrate existing software and frameworks
D7. Apply documentation standards for components
D8. Introduce multiple abstraction layers
D9. Clear interface definitions for hardware and software components
D10. Avoid dependencies between components

The following table gives an overview of the relations between the design goals and the design criteria mentioned above:

	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10
G1	X	X	X	X	X	X				X
G2		X						X	X	X
G3	X	X	X		X	X			X	X
G4	X	X	X	X	X			X	X	X
G5	X	X		X			X			X
G6	X					X			X	X
G7					X	X	X	X	X	X
G8	X		X							
G9			X							X
G10		X			X	X	X	X	X	X
G11	X					X				

Table 1. Relations between design goals and design criteria

3. Component Pool

Having applied the design criteria mentioned in the previous section, a set of standard components in mechanics, hardware and software has been developed to form the VolksBot construction kit. Fig. 1 shows how different components were assembled to form the first version of a VolksBot Indoor robot. Here, a differential drive unit, a catadioptric vision system, batteries, a control notebook and a motor controller are mounted on the central base frame consisting of X-beams. A modular software framework is used for the robot control.

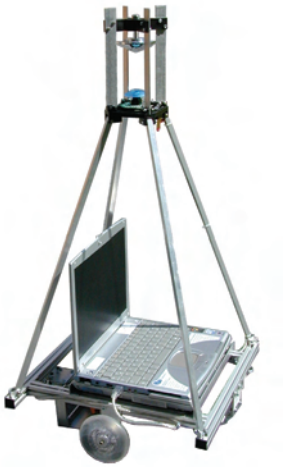


Fig.1. The first VolksBot Indoor variant

3.1 Mechanics

All mechanical components are modeled in SolidWorks (SolidWorks), a commercial CAD software tool, building up a common component library in CAD. This library helps to reduce mechanical design efforts enormously as new robot variants are designed mostly by recombination and adaptation of existing parts.

3.1.1 Frame elements

In accordance to the design criteria listed in the last chapter, we decided to use standard aluminum machine construction extrusions (X-beams) of 20mm width and proper connectors to build up the robot's main frame. They provide high rigidity, are light weight and offer a variety of different connections.
Size and shape of the robot´s main frame can be adapted individually to the needs by simple mechanical processing, i.e. cutting and screwing. By using pluggable t-nut-connectors, it is possible to establish new connections without having to decompose the frame. All sides of the X-beams can be used to connect to additional elements. In our design, all hardware components are connected to the main-frame. Therefore only geometrical dependencies between the component and the main-frame occur, not between the components themselves. All components like batteries, motor controller, drive units and sensors are connected to a rectangular single layered main frame. With this, the repositioning of components and scaling of the platform can be easily done.

3.1.2 Drive system

Fig.2 illustrates the frame construction for the first VolksBot Indoor versions with differential and holonomic drive.
The differential drive unit consists of a 20W DC motor, a claw coupling and a bearing block. The holonomic drive consists of three units which can directly replace the differential drive without any further modification of the robot. The unit itself is built up the same way as the differential drive, except for using stronger motors for higher speed and acceleration and using "Cat-Trak" Transwheels allowing a movement in X, Y and φ. A triangular aluminum adapter block is used to attach the two front drive-units to the main frame by simple screw connection, providing an angle of 120 degrees between the wheel axis. The third wheel is being directly attached to the main frame.

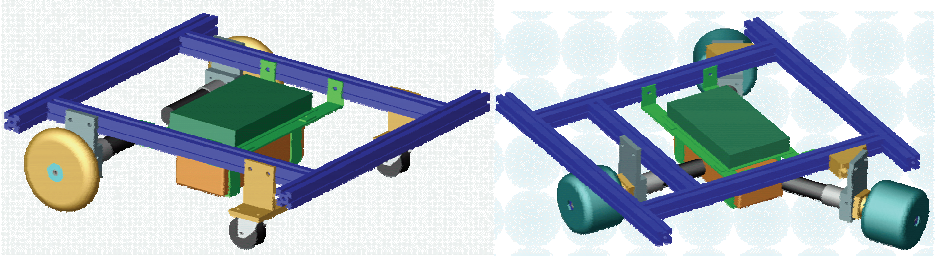


Fig.2. Main frames for differential (left) and holonomic drive (right) of VolksBot Indoor

3.1.3 Universal Drive Unit

The indoor base platforms presented above use a single 40x40cm frame which allows a light-weight construction, but limits the kinds of possible applications. To account for increased rigidity, payload and larger size of the robots, we introduce a double layered main frame consisting of 2 parallel X-beams and a new Universal Drive Unit (UDU). This resulted in the development of VolksBot RT (cf. section 4.1)
The wheels on each side of the VolksBot RT robot are driven by a single 150W DC motor. The force transmission to each wheel is achieved by using the Universal Drive Units (UDU),

a component to establish flexible chain-drive systems. In Fig. 5 (left), a closer view with the wheels removed, shows that a simple repetition of these units in an assembly can create drive systems with a varying number of actuated wheels.
The UDUs (Fig. 5 right) can be mounted with screw connection at various positions along the double layered base frame, which makes it possible to customize the wheel distances easily. The steel shaft of 10mm diameter is supported by two aluminum bearing blocks which allow for payloads of up to 80kg. The shaft can be directly driven by a motor (direct drive) or driven indirectly by belt or chain via the attached sprockets. We use a chain driven system for compactness reasons. The entire drive unit can be encapsulated within the span of 20mm, i.e. the thickness of one X-beam. This allows a complete housing of the transmission.

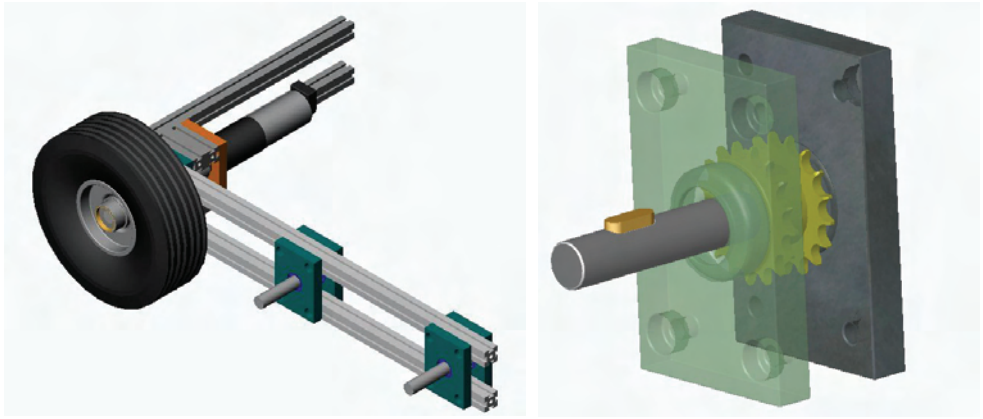


Fig.5. Double layered main frame with attached UDUs (left) and detailed view of UDU (right)

Furthermore, a set of air-filled tires with various diameter and profile provide proper mobility, ground clearance, grip and damping of the robot.
In order to drive the shaft, it is connected to a motor block through a claw coupling. Once assembled, it can be used to either drive other UDUs via chain transmission or drive the wheels directly. In case of different motors only the motor block needs to be modified. A chain transmits the possibly high torque from the 90W or 150W DC-motor which can be equipped with a planetary gear of various ratio ranging from 1:14 to 1:150. It can transmit forces of up to 3000N. The connection of the various tires, ranging from 18 to 40cm diameter, to the shaft requires only one standard hub connector.

3.2 Hardware Components

Also in hardware, a component library has been built up and is constantly being enlarged. The library holds commercially available products as well as own developments. Already integrated commercial components include (D)GPS, laser range finder, inertia sensors, industrial and barebone PC´s, compass, stereo and regular cameras and manipulators.
After the selection of new hardware, the component is tested and interfaces are defined. Providing a demo application and documentation for each component allows efficient testing and integration when a new platform is being built.

In the following an overview of hardware components developed for the construction kit is given.

3.2.1 Motor controllers

A specially designed motor controller TMC200 is connected via serial interface to the control PC. The controller offers odometric data analysis, thermal motor protection, battery voltage monitoring, velocity and current PID control for three DC-motors up to 200 Watts power.
In 2006 a new version called VMC was developed with improved properties in hardware I/O and thermal dissipation. It can be easily exchanged with the old component, and neither mechanics nor software require any structural modifications.
The component integrates also well in other robot platforms. For example, three of the four most successful teams in the RoboCup Middlesize League in the World Championships 2006 in Bremen used it in diverse robots.

3.2.2 AISVision

As a requirement for the soccer robots used in RoboCup Middle Size League a catadioptric vision system AISVision was developed. The vision system includes an IEEE1394 CCD camera and a hyperbolic mirror as shown in Figure 6 (left). Before construction, the system was designed entirely in simulation using the ray-tracing software POV-Ray (POV-Ray). In an iterative process, all relevant geometry parameters of the system were optimized for the use on a RoboCup Middle Size field. These include height of the mirror with respect to the camera, height of the entire vision-system above the ground, diameter of the mirror, focal distance of the camera and especially the two parameters a and b of the mirrors hyperbolic surface equation (1) with r being the radius and z the dimension along the optical axis.

$$\frac{z^2}{a} - \frac{r^2}{b} = 1 \tag{1}$$

The criteria for this optimization were full visibility of all landmarks from any position in the field, including goals and corner-posts, and a good visibility of the close region. The rendered and the real camera image are depicted in Fig. 6 (center, right). The optimization can be repeated for any other scenario with the described method. Different cameras have been already integrated, ranging from cheap webcams to sophisticated industrial cameras.

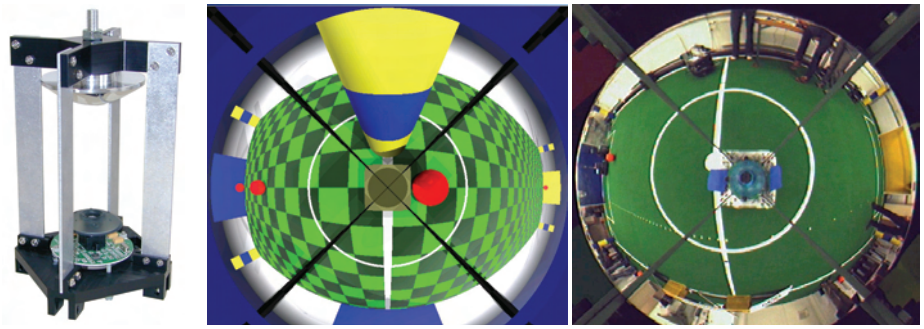


Fig.6. AISVision system (left), rendered (center) and real camera image (right)

3.2.3 3D Laser Scanner

A new continuously rotating 3D laser scanner was developed by extending the concept of the existing tilting 3D scanner (Surmann et al., 2001). Mounted on the robot, it is used for 3D mapping, navigation and localization in particular being useful when applied to outdoor scenarios. Two industrial laser range finders rotate around the vertical axis of the system, acquiring depth and intensity information for a 360° field of view.

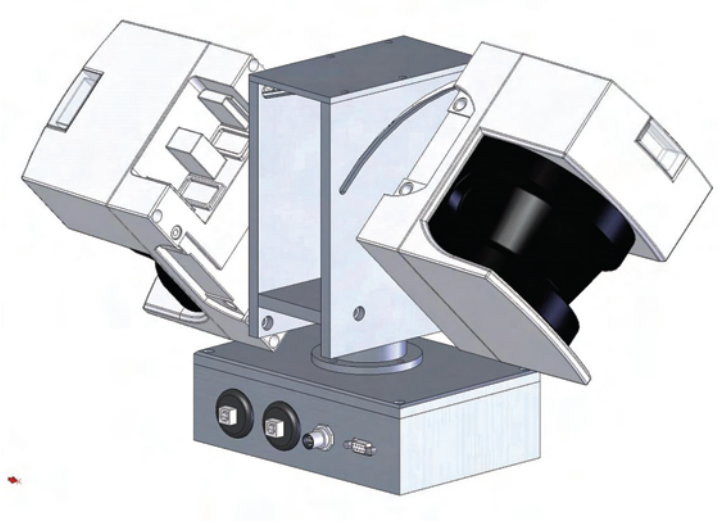


Fig. 7. CAD drawing of the new 3D Scanner

The system consists of 2 SICK LMS 291-S05 laser range finders mounted on an angular adjuster plate. The scanners have an apex angle of 180 and a resolution from 1 to 0.25. The maximum range of the scanners is 80 meters. Being able to adjust the angle of the scan planes allows us to increase the scan resolution or to increase the rotation speed while cutting off the irrelevant top and the lower part of the scanned sphere. The system is equipped with an RS232 and a CAN interface, inputs for the hall sensors and some general purpose digital inputs. Contact rings are used to power the laser scanners and to transmit

the sensor data via RS485 from the scanners. The entire system is IP65 water resistant, weighs 13kg and has a size of 90×330×250mm. The scan resolution depends on the rotation speed and the angular adjustment of the scanners. At an angular adjustment of 60 and a rotation at 0.45Hz we obtain a vertical resolution of 0.5 and a horizontal resolution of 1.7 with an update frequency of 0.9Hz. A separate control PC is used for post processing of the laser scanner data.

3.2.6 Kicking device

For participating in RoboCup, a new simple but effective pneumatic device for lift-kicking a ball was designed and built up within 2 days. The kicking device mainly consists of two X-beams, the same kind already used for the main-frame of the VolksBot. The X-beams form a vertical, adjustable lever mechanism, actuated by a pneumatic cylinder. By adjusting the cylinder-lever attachment, the parabolic path of the kicked ball can be tuned. Having an initial angle of 45° and speed of 7 m/s, a flight over a distance of 5 m can be reached.
In spite of the short development time, this component has been in use with only minor modifications for years now. A structural similar, but bigger variant has been built for the goalkeeper robot.

3.3 Component based Software Design

As valid for mechanics and hardware components, also in software, a clear framework concept with well-defined components is being used. We use ICONNECT (Sicheneder et al., 1998) as framework for the visual composition of signal flow graphs. A main advantage of ICONNECT compared to similar approaches (The MathWorks, Inc.) (Kalman, 1995) is a unique feature that allows to execute module graphs on a PC in real-time without recompilation of the whole module graph and without extra hardware needed. A module in ICONNECT consists of a compiled DLL and has a visual black-box representation with input and output pins in the graph editor. For each module, relevant parameters can be entered in a parameter dialog or can be changed during run-time via separate input pins. In Fig. 3 the ICONNECT programming environment is depicted, including an example of an easy to build graphical user interface.

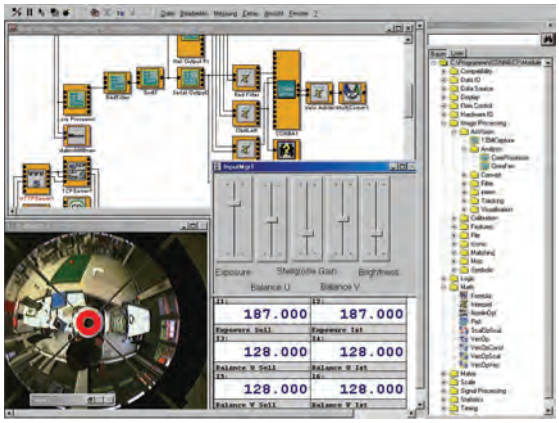


Fig. 8. ICONNECT programming environment with graph, GUI and module library

The existing module library of ICONNECT already contains a lot of functionality needed for controlling a mobile robot including signal processing of sensor data, image processing, control, hardware IO, logic, neural networks, network communication, data visualization and GUI design.
Except from recombining already existing modules, own modules can be written in C++ by use of existing module templates which can be filled with user-specific code. With this method, we extended the ICONNECT framework and added more robot-specific software modules to the library such as: the color vision library AISVision, integration of OpenCV (OpenCV), an interface to Matlab (The MathWorks, Inc.), a corba server (OMG), a module for integration of Dual Dynamics behaviors (Jäger & Christaller, 1997) or a simulator module based on ODE (ODE). For each new module, a HTML documentation has to be written, and a compact, self-explaining example graph has to be built. Combined with the aim to build only modules of fine granular functionality, this fosters reuse and reduces work-in time when new team members enter the project. With this approach, loss of knowledge is being reduced and even people with background other than computer science are able to program the robot on this abstraction level. This approach is especially beneficial for projects with many people of different background working together, like e.g. in RoboCup – as compatibility between modules is ensured by this clear interface definition and all people can have a clear and quite intuitive understanding of the entire robot control software.

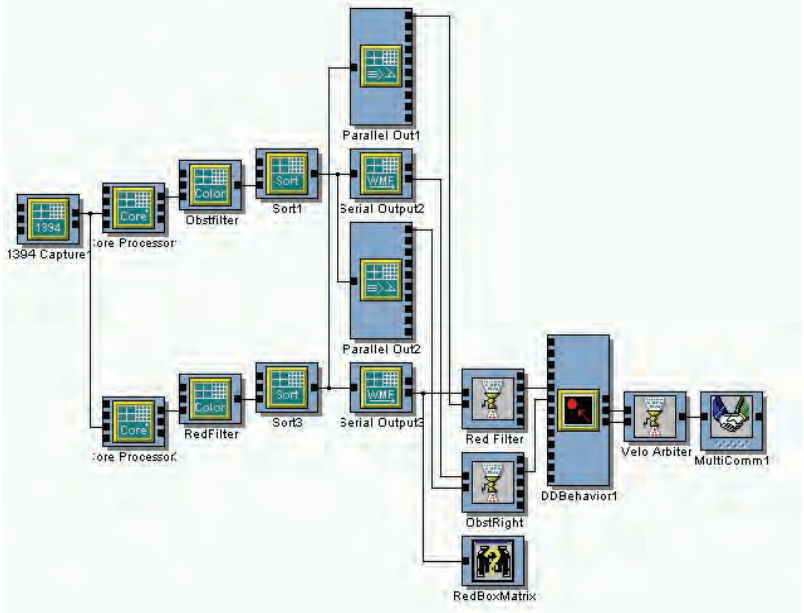


Fig. 9. A robot control graph in ICONNECT including data acquisition, signal processing, behavior and motor control

Combined with the advantages mentioned above, the use of such a framework sets some restrictions to the developer as it limits the choice of possible methodologies when developing software. Another aspect is the use of a particular operating system, such as Windows, which is required for the ICONNECT framework. An approach to overcome these limitations is the use of a more general software library which does not provide all the advantages of a differentiated framework like ICONNECT but may be appealing to a wider group of users.
This way, we started to standardize our inhouse software developments by building a new C++ software library called FAIRLib (Fraunhofer Autonomous Intelligent Robot Library). Here various algorithms and methods in the context of robot navigation, localization mapping, sensor processing and also hardware I/O are being integrated to make them more interchangable and usable.
The FAIRLib offers multiple abstraction levels with the low level consisting of basic I/O functions, data types, and a math library. Here, dependencies on specific hardware and operating systems are tolerated as they are unavoidable. The higher levels contain more sophisticated algorithms. Here the described dependencies can and should be avoided which allows to use the same set of methods for different hardware and operating systems.
The integration of the FAIRLib with the ICONNECT framework was another demand which has been met. Now, before programming an ICONNECT module, the functionality is first implemented into the FAIRLib which ensures even more universal use and reuse by a larger group of users.

4 Application oriented prototyping

Using the described component pools in hardware, software and mechanics various robot platforms have been designed for different scenarios. To show the feasibility of the construction kit approach, some of them are described in the following.

4.1 VolksBot variants

After the development of the first VolksBot Indoor variant (Fig. 1) mentioned in chapter 3, the idea had risen to extend the VolksBot concept to fit to the needs of rough terrain environments and real life applications. As a consequence new demands have to be set for the system including high payload, mobility, ground clearance and rigidity.
The recombination of the UDU and the double layered main frame enabled us to build 3 different RT (Rough Terrain) variants in a very short amount of time. This is due to the fact that we mainly reused existing VolksBot components combined with available standard parts. Only four different parts had to be machined to build up the Universal Drive Unit. Equipped with two 150 watt DC-Motors the 6-wheeled variant (Fig. 10) is able to climb a slope of 43 degrees and has a maximum speed of 1.3m/s. As the motor gears can be exchanged as easily as for the indoor version, the maximum speed can be adjusted according to the demands. So with little effort different variants in size and wheel configuration of the VolksBot RT can be built. The 3-wheeled, 4-wheeled and 6-wheeled variants are depicted in Fig. 10.

Fig.10. 3-wheeled, 4-wheeled and 6-wheeled variant of VolksBot RT

After the RT development, the original indoor version has been redesigned to take advantage of the improved rigidity and payload of the RT as well as to maintain compatibility between the systems. The redesign included the use of the double-layered frame and the UDU. 90W DC motors and larger wheels with 180mm diameter were integrated. With its compact size, and high payload this platform allows for various indoor tasks as e.g. the use in the RoboCup@Home domain (see section 4.2) or the use as an educational platform for mechatronics in vocational schools called ProfiBot (ProfiBot).

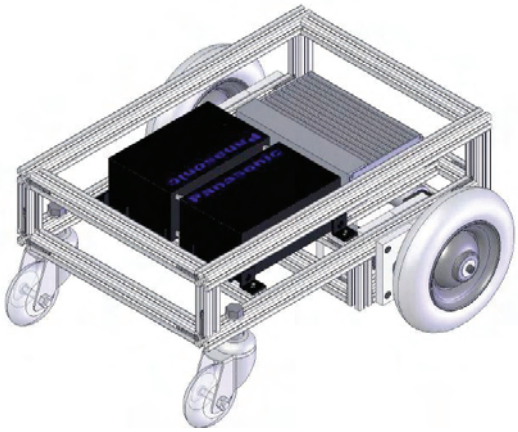


Fig.11. New indoor variant on the basis of RT

All previously described platforms still have limitations with respect to mobility due to their fixed body structure. An interesting approach to overcome these limitations with a wheeled robot platform is shown by the Shrimp Rover from EPFL (Estier et al., 2000) which uses a parallel bogey mechanism to passively adapt the wheel position on uneven terrain. We remodeled and simplified the parallel bogey as a component compatible to the RT kit. The direct drive for the wheels was replaced by a strand of a central hinge unit mounted to an upper and lower horizontal lever unit connecting two vertical leg units. Legs and levers build a parallelogram. Levers, hinge and legs are double bared such that they can accommodate an inlying chained transmission line. All in all we use four chain drives and eight UDUs (see Fig. 12 left). The new component offers various advantages compared to the original design. It only consists of standardized parts and easily allows for variation and expansion of the design.

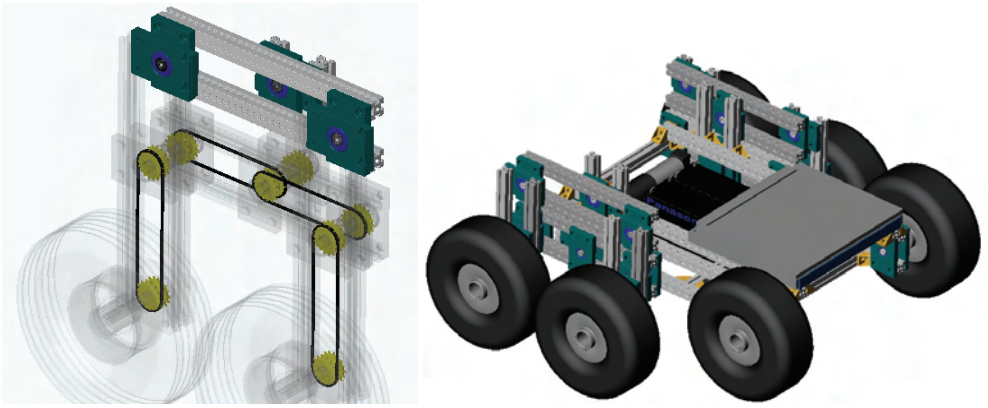


Fig.12. CAD drawing of the parallel bogey component (left) and the complete assembly of VolksBot XT (right)

The indirect chain drive system uses only two actuators and hence can be easily scaled according to motor power and payload. Furthermore, the entire drive system including the motors can be encapsulated and hence protected inside the robots frame. We have modeled an enhanced 6-wheeled variant which we called XT (Fig. 12 right), using the new parallel bogey component and tested its performance in ODE, a physical simulator. An increase of the mobility performance was achieved by use of a genetic algorithm which optimized a set of the robots geometrical parameters in ODE (Wisspeintner et al., 2006). Fig. 13 illustrates the result of this optimization with the robot being able to climb different kinds of stairs and moving over a random step field as used in RoboCup Rescue.

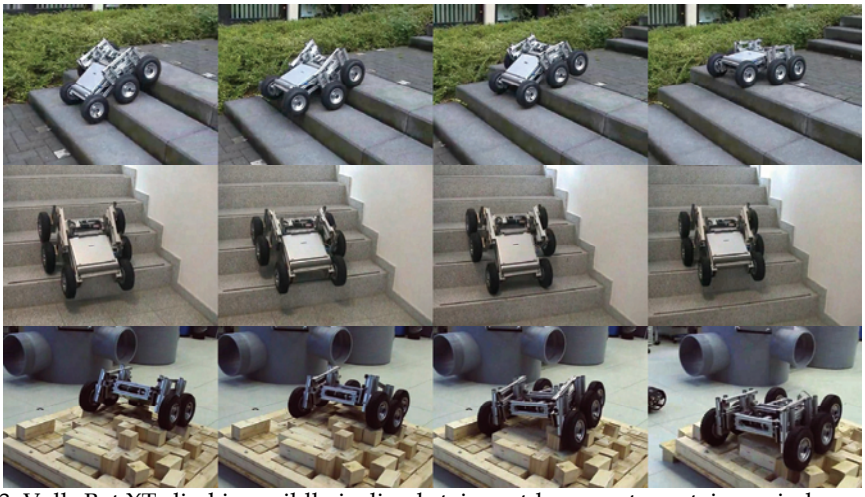


Fig. 13. VolksBot XT climbing mildly inclined stairs outdoors, a steep staircase indoors and moving over a random step field

4.2 RoboCup

Various development projects with VolksBot have been conducted in the context of the RoboCup competitions since 2004. We as well as other teams are using different VolksBot platforms and components to participate in the MiddleSize, Rescue and @Home league. Some of these platforms are presented in the following.

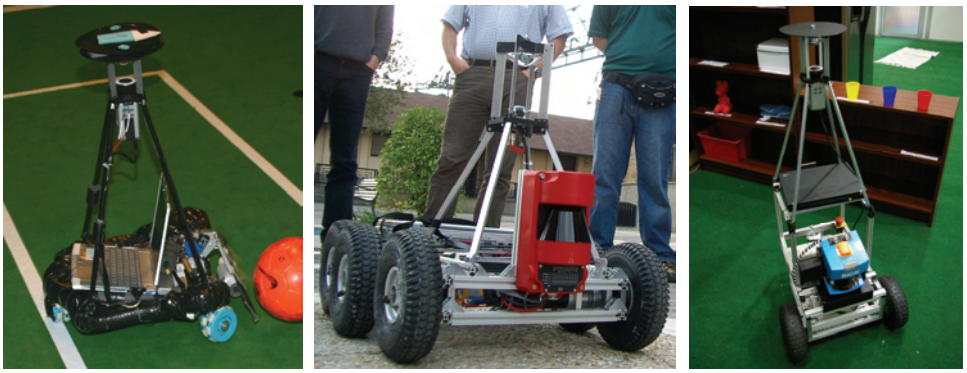


Fig. 14. VolksBots in RoboCup Middle Size (left), Rescue (center) and @Home league (right)

4.2.1 Middle Size League

In the beginning of 2004 an international student-team (AIS/BIT) using VolksBot was built up to participate in RoboCup Middle Size League (MSL). The main demands on MSL robots are quite different from other scenarios, requiring higher dynamics, superior motion control and real time color vision. To meet these demands the team had to introduce only a few additional plug-in components to the existing system. Participation in MSL demanded better image quality especially in high dynamic situations. Therefor the web-cams were replaced by Sony DFW-500 IEEE1394 cameras with very few adjustments.

All of these modifications in hardware required only minor software changes due to the component-based structure of ICONNECT. As each hardware component is directly related to one module, only the module itself had to be changed, without affecting the entire system. This also hold true for the behavior architecture itself, where we focused on the use of Dual Dynamics [14], an architecture based on dynamical systems. We extended the DD-Designer tool to directly generate ICONNECT modules, which made the behavior become an easily interchangeable component.

An important aspect of the development process is simulation. A module incorporating physical simulation of robots based on the ODE engine was developed. It has the same interfaces as the hardware, so the development of behaviors can be done without any special treatment, just by replacing the simulator with the corresponding hardware access modules in the graph.

Later developments included the integration the holonomic drive unit, the new motor controller VMC as well as a more compact frame. Further developments in the soccer domain include the design of an Outdoor soccer variant (Fraunhofer IAIS).

4.2.2 Rescue

The 6-wheeled version of VolksBot RT was used as base platform at the RoboCup Rescue Workshop 2004 in Rome. There, within 15 hours of lab-activities, two groups of three and six persons - with no prior experience of the system - worked together to build a functional rescue robot with autonomous behavior which has been demonstrated at the end of the workshop.

The task of one group was to build up the entire control system on the robot including, signal processing of laser-scanner data, image-processing, compression and WLAN transmission of the AISVision image stream, interfaces for teleoperation and manual override, autonomous behavior and motor-control. An obstacle-avoidance method was modified to achieve the desired (semi-)autonomous behaviors.

The task of the other group, dealing with human-robot interfaces, was to build an interface for the operator including visualization of the robots state, camera image and laser-scanner data. Further on it was required to set the robots state e.g. from manual to autonomous and build an interface to joystick and throttle for proper tele-operation.

The two groups worked together well, first defining the interfaces then testing the results iteratively. In summary it can be said that the VolksBot concept of rapid system development worked out resulting in a running system within very short time.

4.2.3 @Home

For the RoboCup World Championships 2007 in Atlanta a VolksBot robot was prepared to take part in the @Home league about 3 weeks before the tournament. Thanks to the modular design, it was possible to integrate two laptops and various sensors on the robot within the short time limit by a small team consisting of only two persons. With reuse of existing MSL and FAIRlib software, it was possible to participate in 3 different tests and the Open Challenge, almost reaching the finals. Fig. 15 shows the CAD design of a new @Home robot with integrated laser range finder, stereo camera, pan–tilt unit and a Katana arm from Neuronics. The robot construction is will be finished in autumn 2007 and will be used in future @Home competitions.

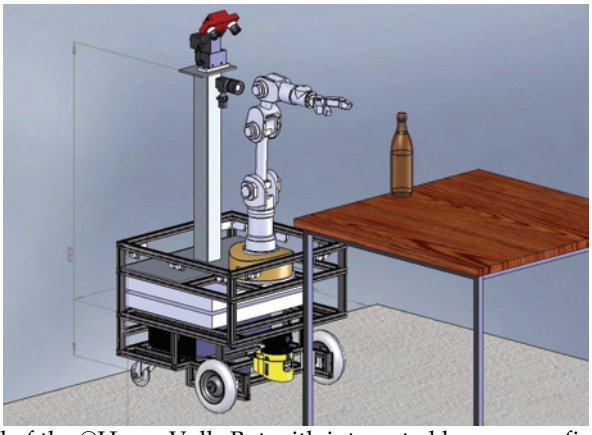


Fig.15. CAD model of the @Home VolksBot with integrated laser range finder, stereo camera and a Katana arm

4.3 Real World Applications

Besides application in RoboCup, the VolksBot construction kit has been applied to various other domains like autonomous transportation, surveillance or even underwater robotics.

4.3.1 PeopleMover

The PeopleMover is an extended version of the three-wheeled RT variant introduced in section 3. By selecting a high gear ratio, elongating the base frame, adding additional sensors and a few mechanical components, it is capable of autonomously transporting a person on a predefined track while avoiding obstacles. The reuse of software developed for the RoboCup Rescue workshop 2004 helped to build this prototype used for demonstration purposes within only 1 week.

Fig.15. PeopleMover, a prototype of an intelligent vehicle on VolksBot basis

4.3.2 MarBot

In cooperation with the Alfred-Wegener-Institute for Polar and Marine Research (Alfred Wegener Institut) we are developing an autonomous underwater robot for marine seabed analysis on the basis of the VolksBot kit. Instead of providing a complete housing for the robot's, only the robots sensitive hardware parts like the motor, the motor controller, the batteries or the control PC had to be shielded from the surrounding salt water. Besides the underwater environment the robot was designed for, various other demands had to be met regarding the design of the MarBot. Payload and size of the platform had to be increased to allow the installation of additional sensors and actuators like a mass spectrometer for advanced soil analysis which is mounted on a three-axis manipulator. Therefore an exchangeable center frame was designed carrying the additional hardware. Also the ground clearance had to be increased to 400mm to minimize the dispersion of sediments while driving. The resulting platform is illustrated in Fig. 16. It has six actuated wheels of 400mm diameter, a total size of 1200x700x650mm, a maximum speed of 1m/s and it weighs 30kg. The construction followed the design principles of the VolksBot RT series using the UDU with chain transmission. Only a few drive unit parts like the bearing blocks and bearings had to be replaced by plastic parts to avoid corrosion.

A Nano ITX barebone PC is used for the control of the robot. It can communicate to a base station via WLAN and UDP connection in shallow water allowing remote control and monitoring of the sensor data.
In software, both, a cockpit for the operator and the robot control software has been implemented in ICONNECT by use of the existing module library. Future development will include autonomous operation by use of multiple sensors like GPS, IMU, compass and vision allowing the robot to go from shallow water into depths of up to 30m.

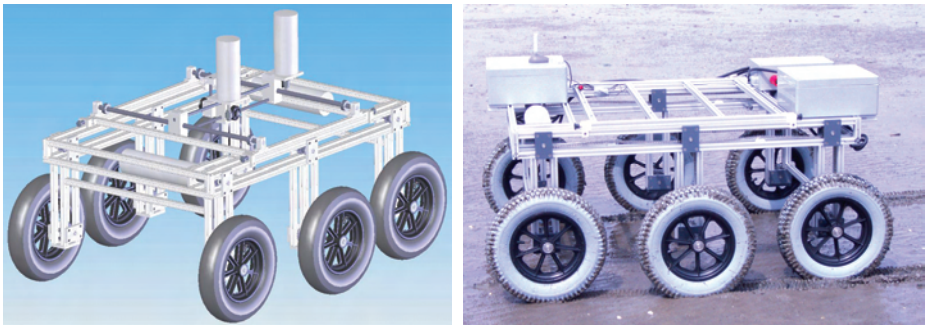


Fig. 16. CAD assembly (left) and image of MarBot (right), an underwater VolksBot variant

4.3.3 FuelBot

In cooperation with the Fraunhofer Institute ISE (Fraunhofer ISE) we developed a VolksBot variant which uses a fuel cell as central power supply. The fuel cell provides up to 400W power at 24 VDC. Depending on the mode of operation and the size of the metal hydride tanks filled with hydrogen it allows up to 24h of continuous operation. In this case, applying our prototyping concept allowed us to design the robot around the existing fuel cell, which is another good indicator for the flexibility of the concept. A VolksBot RT3 variant was used as the basis for this development. The robot then was equipped with a SICK Laserscanner, an industrial PC and a TFT display and was presented at the Hannover fair 2007 in Germany.

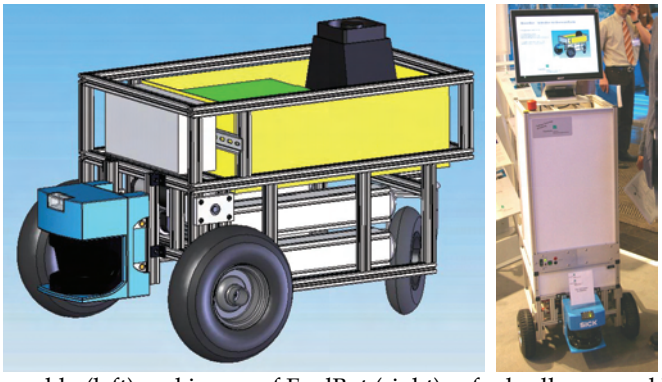


Fig. 17. CAD assembly (left) and image of FuelBot (right), a fuel cell powered VolksBot

6. Conclusion

In this chapter we have presented the a concept for application oriented prototyping of mobile robots. After defining design goals and deriving design criteria, we presented the mechanical, hardware and software components which followed these criteria and form the VolksBot construction kit. Then, variants of robot platforms for diverse applications were presented, including general platforms for indoor (VolksBot Indoor) and outdoor (VolksBot RT) use, a platform for high mobility applications (VolksBot XT), robots for participation in the Robocup Middle Size, Rescue and @Home league, a demonstrator for autonomous transportation (PeopleMover), an underwater variant (MarBot) and finally a fuel cell powered VolksBot (FuelBot).
Having successfully designed and constructed this number of robots indicates the feasibility and effectiveness of our design approach. Future work will include a constant enhancement of the construction kit allowing for even more applications. This includes the development of a modular, steerable drive unit, the use of tracks instead of wheels and a redesign of the XT variant for even higher mobility and payload. In software the FAIRlib will be further developed, including reusable modules for indoor and outdoor localization and navigation and various functionality in the @Home domain.

7. References

Birk, A. (2004). Fast Robot Prototyping with the CubeSystem. In *Proceedings of the IEEE International Conference on Robotics and Automation* (ICRA '04), New Orleans LA, USA, April 2004

Enderle, S.; Sablatnög, S.; Simon, S. & Kraetzschmar, G. (2000). Tetrixx: a Robot Development Kit. In *First International Workshop on Edutainment Robots*. T. Christaller, G. Indiveri, and A. Poigné, Eds.

Estier, T.; Piguet, R.; Eichhorn, R. & Siegwart, R. (2000). Shrimp, a Rover Architecture for Long Range Martian Mission. In *Proceedings of the Sixth ESA Workshop on Advanced Space Technologies for Robotics and Automation* (ASTRA '2000), The Netherlands, December 2000.

Ferrari, M.; Ferrari, G. & Hempel, R. (2001). *Building Robots With Lego Mindstorms : The Ultimate Tool for Mindstorms Maniacs*, 1st ed. Syngress, no. ISBN: 1928994679

Gerecke, U.; Hohmann, P. & Wagner, B. (2004). Concepts and Components for Robots in Higher Education. In *Proceedings of the World Automation Congress* (WAC'2004), Sevilla, Spain, July 2004.

Jäger, H. & Christaller, T. (1997). Dual Dynamics: Designing Behavior Systems for Autonomous Robots In *The Sixth International Symposium on Artificial Life and Robotics* (AROB 6th ´01)

Kalman, CJ. (1995): LabVIEW: a software system for data acquisition, data analysis, and instrument control. *Journal of Clinical Monitoring.* Vol 11 (1), pp 51–58

Mikhak, B.; Berg, R.; Martin, F.; Resnick, M. & Silverman, B. (2000). To Mindstorms and Beyond: Evolution of a Construction Kit for Magical Machines. In *Robots for Kids: Exploring New Technologies for Learning,* no. ISBN:1-55860-597-5

Reshko, G.; Mason, M. & Nourbakhsh, I. (2002). *Rapid Prototyping of Small Robots*. Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-02-11, March 2002.

Sicheneder, A.; Bender, A.; Fuchs, E.; Mandl, R. & Sick, B. (1998). A Framework for the Graphical Specification and Execution of Complex Signal Processing Applications. In *Proceedings of the ``1998 IEEE International Conference on Acoustics, Speech, and Signal Processing* (ICASSP '98) Seattle, May 1998

Surmann, H.; Lingemann, K.; Nüchter, A. & Hertzberg, J. (2001). A 3D laser range finder for autonomous mobile robots. In *Proceedings of the 32nd International Symposium on Robotics* (ISR '01), Seoul, Korea, April 2001.

Wisspeinter, T.; Bose, A. & Plöger, P. (2006). Robot Prototyping for Rough Terrain Applications and High Mobility with VolksBot RT. In *Proceedings of the International Workshop on Safety, Security and Rescue Robotics* (SSRR '06), Gaithersburg , Maryland , USA , August 2006

Wisspeintner, T.; Nowak, W. & Bredenfeld, A. (2005). VolksBot – A flexible component-based mobile robot system. In. *Proceedings of the RoboCup International Symposium,* Osaka, Japan, July 2005

Won, J.; DeLaurentis, K. & Mavroidis, C. (2000). Rapid Prototyping of Robotic Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation* (ICRA 2000), San Francisco, CA, April 2000

Internet Links

ActiveMedia, Pioneer. http://www.activrobots.com /ROBOTS/p2dx.html
Alfred Wegener Institut für Polar- und Meeresforschung: http://www.awi.de
Evolution Robotics ER1. http://www.evolution.com /er1
Fischertechnik. http://www.fischertechnik.de/english
Fraunhofer IAIS: VolksBot. http://www.volksbot.de
Fraunhofer IAIS: Project OUTDOOR: http://www.iais.fhg.de/602.html
Fraunhofer ISE: http://www.ise.fraunhofer.de
K-Team, Koala robot. http://www.kteam.com/robots/koala/index.html
Microsoft, Microsoft Robotics Studio. http://msdn.microsoft.com/robotics
ODE: http://www.ode.org
OMG, Robotics DTF. http://robotics.omg.org
OMG, CORBA: http://www.corba.org
OpenCV: http://sourceforge.net/projects/opencvlibrary
POV-Ray: http://www.povray.org
ProfiBot. http://www.profibot.de
SolidWorks. http://www.solidworks.com
The MathWorks, Inc.: www.mathworks.com/products/simulink

28

Collaborative Localization and Gait Optimization of SharPKUngfu Team

Qining Wang, Chunxia Rong, Guangming Xie and Long Wang
Intelligent Control Laboratory, College of Engineering, Peking University
China

1. Introduction

In this chapter, we introduce the recent progress of sharPKUngfu Team which participates in the RoboCup Four-Legged League since 2004. sharPKUngfu Team is a robot soccer team from Peking University, China. In July 2005, we got the third place in the RoboCup China Open. In June 2006, our sharPKUngfu Team has participated in the technical challenge of RoboCup 2006. In this event, our *Medal Awarding* challenge got the eighth place in the Open Challenge. In October 2006, we got the champion in the RoboCup China Open 2006, both in soccer competition and technical challenge. In July 2007, we participate in the RoboCup 2007 and got the fourth place in the technical challenge. Our research in robot soccer focuses on robot vision, multi-robot cooperation strategy, collaborative localization in dynamic environment, quadruped gaits optimization and intelligent behavior.

We focus this chapter on localization and gait optimization which are the fundamental parts in soccer robotics. Recently, we successfully apply self-learning image-retrieval approach and collaboration in self localization in robot soccer. This improvement eliminates the problems of image-retrieval method and collaboration mentioned in previous research. By using this approach, robots can play soccer under more natural conditions towards real human soccer environment. We organize the localization part as follows. At first, a brief overview of current self-localization approaches is presented. Secondly, we introduce the human cognition inspired localization with self-learning experience. Specific algorithms for image features collection and self-learning process are described. Then, the dynamic reference object based method for collaborative localization is demonstrated in detail. Experimental results in real robot soccer are shown in the end. We also discuss current challenges and future works of localization in soccer robotics.

How to get high-speed walking and running gaits is another problem in soccer robotics. Different to existing literature which uses Genetic Algorithms (GA) based gait optimization methods, we present the implementation of Particle Swarm Optimization (PSO) in generating high-speed gaits for a quadruped robot, specifically the Aibo, which is the commercial robot made in Sony. PSO has been proven to be effective in solving many global optimization problems and in some areas outperform many other optimization approaches including Genetic Algorithms. In this part, at first, we overview the basic PSO and Adaptive PSO (APSO) with comparison to other optimization approaches. After that, with the

knowledge of using higher lever parameters to represent the gait which focus on the stance of the body and the trajectories of the paw, the inverse kinematics model is explained. Moreover, the control parameters and optimization problem are proposed. In addition, how to implement PSO in the quadruped gaits learning is introduced in detail. The whole learning process is running automatically by the robot with onboard processor. In robot experiments, we achieved an effective gait faster than previous hand-tuned gaits, using Aibo as the test platform.
Our progress of intelligent behaviors in real soccer competition is described briefly in the end of the chapter. All the details about real robot experiments and how to use the debugging tools can be found in (Wang, 2006b) or the official website of sharPKUngfu team.

2. Multi-Robot Collaborative Localization

In soccer robotics, for example in the RoboCup, several probabilistic methods for global self-localization have been implemented in various teams from different leagues (eg. Fox et al., 2000; Röfer et al., 2003; Schmitt et al., 2002). However, most current localization approaches used in robot soccer depend on standard landmarks and static environment. On the move to real human soccer conditions, current localization approaches in robot soccer seem not enough. In the human soccer, there are two aspects which may inspire the self localization of mobile robot systems. On the one hand, the features surrounding the soccer field may be exploited as the sensory information in probabilistic approaches. Inspired by the features, some robot systems have applied image-retrieval approach in localization (Wolf et al., 2005; Wang et al., 2006a, 2006b). There are several limitations by using such image-retrieval method. First, the computational cost of this approach is expensive. Besides, the requirement of building a huge database is not so practical, especially in the complex environment. On the other hand, collaboration among the robot team, which is only used in the strategy modules of current robot soccer teams, may be considered as another part of the sensory information. Previous research in localization has proven that the cooperation in self-localization among multiple robots has impressive performance in real robot systems (see Arkin & Balch, 1998 for overview). The limitation of such robot systems is that the robot needs to identify other one precisely. It is quite difficult to perform collaborative localization for robots dealing with situations where they can detect but not identify other robots. In addition, taking the uncertainty of sensors into account, the result of detecting individual robot is not so reliable. Those limitations of the approach make it not so applicable for real robots localizing in complex environments.
To apply image-retrieval approach and collaboration in self localization in robot soccer, we focus our work on two aspects. In the image-retrieval system, an efficient method of calculating image features is implemented. To simulate real human soccer conditions, colourful advertisement is placed around the field which is similar to the real soccer field. Our method divides one image into several parts to calculate features respectively. To construct the image feature database, the robot learns the relationship between images and positions autonomously. This improvement eliminates the problems of image-retrieval method mentioned in previous research (Wolf et al., 2005). By using the efficient approach, robots can play soccer under more natural conditions towards real human soccer environment. In addition, to introduce collaboration among team members in localization module, we integrate the image-retrieval approach with collaboration. In real robot soccer, it

may not so easy to identify the specific robot who is nearby, especially in the dynamic environment of soccer competitions. In human soccer, players can localize in the field by the distance to ball and team members. Inspired by this technique, a dynamic reference object based method is implemented in the real robot competition. This collaborative approach can improve the self localization in the field with less artificial landmarks. Positive impact on localization through our approach is shown in experiments using the Sony Aibo ERS-7 robot.

2.1 Landmark & Experience Based Markov Localization

To improve the probabilistic approach, we created an efficient method to construct environment features as experience, which is collected by the robot autonomously. By using such experience, robot can localize in the field with less artificial landmarks towards real human soccer environment.

Most robot localization systems use landmarks as the tool to predict and correct current positions of mobile robots. For example, (Röfer et al., 2003) proposed and improved the landmark based Markov localization. In this approach, the current position of the robot is modelled as the density of a set of particles which are seen as the prediction of the location. Initially, at time t, each location l has a belief:

$$Bel_t(l) \leftarrow P(L_t^{(0)}) \tag{1}$$

To update the belief of robot possible location, at first, this approach uses the new odometry reading o_t:

$$Bel_t(l) \leftarrow \int P(l \mid o_t, l^-) Bel_t(l^-) dl^- \tag{2}$$

If robot receives new sensory information s_t, then it updates the belief with α being the normalizing constant:

$$Bel_t(l) \leftarrow \alpha P(s_t \mid l) Bel_t(l) \tag{3}$$

Considering the mobile robot with complex motions, let the geometric centre of robot body as the location vector ϕ, which contains the x/y- global coordinates of the centre point. Another vector θ is defined as the heading direction. Then every particle is updated by the motion model as follows when the robot moves:

$$\phi_t = \phi_{t-1} + \Delta_t \tag{4}$$

where Δ_t represents the displacement in x/y coordinates and heading direction.

To implement image retrieval system in Markov localization, we divide the sensory update into two parts: updating position probability by landmark perception and experience matching. If the robot recognizes landmarks well enough, landmark based sensor model will update the belief of position with the new landmark reading s_t:

$$Bel_t(\phi_t) \leftarrow \beta P(s_t \mid \phi_t) Bel_t(\phi_t) \tag{5}$$

where β is a normalizing constant. It is natural that the robot may miss some landmarks with real-time recognition for a period. Thus, we set $N_1(t)$ which is the amount of lasting frames of having no landmark perception from t as a condition to activate the experience system. If $N_1(t)$ is great enough, the experience based sensor model will update the probability as follows with e_t being the new reading experience with γ being the normalizing constant different from β:

$$Bel_t(\phi_t) \leftarrow \gamma P(e_t \mid \phi_t) Bel_t(\phi_t) \tag{6}$$

2.2 Experience Construction

The feature that is exploited from images with no landmark in the view, and represents the invariant character of images obtained at positions where collisions and other negative effects more likely occur is defined as *Experience*. In our system, we make the robot to collect the image features autonomously, which is named self-learning experience. The experience contains image features in divided areas and the whole image respectively. In the following paragraphs, we introduce our efficient method to construct experience in detail.

(a) Image Features in Divided Areas

In our method, we divide one image which is obtained by the robot camera into six parts. First, image features including average colour value $f_{i,j}$ and colour variance d_i in the divided areas are calculated by the following equations:

$$f_{i,j} = \frac{\sum_{x,y} M[y][j][x]}{N_i}; \{j = 0,1,2; i = 1,2,3,4,5,6\} \tag{7}$$

where $f_{i,j}$ is the average value in the colour channel j of the area i. $M[y][j][x]$ represents the value in the colour channel j at the position (x, y) in the image. N_i is the number of the pixels in area i. Clearly, $f_{i,j}$ is in the range from 0 to 255.

$$d_i = \frac{\sum_{x,y} (|M[y][0][x] - f_{i,0}| + |M[y][1][x] - f_{i,1}| + |M[y][2][x] - f_{i,2}|)}{N_i} \tag{8}$$

where i=1, 2, 3, 4, 5, 6. d_i is in the range from 0 to 382.5. When the value of colour variance in the certain area gets maximum, d_i is 382.5.

(b) Image Features in The Whole Image

After calculating features in divided areas, we collect average colour value F_j and colour variance D in the whole image which are calculated by the following equations:

$$F_j = \frac{\sum_i f_{i,j}}{S}; \{j = 0,1,2\} \tag{9}$$

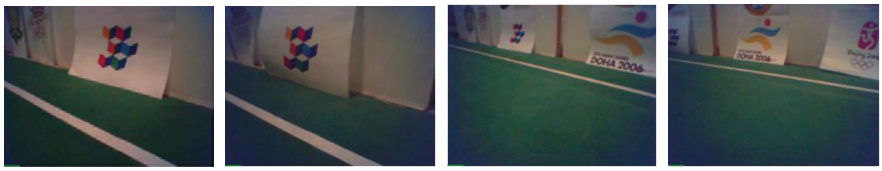
where F_j represents the average value in the colour channel j of the whole image. S is the number of divided areas in the image.

$$D = \frac{\sum_i (|f_{i,0} - F_0| + |f_{i,1} - F_1| + |f_{i,2} - F_2|)}{S} \tag{10}$$

where D is in the range from 0 to 382.5.

(c) Experience Construction

In our system, the invariant features of images includes $f_{i,j}$, d_i, F_j, and D. All the features are calculated from images collected in certain places where the robot needs experience to help. We construct experience database embedded in robot's memory. This database stores the feature along with the global coordinates of the position where the image is taken. All the features are calculated off-line and stored in the database as experience. When the experience module is activated, the feature of current image taken by camera is computed on-line notated as *imageFeature* which includes average colour value $Q_f_{i,j}$ and colour invariance Q_d_i in the divided areas, average colour value Q_F_j and colour invariance Q_D in the whole image. Meanwhile, the record notated as *bestRecord* whose feature is most similar to *imageFeature* is selected from the database. Fig. 1 shows the result of finding the best pose in database based on experience. The query image is on the left while its most similar image in the database is on the right. Their poses are represented by (x, y, θ). x, y are calculated in millimeter, while θ is in degree. Algorithm 1 presents how to calculate the difference *Diff* between the image for query and the image in database, where A_1, A_2, A_3, A_4, B, C_1, C_2 are control constants.



(a) (b) (c) (d)

Fig. 1. Examples for finding the best pose in image database. Images in the database are collected in the areas of the field where the robot can not see any landmark every 100mm in x, 100mm in y and 45° in θ. (a) is the current image taken by robot's camera when its real position is (−1660, 1520, 135°). (b) is the most similar picture to image (a) in the experience database which the corresponding position of the robot is (−1600, 1500, 135°). The location error is 60mm in x, 20mm in y, and 0° in θ. (c) is the random sample image taken after (a) when the real robot position is (−1040, 1220, 135°). The location error in experience image (d) is 240mm in x, 120mm in y, and 0° in θ

When the experience module is activated, difference between *imageFeature* and the feature of *bestRecord* is calculated. If the difference is small enough, the pose of *bestRecord* is transferred

into *bestPose* notated as l_{best} which is in the form of world coordinates in the robot system. With such *bestPose*, probabilities of all the sample poses are updated and new pose templates which are random poses near the *bestPose* are generated to perform the resample procedure in Markov localization. It is true that the more experience in database, the more precisely the calculation is. However, building such database is expensive in time cost and even unreachable in complex environments. As a part of the sensor update module, experience can help the Markov localization converge as soon as possible, which means the robot can know own position immediately. In our approach, we only need to construct the database in those really difficult situations. This method works well in real robot applications.

(d) Self Learning in Experience Collection

One of the difficulties in applying image-retrieval system into real robot localization is how to collect the experience efficiently and correctly. In our system, we create a self learning method for experience collection. The robot can collect images along with corresponding positions autonomously. When construct the experience database, we use the black-white stripes to adjust robot body which is similar to the one used in gait optimization mentioned in (Röfer, 2004). In the self learning procedure, at first, the robot adjusts its own body to the initial position which is preset by our control system. By using the stripes, the robot walks to the next position and stops to capture images in left and right view respectively as shown in Fig. 2. The black-white stripes help robot go to the preset position precisely.

(a) (b) (c)

Fig. 2. Self learning procedure in experience collection. (a) shows the Black-white stripes for body adjusting. The robot captures image in the left view and right view as shown in (b) and (c) respectively

Algorithm 1. Calculate the difference between the query image and the image in database

```
1: procedure Calculate the difference (query image, database)
2:    for all images in database do
3:       if  <A1 &&  <A2  then
4:          NumberOfAreasBeOK=0, Diff=0
5:          for  (i=1; i<S; i++)
6:             diff_f[i] =
7:             diff_d[i]=
8:             Diff+=C1*diff_f[i]+C2*diff_d[i]
9:             if  diff_f[i] < A3 && diff_d[i]< A4  then
10:               NumberOfAreasBeOK++
11:            end if
12:         end for
13:         if  NumberOfAreasBeOK > B then
```

```
14:             if  Diff < minDiff  then
15:                minDiff = Diff
16:                bestRecord = the current image in database
17:             end if
18:          end if
19:       end if
20:    end for
21: end procedure
```

2.3 Incorporating Experience in Markov Localization

In Markov localization, every sample pose has a belief which represents the probability of predicted position. In our approach, the sensor module updates the probability using the following equation:

$$p_i(t) = \prod_{j=1}^{K} q_i^{(j)}(t); i \in [1, S] \tag{11}$$

where K is the sum of sensor module types, while S is the number of all sample particles. Every $q_i^{(j)}(t)$ describes the position probability at time t using certain type of perception. Specifically, to incorporate the experience module in Markov localization, we set $q_i^{(j)}(t)$ as the quality for experience perception to every sample pose. The sum of the dimensionless distance and the dimensionless angle between *bestPose* and the sample pose is used as a criterion to update the quality with the fact that the quality is higher if the sample pose is nearer to *bestPose*. The experience quality of every sample pose is in the form of the following equation:

$$q_i^{(k)}(t) = \begin{cases} q_i^{(k)}(t-1) - \eta; v < q_i^{(k)}(t-1) - \eta \\ q_i^{(k)}(t-1) + \xi; v > q_i^{(k)}(t-1) + \xi \\ v; other. \end{cases} \tag{12}$$

where η and ξ are constants used for tuning quality not to change too fast. Thus, the quality can be controlled in a certain range. The criterion v is defined as follows:

$$v = e^{-(\sigma+\tau)^2} \tag{13}$$

Here σ is the dimensionless distance between *bestPose* and current sample pose, while τ is the dimensionless angle. Supposing that current sample pose is l_c (x_c, y_c, θ_c) and the *bestPose* is l_{best} (x_b, y_b, θ_b), then σ and τ are calculated in equations below:

$$\begin{aligned} \sigma &= \frac{\sqrt{(x_c - x_b)^2 + (y_c - y_b)^2}}{D_0} \\ \tau &= \frac{|\theta_c - \theta_b|}{A_0} \end{aligned} \tag{14}$$

where D_0 and A_0 are the constants which are used to control qualities of σ and τ. Normally, σ=0.05, τ=0.1. Moreover, we set sudden increases of both σ and τ in order to reduce greatly the qualities of the sample poses that are far away from *bestPose*. Using such method, the procedure of resample can be more effective and efficient. The useless particles can be eliminated as soon as possible. The time cost of the Markov localization convergence is relatively satisfied. Incorporating experience in Markov localization makes the probability update procedure more robust, especially when collisions or other negative effects occur.

2.4 Collaborative Localization

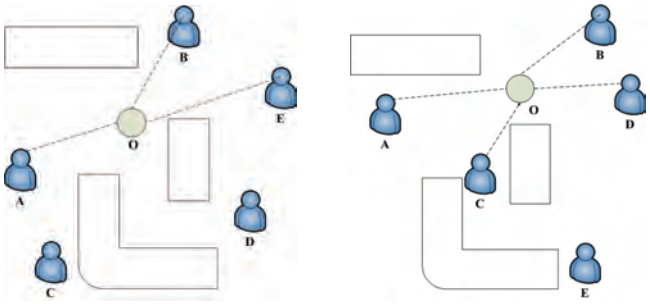
(a) The Notion Of Dynamic Reference Object

In RoboCup, static reference objects like beacon, and goal can be used to help localize in complex environments. However, global coordinates of such objects need to be known beforehand. Those static reference objects are not applicable in an unknown environment. To solve this problem, we propose the concept of *Dynamic Reference Object*. The object that can be detected by more than one robots among the team will be the candidate dynamic reference object. If the frequency of clearly recognizing the object is high enough, it may be set as the dynamic reference object. There is no need to know the object's position as a precondition. If a robot can localize itself accurately, the position of the dynamic reference object calculated by this robot is reliable. Meanwhile, another robot that has seen the reference object can use this calculated position of the object to measure own location. This information is useful for decreasing the time cost of Markov localization convergence and improve the result of position estimate especially for multiple robots collaboration.

There are several challenges to implement this approach in real robot systems. First of all, every robot that has detected the object will broadcast the calculated position to every other robot. Then the robot that needs help may be not able to figure out which position is correct. In addition, the result of the reference object position calculated by a robot may be wrong when another robot needs this information to measure own location. Time delay of the communication is another problem which may bring negative effect to the measurement. To solve problems mentioned above, with the assumption that robots can communicate with each other, our approach integrates *Reference Object Position Possibility* in the team message which will be broadcasted to every robot. The item which is relevant to the object position in team message includes calculated position, robot ID, time, and position possibility. This position possibility is due to the accuracy of the robot self localization. In our system, the object position possibility is notated as P_r is measured by the following equation:

$$P_r = e^{-\mu^2} P_l + e^{-\omega^2} P_e \tag{15}$$

respectively. μ is the sum of lasting frames after detecting the latest landmark, while ω is the sum of lasting frames after exploiting good experience. In real robot application, P_r will be normalized less than 1. If P_r is high enough, the calculated result by this robot will be the most reliable one among different robots perception. A robot that needs help always uses the most possible position of the reference object at the same time when it detects the object by itself. To illustrate the method, a common robot system is shown in Fig. 3 with five mobile robots. Object *O* is supposed to be the dynamic reference object. Table 1 is the real-time information in team message of the system in Fig. 3.



(a) (b)

Fig. 3. A simple system with five mobile robots and a dynamic reference object: (a) At time t_1, robot *A*, *B* and *E* can see the dynamic reference object *O*. They all use their own perception to calculate the position of the object and broadcast to every robot in the team. If at this time robot *A*, for example, needs the reference object to help, A will use the calculated position of the object from *B* or *E*. Querying the most possible position in team message shown in Table 1, *A* will take the calculated result by *B* as the reference. (b) At time t_2, *C* and *D* have not detected any landmark or experience for a period. Thus their answers to the object position are relatively unreliable. Position possibilities of them are shown to be low in Table 1. The reference object position will be set as *B* percepts.

Calculated Position	Robot ID	Time	Position Pssibility
(2388, 700)	A	t_1	0.71
(2264, 658)	B	t_1	0.92
(2530, 710)	E	t_1	0.86
(2368, 803)	A	t_2	0.81
(2401, 801)	B	t_2	0.91
(2103, 743)	C	t_2	0.32
(2215, 725)	D	t_2	0.43

Table 1. Team message relevant to dynamic reference object

(b) Multi-Robot Markov Localization

To illustrate how to integrate the dynamic reference object module in Markov localization, let us assume that robot *i* uses the reference object position calculated by robot *j*. Then robot *i* updates own position belief as follows with a normalizing constant ε :

$$Bel_t^{(i)}(\phi_t^{(i)}) \leftarrow \varepsilon Bel_t^{(i)}(\phi_t^{(i)}) P(\phi_t^{(i)} \mid r_t) Bel_t^{(j)}(\phi_t^{(j)}) \tag{16}$$

where r_t is the dynamic reference object position. The specific probability function using for collaborative approach is similar to the one in the experience model mentioned in equation (12).

In our approach, collaboration is a part of probability update modules in Markov localization. There is a problem that robots should known when to activate the collaboration

module using the dynamic object as a reference. To improve Markov localization using our collaborative approach, the collaboration module will be activated in two situations. We set $N_2(t)$ by using as the sum of lasting frames of having no landmark perception or experience as a condition to activate the collaboration system. If $N_2(t)$ is great enough and the robot has detected the dynamic reference object, the collaboration module will update the probability of every poses. In addition, if the robot has a perception of the object which has a relatively high position possibility, the robot will use this reference to improve the Markov localization in a collaborative way.

2.5 Real robot experiments

(a) Localization Environment

The experience-based collaborative approach presented above has been implemented on the Sony Aibo ERS7 legged robot in RoboCup environment. Fig. 4(a), (b) show the environment in 2006 and 2007 respectively. In our localization experiment field, we use the field similar to the standard field in four-legged soccer field 2007. However, we remove the beacons. As shown in Fig. 4(d), our field is surrounded by colorful advertisement which simulates the real human soccer environment.

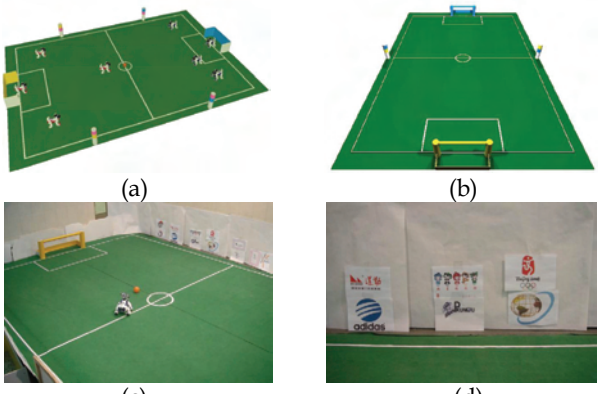


Fig. 4. Experimental field. (a) is the figure which shows the four-legged soccer field with four artificial beacons in 2006. (b) is the soccer field with two colorful beacons in 2007. (c) shows field with no beacon which is used to test our localization approach. (d) is the colorful advertisement placed around our test field which simulates the real human soccer environment

(b) Individual Robot Localization

Go to Certain Position: In the experiment, we use one four-legged robot to perform localization in our test environment shown in Fig.4 (c). Initially, the robot is placed at one center facing out of the field. Then the robot walks to a position with certain global coordinates and body facing angle. On the way to the destination, we pick the legged robot up for a while to effect the odometry in a negative way. This procedure makes the odometry not so reliable to imitate real dynamic environment in soccer competitions. In the experiment, the certain destination position is set to be $(-1450,-300,\ 0^{\circ})$. After 42 seconds,

the robot walks to the position (−1380,−350, 6°). The localization error is 70mm in x, 50mm in y, and 6° in body angle.

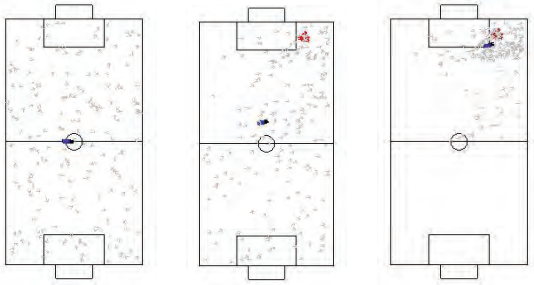
Randomly Walking: The robot is walking on the field with no beacon. We randomly select 8 points to test the self localization results. The robot is expected to go to the preset positions through localization. When it stops, we calculate the real positions on the ground. Table 2 shows the results in detail.

Point Number	Expected Postion (x, y, θ)	Real Postion (x, y, θ)	Error (x, y, θ)
1	(-1290, -440, 15)	(-1496, -713, 147)	(206, 273, 132)
2	(-1450, -300, 0)	(-1410, -150, 0)	(40, 150, 0)
3	(-180, -670, 45)	(-230, -610, 9)	(50, 60, 36)
4	(1430, -250, 55)	(-1909, -1162, 132)	(461, 912, 76)
5	(-650, 170, 0)	(-404, -427, 5)	(246, 597, 5)
6	(270, -480, -90)	(102, -402, -48)	(168, 78, 42)
7	(-1440, -340, 10)	(-1322, -332, 5)	(78, 8, 5)
8	(-2160, -390, 0)	(1979, -454, 8)	(181, 64, 8)

Table 2. Results of self localization in randomly walking. x, y are calculated in millimetre, while θ is in degree

(c) Collaborative Localization

In this experiment, the orange ball used in the four-legged league is considered as the dynamic reference object. We use three robots to perform multi-robot localization. Every robot uses the hybrid system tested in the individual experiment mentioned above. We set one of the three robots as a sample to estimate our collaborative approach. The other two robots move randomly to catch the ball and broadcast the ball position with position possibilities mentioned in section 3. We receive the calculated result from the sample robot. To imitate the outdoor environment, this robot stands in a certain position on the field where we eliminate the landmark which the robot can easily detect. Only experience and collaboration can help the robot localize. The localization result of the sample robot which has used the collaborative approach is shown in Fig. 5. The probability distribution can converges quickly after 3-9 seconds when the dynamic reference object is taken into account.



(a) t=0s (b) t=3s (c) t=9s

Fig. 5. The localization result of applying collaborative approach with dynamic reference object. Solid arrows indicate MCL particles(100). The calculated robot position is indicated by the solid symbol. (a) is the initial uniform distribution. (b) is the calculated result after 3 seconds. (c) is the well localization result after 9 seconds

2.6 Discussion

We have demonstrated an experience based collaborative approach that combines image database for experience without landmarks and real-time sensor data for vision-based mobile robots to estimate their positions under more natural conditions towards real human soccer environment. We used the team message of dynamic reference object to improve the Markov localization for multiple mobile robots. On the one hand, our approach presented a fast and feasible system for vision-based mobile robots to localize in the dynamic environment even if there is no artificial landmark to help. On the other hand, we showed the collaborative method with introduction of *Dynamic Reference Object* to improve the accuracy and robustness of self localization, even in the circumstance that the robot can not localize individually or has no idea of who is nearby. In real robot experiments, we have shown the positive result for legged robot localization using our experience-based collaborative approach. With limit experience, robot can perform better for localization in RoboCup environment. All the experience was collected by the robot autonomously through self learning process. In collaboration, the ball with unique colour was considered as the dynamic reference object. Experiments showed the reliability of our approach in dynamic environment with collisions and sudden position changes. Experiments will be continued in more complex environment with no symmetry.

3. Autonomous Gaits Evolution Using Particle Swarm Optimization

Over the past years, plenty of publications have been presented in the biomechanics literature which explained and compared the dynamics of different high-speed gaits including gallop, canter, bound, and fast trot (eg. Alexander et al., 1980, 1983). To study and implement legged locomotion, various robot systems have been created (eg. Holmes et al., 2006; Raibert, 1986; Collins et al., 2005). However, most of the high-speed machines have passive mechanisms which may be not easy to perform different gaits. To understand and apply high-speed dynamic gaits, researchers have implemented different algorithms or hand-tune methods in the simulation (Krasny & Orin, 2004) and real robot applications (Papadopoulos & Buehler, 2000; Hornby et al., 1999; Kim & Uther, 2003). Much published research in learning gaits for different quadruped robot platforms used genetic algorithm based methods. Different from genetic algorithms, Particle Swarm Optimization (PSO) described in (Eberhart & Kennedy, 1995; Angeline, 1998; Naka et al., 2002) eliminated the crossover and mutation operations. Instead, the concept of velocity was incorporated in the searching procedure for each solution to follow the best solutions found so far. PSO can be implemented in a few lines of computer code and requires only primitive mathematical operators. Taking the memory and processing limitation onboard into account, PSO is more appropriate in gaits learning comparing with the genetic algorithm based methods for quadruped robots, especially those commercial robots with kinds of motors.

Our research focused on the gait optimization of legged robot with motor-driven joints. The commercial available quadruped robot, namely the Sony Aibo robot, which is the standard hardware platform for RoboCup four-legged league, is the main platform that we analyze and implement algorithms. Aibo is a quadruped robot with three degrees of freedom in each of its legs. The locomotion is determined by a series of joint positions for the three joints in each of its legs. Early research in gait learning for this robot employed joint positions directly as parameters to define a gait, which was the case in the first attempt to generate learned gait for Aibo. However, being lack of consistency in representing the gaits,

these parameters failed to exhibit the gait in a clear way. Most of the recent research used higher lever parameters to symbolize the gait which focus on the stance of the body and the trajectories of paw. An inverse kinematics algorithm was then implemented to convert these higher lever parameters into joint angles.
The general high-lever parameters used to describe the gait for Aibo can be divided into three groups. One group is for determining the gait patter by the relative phase for each leg. (Stewart, 1996) mentioned that there exist eight types of gait patters for quadruped animals in nature. (Hornby et al., 1999) described three of the most effective gaits for quadruped robot especially for Aibo, which are the crawl, trot and pace. Another group of the parameters is associated with the stance of robot. The last group of parameters describes the locus of the gait. Most of the gaits developed for Aibo based on this high lever parameter represent method differ in the shaped of the locus of paws or the representation of the locus, that is the actual parameters used to trace out the locus, eg. (Röfer et al. 2004, 2005).
In this part, we present the implementation of Particle Swarm Optimization in generating high-speed gaits for the quadruped robot, specifically the Aibo. First, an overview of the basic PSO and Adaptive PSO (APSO) are introduced. Our gait learning method is based on APSO. With the knowledge of using higher lever parameters to represent the gait which focus on the stance of the body and the trajectories of the paw, the inverse kinematics model is explained. Moreover, the control parameters and optimization problem are proposed. In addition, how to implement PSO in the quadruped gaits learning is introduced in detail. The whole learning process is running automatically by the robot with onboard processor. In robot experiments, we achieved an effective gait faster than previous hand-tuned gaits, using Aibo as the test platform.

3.1 Particle Swarm Optimization

(a) Overview of the Basic PSO

Particle Swarm Optimization (PSO) is a stochastic optimization technique, inspired by social behavior of bird flocking or fish schooling (Reynolds, 1987). It is created by Dr. Eberhart and Dr. Kennedy in 1995 (Eberhart & Kennedy, 1995). Similar with Genetic Algorithms, PSO method searches for optimal solutions through iterations of a population of individuals, which are called a swarm of particles in PSO. However, the crossover and mutation operation are replaced with moving inside the solution space decided by the so-called velocity of each particle. PSO has proved to be effective in solving many global optimization problems and in some areas outperform many other optimization approaches including Genetic Algorithms.
PSO theory derives from imitation of social behavior of bird flocking or fish schooling. It is discovered that each bird, when hunting for food in a bird flock, changes its flying direction based on two aspects: one is the information of food found by itself; the other is information of flying directions of other birds. When one of the birds gets food, the whole flock has food. It is similar to social behavior of human being. People's decision making is not only influenced by their own experience but also affected by other people's behavior.
For an optimization procedure, hunting food by bird flock becomes searching for an optimal solution to this problem. One solution of the problem corresponds to the position of one bird (called particle) in the searching space. Each particle remembers the best position which was found by itself so far, and this information together with its current position makes up the personal experience of that particle. Besides, every particle is informed of the best value

obtained so far by particles in its neighborhood. When a particle takes the whole flock as its topological neighbors, the best value is a global one. Each particle then changes its position in according to its velocity relied on this information: the personal best position, current position and the global best position.

In the realization of the PSO algorithm, a swarm of N particles is constructed inside a D-dimensional real valued solution space, where each position can be a potential solution for the optimization problem. The position of each particle is denoted X_i ($0 < i < N$), a D-dimensional vector. Each particle has a velocity parameter V_i ($0 < i < N$), which is also a D-dimensional vector. It specifies that the length and the direction of X_i should be modified during iteration. A fitness value attached to each location represents how well the location suits the optimization problem. The fitness value can be calculated by the objective function of the optimization problem.

At each iteration, the personal best position $pbest_i$ ($0 < i < N$) and the global best position $gbest$ are updated according to fitness values of the swarm. The following equation is employed to adjust the velocity of each particle:

$$v_{id}^{k+1} = v_{id}^{k} + c_1 r_{1d}^{k}(pbest_{id}^{k} - x_{id}^{k}) + c_2 r_{2d}^{k}(gbest_{d}^{k} - x_{id}^{k}) \tag{16}$$

Where v_{id}^{k} is one component of V_i (d donates the component number) at iteration k. Similarly, x_{id}^{k} is one component of X_i at iteration k. The velocity in equation (16) consists of three parts. One is its current velocity value, which can be thought as its momentum. The second part is the influence of the personal best. It tries to direct the particle back to the best place it has found. The last part associated with the global best attempts to move the particle toward the gbest. c_1 and c_2 are acceleration factors. They are used to tune the maximum length of flying in each direction. r_1 and r_2 are random numbers uniformly distributed between 0 and 1. They contribute to the stochastic vibration of the algorithm. It should be noted that each component of the velocity has new random numbers, not that all the components share the same one. In order to prevent particles from flying outside the searching space, the amplitude of the velocity is constrained inside a spectrum $[-v_d^{\max}, +v_d^{\max}]$. If $v_d^{\max}$ is too big, the particle may fly beyond the optimal solution. If $v_d^{\max}$ is too small, the particle will easily step into the local optimum. Usually, $v_d^{\max}$ is decided by the following equation:

$$v_d^{\max} = k x_d^{\max} \tag{17}$$

where $0.1 \leq k \leq 1$. Now the current position of particle i can be updated by the following equation:

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1} \tag{18}$$

PSO algorithm is considerably easy to realize in computer coding and only a few primitive mathematical operators are involved. Furthermore, it has the advantage of multiple points searching at the same time. Most importantly, the speed of converging is remarkably high in

many learning processes. It is a critical virtue when it comes to learning gaits in a physical robot, because it minimizes damage to the robot.

The basic PSO is an algorithm base on stochastic searching, so it has strong ability in global searching. However, in the final stage of searching procedure, it is difficult to converge to a local optimum because the velocity still has much momentum. To improve the local searching ability in the final stage of optimization process, the influence of previous velocity on the current velocity needs to decrease. Thus, we proposed the using of adaptive PSO with changing inertia weight in this study.

(b) Adaptive PSO with Changing Inertia Weight

In equation (16), by multiplying inertia weight to the momentum part of the velocity vibration can control the impact of previous velocity on the current velocity. The update equation for velocity with inertial weight is as follows:

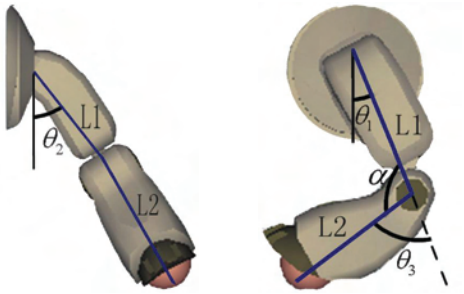
$$v_{id}^{k+1} = wv_{id}^{k} + c_1 r_{1d}^{k}(pbest_{id}^{k} - x_{id}^{k}) + c_2 r_{2d}^{k}(gbest_{d}^{k} - x_{id}^{k}) \tag{19}$$

where w is the inertia weight. PSO with larger inertial weight results in better global searching ability for the reason that the search area is expanded with more momentum. Small inertial weight limits the search area thus improving local searching ability. Empirical results show that PSO has faster convergent rate when w falls in the range from 0.8 to 1.2. With the intention of realizing both fast global search at the beginning and intensive searching in the final stage of iteration, the value of w should vary gradually from high to low. It is similar to the annealing temperature of Simulated Annealing Algorithm. In this way, both global searching in a broaden area at the beginning and intensive search in a currently effective area at the end can be realized.

3.2 Optimization Problem

(a) Inverse Kinematics Model

The high-lever parameters that we adopt to represent the gait need to be transferred to joint angles of legs before they can be implemented by the robot. An inverse kinematics model can be used to solve this problem. For a linked structure with several straight parts connecting with each other, the position of the end of this structure relative to the starting point can be decided by all angles of linked parts and only one position results from the same angle values. The definition of the kinematics model is the process of calculating the position of the end of a linked structure when given the angles and length of all linked parts. In this robot Aibo case, given the angles of all the joints of the leg, the paw positions relative to the shoulder or the hip will be decided. Inverse kinematics does the reverse. Given the position of the end of the structure, inverse kinematics calculates out what angles the joints need to be in to reach that end point. In this study, the inverse kinematics is used to calculate necessary joint angles to reach the paw position determined by gait parameters. Fig.6 shows the inverse kinematics model and coordinates for Aibo. The shoulder or hip joint is the origin of the coordinate system. l_1 is the length of the upper limb, while l_2 is the length of the lower limb. Paw position is represented by point (x, y, z). The figures and equations below only give the view and algorithm to get the solution for left fore leg of robot. In according to the symmetrical characteristic of legs, all other legs can use the same equations with some signs changing.



(a) (b)

Fig. 6. The inverse kinematics model and coordinates for Aibo. (a) is the front view of left fore leg. (b) is the side view of left fore leg

The following equations shows the inverse kinematics model:

$$\begin{aligned} x &= l_2 \cos\theta_1 \sin\theta_3 + l_2 \sin\theta_1 \cos\theta_2 \cos\theta_3 + l_1 \sin\theta_1 \cos\theta_2 \\ y &= l_1 \sin\theta_2 + l_2 \sin\theta_2 \cos\theta_3 \\ x &= l_2 \sin\theta_1 \sin\theta_3 - l_2 \cos\theta_1 \cos\theta_2 \cos\theta_3 - l_1 \cos\theta_1 \cos\theta_2 \end{aligned} \tag{20}$$

The inverse kinematics equation to get θ_1, θ_2, θ_3 by the already known paw position (x, y, z) is as follows:

$$\begin{aligned} \theta_1 &= \cos^{-1} \frac{x^2 + y^2 + z^2 - l_1^2 - l_2^2}{2 l_1 l_2} \\ \theta_2 &= \sin^{-1} \frac{y}{l_2 \cos\theta_3 + l_1} \\ \theta_3 &= -\tan^{-1} \frac{a}{b} \pm \cos^{-1} \frac{x}{a^2 + b^2} \end{aligned} \tag{21}$$

where $a = l_2 \sin\theta_3$, $b = -l_2 \cos\theta_2 \cos\theta_3 - l_1 \cos\theta_2$.

One problem with the inverse kinematics is that it always has more than one solution for the same end point position. However, as to Aibo, only one solution is feasible due to the restriction on the joint structure. As a result, when using inverse kinematics to calculate joint angles, it is necessary to take joint structure limitation into consideration to get the right solution. Otherwise, it will possibly cause some physical damage to the robot platform.

(b) Control Parameters

Before we run the learning gait procedure, the control parameters representing a gait need to be decided. There are two rules based on which we choose our parameters: One is the sufficient representation of the gait that makes it possible to get a high-performance gait in an expanded area. The other one is the attempt to limit the number of control parameters in order to reduce the training time. These two rules are to some extent contradicted with each other. We have to find a better way to compromise these two policies manually. We have done some work on the robot's gait patters and found out that trot gait is almost always the

most effective pattern in terms of both stability and speediness,thus we limit the gait pattern to mere trot gait.

For stance parameters, based on our observation and analyze of the motion for Aibo, we conclude that forwardleaning posture can speed up the walking, thus we constrain the range of stance parameters to keep robot in forwardleaning posture, that is the height of hip higher than that of chest. As to locus, we choose rectangle shape because it has proved to be effective in quadruped gaits and it is simple to be represented. And because of the symmetry of right and left side when moving straight forward, we use the same locus for right legs and left legs. In all, we choose our parameters of gait as shown in Table 3.

Parameter Name	Definition
fore height	vertical height from paw to chest
hind height	vertical height from paw to hip
fore width	transverse distance between paw and chest
hind width	transverse distance between paw and hip
fore length	forward distance between paw and chest
hind length	forward distance between paw and hip
step length	time for one complete step in 0.008 second units
fore step height	fore height of the locus
hind step height	hind height of the locus
fore step width	fore width of the locus
hind step width	hind width of the locus
fore ground time	fore paw fraction of time spent on ground
hind ground time	hind paw fraction of time spent on ground
fore lift time	fore paw fraction of time spent on lifting
hind lift time	hind paw fraction of time spent on lifting
fore lowing time	time spent on fore paw lowing around locus
hind lowing time	time spent on hind paw lowing around locus

Table 3. Control parameters in gaits evolution

3.3 Implementation of PSO

Given the parametrization of the walking defined above, we formulate the problem as an optimization problem in a continuous multi-dimensional real-value space. The goal of the optimization procedure is to find a possibly fastest forward gait for the robot, therefore the objective function of the optimization problem is simply the forward speed of the walking parameters. Particle Swarm Optimization is then employed to solve this problem with a particle corresponding to a set of parameters. A predetermined number of sets of parameters construct a particle swarm which will expose to learning by PSO, with the forward speed of each parameter being the fitness.

(a) Initialization

Initially, a swarm of particles are generated in the solution space, which is a set of feasible gait parameters. These particles can be represented by $\{p_1, \Lambda, p_N\}$ (where N=10 in this case).

These sets of parameters are acquired by random generation within the parameter limits decided by the robot mechanism. A lot of previous work done on learning gaits start from a hand-tune set of parameters. Comparing with previous work, random generation of initial values has the advantage of less human intervention, and more importantly, has more possibility to lead to different optimal values among different experiments. Initial velocities for all particles are also generated randomly in the same solution space within given ranges.

The width of the range is chosen to be half of that of the corresponding parameters. Velocity calculated later is also constrained inside the spectrum. The spectrum is denoted by ($-V^{\max}$, $+V^{\max}$), where $V^{\max} = \frac{1}{4}(x^{\max} - x^{\min})$, with ($x^{\max}, x^{\min}$) as the changing range of particle P_i.

The ranges we chosen turn out to be appropriate to avoid the two problems mentioned in Section 3.1. To expedite the search process, c_1 and c_2 are set to 2. The initial *pbest*s are equal to the current particle locations. There is no need to keep track of *gbest* while it can be acquired from *pbest*s, that is the pbest with the best fitness is *gbest*.

(b) Evaluation

The evaluation of parameters is performed using sole speed. Since the relation between gait parameters and speed is impossible to acquired, we do not know the true objective function. There is no sufficiently accurate simulator for Aibo due to the dynamics complexity. As a result, we have to perform the learning procedure on real robots. In order to automatically acquiring speed for each parameter set, the robot has to be able to localize itself. We use black and while bar for Aibo to localize, because given the low resolution of Aibo's camera, it is faster and more accurate to detect black-while edge than other things. We put two pieces of boards with the same black and while bars in parallel so the robot can walk between them.

During evaluation procedure, the robot walks to a fixed initial position relative to one of the boards, then load the parameter set needed to be evaluated, walk for a fixed time, 5s, stop and determine the current position. It should be noted that both before and after the walk, robot is in static posture, so the localization is better compare to localizing while running. Now the starting and ending location have been acquired from detecting the bars, speed can be calculated out. After that, robot turns around by 90 degree, and localizes according to the other board, if the position is far from the fixed position, adjust it or else go to the next step, loading another set of parameters and then begin another trial. The total time for testing one set of parameter including turning, localizing, and walking time. Because of the ease of localizing, usually it takes less than 3s to turn and get to the right position. As a result, the test time of one particle is less than 8s.

(c) Modification

After all particles of the swarm are evaluated, *pbest*s are updated by comparing them with corresponding particles. If the performance of P_i is better than pbesti, which means the fitness value of P_i is higher than that of pbesti, pbesti will be replaced by the new position of P_i. In addition, the fitness value of P_i is recorded as fitness value of $pbest_i$ for future comparing. Subsequently, the new *gbest*, the best among pbests can be acquired. It should be noted that the update of *gbest* is not done anytime a particle is evaluated but after the whole swarm is evaluated. The difference does not change the principle of the algorithm or empirically influence the converge rate.

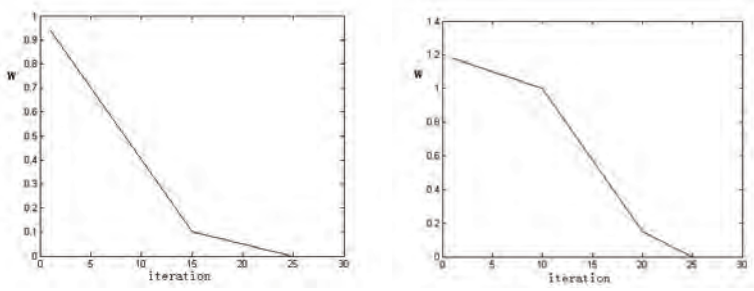
As mentioned in section 3.1, in order to realize global search in a broaden area at the beginning of the learning procedure and intensive search in a currently effective area at the end, we employ adaptive PSO with piecewise linearity declining inertial weight to perform the learning procedure. When inertial weight value ω is around 1, it presents global search characteristics and results in fast converge rate. When ω is a lot less than 1, intensive search is realized.

3.4 Real Robot Experiments

Using the method described above, we take two separate experiences and achieve favorable results. In the first experience, since large inertial weight will extend the searching area, resulting in a long time of training, we take a conservative move and reduce inertial weight quickly from the start with initial value being 1. The inertial weight is determined by equation (22). Fig. 7(a) shows the vibration of ω through iterations. By iteration 15, ω has decreased to 0.1. The global search is diminished, while the intensive search is enhanced. Fig. 8(a) shows the result through iterations. We can see that the learning process is converging quite fast from 1 to 10 iteration. After that, the result improve slowly but firmly until around 25 iteration. Although we get a high-performance gait in a short time in this experiment, we think it is possible that we can have a better result when extending the search area a little by not reducing ω so fast. So we tried to use another equation (23) to update ω, Fig. 7(b) shows the vibration of ω, and Fig. 8(b) shows the learning result.

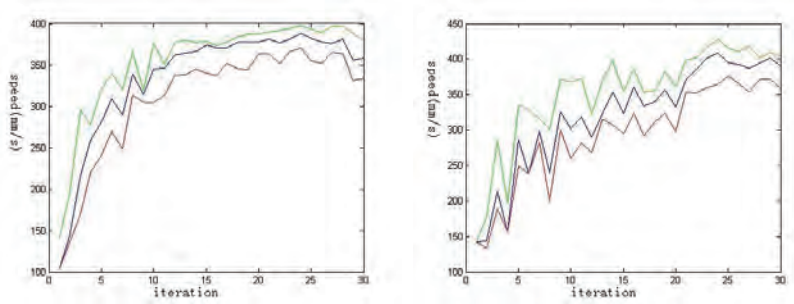
$$\begin{aligned} &\omega = 1 - 0.06 \times iter; (iteration \le 15) \\ &\omega = 0.1 - 0.01 \times (iter - 15); (15 < iteration \le 25) \\ &\omega = 0; (iteration > 25) \end{aligned} \tag{22}$$

$$\begin{aligned} &\omega = 1.2 - 0.02 \times iter; (iteration \le 10) \\ &\omega = 1 - 0.085 \times (iter - 10); (10 < iteration \le 20) \\ &\omega = 0.15 - 0.03 \times (iter - 20); (20 < iteration \le 25) \\ &\omega = 0; (iteration > 25) \end{aligned} \tag{23}$$



(a) (b)

Fig. 7. Vibration of inertial weight ω through iteration in real robot experiments. (a) shows the vibration of inertial weight ω through iterations in the first experiment. (b) shows the vibration of inertial weight ω through iteration in the second experiment



(a) The first experiment (b) The second experiment

Fig. 8. Optimization results. (a) is the best(in the green line), average of the whole swarm(in the red line)and average of the best half part of the swarm(in the blue line) in real robot experiments. (b) the best result of every iteration in both the two experiments. The green is the first one, and the blue is the second one

We can see that the second experiment achiever better result than the first one. It is interesting that they both reach their peak in the 25 iteration, when ω becomes zero. It's possible that PSO has little local optimization when current velocity is no longer influenced by previous velocity which is contradicted to what we assumed.

We can also note that there are both advantage and disadvantage comparing these experiments with each other. For one thing, the learning curve of the first experiment is a lot smoother than that of the second one. It means that the second learning process has more undulation. In fact, during the second experiment, there are still new sets of parameters that perform very poor after the 10 iteration due to the extended searching area. This problem cause more damage to physical robot. However, the second experiment acquire better parameters also because of the extended searching area. Fig. 9 shows the best result of every iteration in both the two experiments.

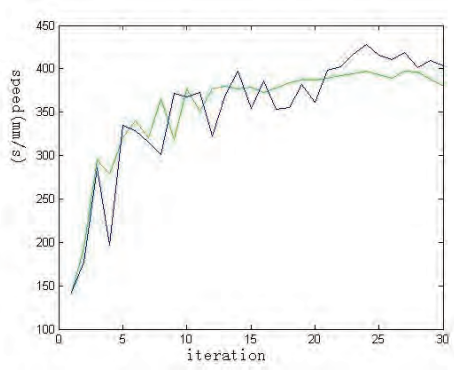


Fig. 9. The best result of every iteration in both the two experiments. The green line is the first one, while the blue line is the second one

3.5 Discussion

In this part, we have demonstrated a novel evolutionary computation approach to optimize fast forward gaits using Particle Swarm Optimism. PSO has been proven to be remarkable effective in generating optimal gaits in the robot platform Aibo. Our method was easily coded and computationally inexpensive. Moreover, by using PSO, the evolution converged extremely fast and the training time was largely reduced. That is an essential advantage for physical robot learning, minimizing possible damage to the robot. Another contribution of our method was its initial sets of parameters are randomly generated inside the value range instead of mutation from a hand-tune set of parameters. It reduced the human work as well as generating evolutional results varied a lot in different experiences. Through experiments which took about 40 minutes each, we achieved several highperformance sets of gait parameters which differ a lot from each other. These gait parameter sets were among the fastest forward gaits ever developed for the same robot platform.

In the future, we will compare different high-performance gait parameters and analyze the dynamics model of the robot and in an attempt to get a deeper sight into the relation between parameter and its performance. After that, we will be able to generate more effective gaits in less learning time. Through analysis, we find that the gait actually executed by robot differ significantly from the one we design. There are several reasons accounting for that. The most important one is the interaction with environment prevents the implement of some strokes of robot legs. Although with learning approach, factors that cause the difference between actual gait and planned gait do not have to be taken into consideration. However, we assume that if the planned gait and actual gait can conform with each other, Aibo will walk more stable and fast. In order to solve the problem, the analysis of dynamics between the robot and the environment is necessary. In this gait learning procedure, we only evolve fast forward gait and choose forward speed as the fitness. Later on, we will try to learn effective gaits in other directions, for example, gaits for walking backward, sideward and turning. We also consider exploring optimal omnidirectional gaits. With gaits working well at all directions, robots will be able to perform more flexibly and reliably.

4. Intelligent Behaviors

4.1 Obstacle Avoidance

In robot soccer competition, we introduce time-variable limit cycle to help robot avoid obstacles. To show the approach, we simply describe the shape of Aibo as a cycle in the two dimensional plane. Considering the following nonlinear system for dynamic limit cycle applying in Aibo:

$$\begin{aligned} \dot{\tilde{x}} &= \rho(\tilde{y} + \gamma\tilde{x}(\frac{1}{4}\overline{v}^2 - \tilde{x}^2 - \tilde{y}^2)) \\ \dot{\tilde{y}} &= \rho(-\tilde{x} + \gamma\tilde{y}(\frac{1}{4}\overline{v}^2 - \tilde{x}^2 - \tilde{y}^2)) \end{aligned} \tag{24}$$

where ρ is the character factor of the obstacle which is set to be a positive value. γ is the convergence factor. And v is the relative velocity to the obstacle which is dynamic when the robot moves. The size of limit cycle is changing when system (24) switches. To prove the

circle $\tilde{x}^2+\tilde{y}^2=\frac{1}{4}\overline{v}^2$ is the dynamic limit cycle of the switched system(1), we use the common Lyapunov function:

$$V(\tilde{x},\tilde{y})=\tilde{x}^2+\tilde{y}^2 \tag{25}$$

Such that:

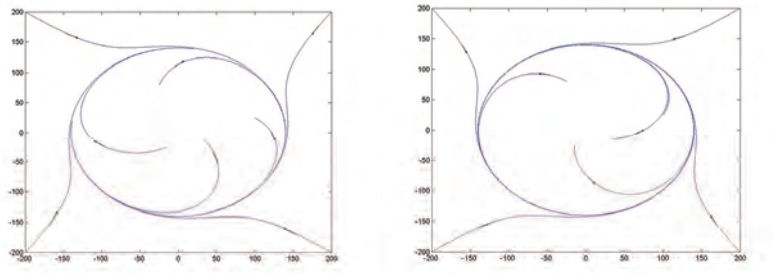
$$\dot{V}(\tilde{x},\tilde{y})=2\rho\gamma(\frac{1}{4}\overline{v}^2-\tilde{x}^2-\tilde{y}^2)(\tilde{x}^2+\tilde{y}^2) \tag{26}$$

For limit cycle, we can see that $\dot{V}(\tilde{x},\tilde{y})<0$ when $\dot{V}(\tilde{x},\tilde{y})>\frac{1}{4}\overline{v}^2$, while $\dot{V}(\tilde{x},\tilde{y})>0$ when $\dot{V}(\tilde{x},\tilde{y})<\frac{1}{4}\overline{v}^2$. This shows the following region is absorbing.

$$B=\{\rho_1\le V(\tilde{x},\tilde{y})\le\rho_2,|\,0<\rho_1<\frac{1}{4}\overline{v}^2,\rho_2>\frac{1}{4}\overline{v}^2\} \tag{27}$$

Since this argument above is valid for any $0<\rho_1<\frac{1}{4}\overline{v}^2$, and $\rho_2>\frac{1}{4}\overline{v}^2$, when ρ_1, ρ_2 get close to $\frac{1}{4}\overline{v}^2$, region B shrinks to the circle $V(\tilde{x},\tilde{y})=\frac{1}{4}\overline{v}^2$. This shows that the circle is a periodic orbit as shown in Fig. 10(a) when $\overline{v}$ = 280, ρ = 0:01, γ = 0:0001. This periodic orbit is called a limit cycle. We can see the trajectory from any point $(\tilde{x},\tilde{y})$ moves toward and converges to the limit cycle clockwise when close. The counterclockwise condition can be derived by the following system (shown in Fig. 10(b)):

$$\begin{aligned}\dot{\tilde{x}}&=\rho(-\tilde{y}+\gamma\tilde{x}(\frac{1}{4}\overline{v}^2-\tilde{x}^2-\tilde{y}^2))\\ \dot{\tilde{y}}&=\rho(\tilde{x}+\gamma\tilde{y}(\frac{1}{4}\overline{v}^2-\tilde{x}^2-\tilde{y}^2))\end{aligned} \tag{28}$$



(a) clockwise (b) counterclockwise

Fig. 10. Phase portrait of limit cycle

Considering that the trajectory from any point $(\tilde{x},\tilde{y})$ inside the limit cycle moves outward the cycle, and the trajectory from any point $(\tilde{x},\tilde{y})$ outside the limit cycle approaches the cycle with distance determined by the relative speed $\overline{v}$, the limit cycle provides a method for obstacle avoidance among multiple mobile robots.

In RoboCup Four-legged League, there are many obstacles during the game. Robots can be considered as motive obstacles. When the robot approaches a teammate holding ball, it must stay out of the area where teammate handles ball, and be ready to perform cooperative strategies. If the robot holding ball encounters an opponent, it must control the ball and quickly avoid the approaching robot, especially when perform kicking ball in front of opponent goalie. Own penalty area is another one that can be taken for an obstacle. If the robot moves parallel to own ground line, it must avoid from walking into the own penalty area.

When the robot is in a safe region, by the dynamic limit cycle approach, it will move away the obstacle toward the safe circle with a radius relevant to the speed of the obstacle. Let α denote the orientation of the obstacle, (x_0,y_0) the centre point of the obstacle. With the following transformation, we get the expression of system (24) in the original frame:

$$\begin{aligned} x &= \cos\alpha(\tilde{x}+x_0)-\sin\alpha(\tilde{y}+y_0) \\ y &= \sin\alpha(\tilde{x}+x_0)+\cos\alpha(\tilde{y}+y_0) \end{aligned} \tag{29}$$

Let v denote the translational velocity of the robot in the original frame, θ the direction of the motion. The kinematic model of the robot is described by:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \frac{\dot{x}}{\dot{y}} &= \tan\theta \end{aligned} \tag{30}$$

Then we can see:

$$\begin{aligned} v &= \sqrt{\dot{x}^2+\dot{y}^2} \\ \theta &= \arctan\frac{\dot{x}}{\dot{y}}+\alpha \end{aligned} \tag{31}$$

Different obstacles have their own characters, with ρ matching to characters respectively. Using ρ in different values can control the magnitude of the absolute speed.

With the dynamic radius of the limit cycle, robot can perform more flexibly and rationally. Satisfactory results are obtained in robot experiments. The implementation of this method is introduced in (Wang, 2006b) in detail.

4.2 Perform Near Border

In real robot soccer, behaviors and strategies correlated to border line are important. Any inappropriate behavior near border may cause a negative impact. For example, if the ball is

near border in own half field, it is dangerous for the defender to handle ball inappositely to let it out of field. Because it may benefit the opponent striker to control the ball. To avoid this situation, we implement the near border behavior.

In competition of RoboCup Four-Legged League, we define that for a player, if the distance to border line is less than 600mm, it enters the *near border area*. It is simple that if the player handles ball near border, it can hold ball and move it along the direction vertical to borderline. However, actual test shows that different gaits along with grabbing ball motion may not help control ball well. Therefore, we divide the circle area around player into four parts. Fig. 11 shows the different parts of the *near border area* which may activate strategies respectively. We define the variable *robotPose.anlgle-to-border* which represents the absolute value of angle between robot's body direction and normal line to the border. Area 1 is the place where the *angle-to-border* is in range from 120° to 180°. In area 2 and 3, the angle is between 80° and 120°. Area 4 means the angle is less than 80°. In area 1, the robot grabs ball and adjusts its body direction first. Then the robot performs a sideways walk moving ball into field. In area 2 and 3, the robot performs a sideways walk directly. Player walks forward directly to the field if enters the area 4.

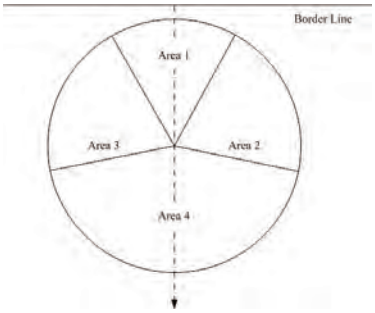


Fig. 11. Strategy field in near border area

5. Conclusion

We have made improvement in localization, locomotion and behavior modules. In RoboCup 2006, we perform our technical improvement in open challenge, passing ball and new goal challenge. After RoboCup 2006, we participated in RoboCup China Open 2006. Advantages in sharPKUngfu 2006 help our team make great success in this event. We got champions both in soccer competition and technical challenge. After the event, we focus our research on further study in collaborative localization, navigation and gaits optimization. All the improvement is explained above in detail. We have applied experience-based collaborative approach for localization which is important to make robots more rational and efficient. In gaits optimization, we implemented PSO based approach to get relatively high-speed forward gaits. To perform better under the soccer rule 2006, new behaviors and relevant actions have been created to hold ball in the field to get better performance. Besides, we tried to apply new approach to percept robots and avoid dynamic obstacles. Experiments in our lab show positive effect by using the real-time approach.

In the future, we plan to let robot play in the environment without any landmark towards real human soccer conditions. Further study should be continued to exploit enough

surrounding information to help self-localization. In vision module, we plan to implement color-edge based method to recognize beacons and goals which are newly defined in soccer rule. Beside of forward gaits optimization, we will implement PSO in other different walking types to gain optimized motion parameters. In multi-robot coordination, the research on formation control will continue. In addition, we will continue to get involved in challenges of passing ball and obstacle avoidance. The final version of our code 2006 is now available on our web site.

6. Acknowledgment

This work was supported by National Natural Science Foundation of China (No. 60404001 and No. 60274001), National Key Basic Research and Development Program (2002CB312200) and 985 Project of Peking University.
The authors gratefully acknowledge the contribution of other team members of the sharPKUngfu Legged Robot Team, including Lianghuan Liu, Yan Huang, Xin Guan, Jiajie Guo and Kaituo Zhou. The source code and experiment video can be downloaded from the sharPKUngfu official web site: http://www.mech.pku.edu.cn/robot/fourleg/.

7. References

Alexander, R. M.; Jayes, A. S. & Ker, R. F. (1980). Estimates of energy cost for quadrupedal running gaits, *J. Zoolog*., vol. 190, pp. 155C192, 1980.

Alexander, R. M. & Jayes, A. S. (1983). A dynamic similarity hypothesis for the gaits of quadrupedal mammals, *J. Zoology Lond*., vol. 201, pp. 135C152, 1983.

Angeline, P. (1998). Using selection to improve particle swarm optimization, *Proceedings of the International Conference on Evolutionary Computation*, 1998, pp. 84-89

Arkin, R. C. & Balch, T. (1998). Cooperative multiagent robotic systems, In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, MIT/AAAI Press, Cambridge, MA

Collins, S.; Ruina, A.; Tedrake, R. & Wisse, M. Efficient bipedal robots based on passive dynamic walkers, *Science*, vol. 307, 2005, pp. 1082-1085.

Eberhart, R. & Kennedy, J. (1995). Particle swarm optimization, *Proceedings of the IEEEConference on Neural Network*, Pelfh, Australia, 1995, pp. 1942-1948

Fox, D.; Burgard, W.; Dellaert, F. & Thrun, S. (1999). Monte Carlo localization: Efficient position estimation for mobile robots, *Proceedings of the National Conference on Artificial Intelligence*, pp. 343-349

Fox, D.; Burgard, W.; Kruppa, H.; & Thrun, S. (2000). A probabilistic approach to collaborative multi-robot localization, *Autonomous Robots*, Vol. 8, No. 3, 2000, pp. 325-344.

Holmes, P.; Full, R. J.; Koditschek, D. & Guckenheimer, J. (2006). The dynamics of legged locomotion: Models, analyses, and challenges, *SIAM Review*, Vol. 48, No. 2, 2006, pp. 207-304.

Hornby, G. S.; Fujita, M.; Takamura, S.; Yamamoto, T. & Hanagata, O. (1999). Autonomous evolution of gaits with the Sony quadruped robot. In Banzhaf, W. et al. eds., *Proceedings of the Genetic and Evolutionary Computation Conference*, vol. 2, 1297-1304. Orlando, Florida, USA: Morgan Kaufmann, 1999.

Kim, M. S. & Uther, W. (2003). Automatic gait optimisation for quadruped robots. *Australasian Conference on Robotics and Automation*, 2003.

Krasny, D. P. & Orin, D. E. (2004). Generating high-speed dynamic running gaits in a quadruped robot using an evolutionary search, *IEEE Transactions on Systems, Man, and Cybernatics - Part B: Cybernetics*, vol. 34, No. 4, 2004, pp. 1685-1696.

Naka, S.; Genji, T.; Yura, T. & Fukuyama, Y. (2002). Hybrid particle swarm optimization based distribution state estimation using constriction factor approach, *Proc. Int. Conf. SCIS & ISIS*, vol. 2, 2002, pp. 1083-1088.

Papadopoulos, D. & Buehler, M. (2000). Stable running in a quadruped robot with compliant legs, *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, 2000, pp. 444-449.

Raibert, M. H. (1986) *Legged robots that balance*, MIT Press, Cambridge; 1986.

Reynolds, C. (1987). Flocks, herds, and schools:Adistributed behavioural model, *Comp. Graph.*, vol. 21, no. 4, pp. 25-34, 1987.

Röfer, T. & Jüngel, M. (2003). Vision-Based Fast and Reactive Monte-Carlo Localization, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 856-861, 2003, Taipei, Taiwan

Röfer, T.; Burkhard, H. D.; Düffert, U.; Hoffmann, J.; Göhring, D.; Jüngel, M.; Lötzsch, M.; Stryk, O. v.; Brunn, R.; Kallnik, M.; Kunz, M.; Petters, S.; Risler, M.; Stelzer, M.; Dahm, I.; Wachter, M.; Engel, K.; Osterhues, A.; Schumann, C. & Ziegler, J. (2004). GermanTeam RoboCup 2004, technical report, Available online: http://www.germanteam.org/GT2004.pdf

Röfer, T.; Burkhard, H. D.; Düffert, U.; Hoffmann, J.; Göhring, D.; Jüngel, M.; Lötzsch, M.; Stryk, O. v.; Brunn, R.; Kallnik, M.; Kunz, M.; Petters, S.; Risler, M.; Stelzer, M.; Dahm, I.; Wachter, M.; Engel, K.; Osterhues, A.; Schumann, C. & Ziegler, J. (2005). *GermanTeam RoboCup 2005*, Technical report, 2005.

Schmitt, T.; Hanek, R.; Beetz, M.; Buck, S. & Radig, B. (2002). Cooperative probabilistic state estimation for vision-based autonomous mobile robots, *IEEE Transactions on Robotics and Automation*, 2002, Vol. 18, Issue 5, pp. 670-684

Shi, Y. & Eberhart, R. (1998). A modified particle swarm optimizer, *Proc. IEEE Int. Conf. Evolutionary Computation*, 1998, pp. 60-73

Stewart, I. (1996). Nature's Numbers. 1996.

Wang, Q.; Liu, L.; Xie, G. & Wang, L. (2006a). Learning from human cognition: collaborative localization for vision-based autonomous robots, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3301-3306, October 2006, Beijing, China

Wang, Q.; Rong, C.; Liu, L.; Li, H. & Xie, G. (2006b). The 2006 sharPKUngfu Team Report, technical report, http://www.mech.pku.edu.cn/robot/fourleg/en/resources.htm

Wang, Q.; Huang, Y.; Xie, G. & Wang, L. (2007). Let robots play soccer under more natural conditions: experience-based collaborative localization in four-legged league, *11th International Symposium on RoboCup 2007* (Robot World Cup Soccer Games and Conference), Lecture Notes in Artificial Intelligence, Springer, 2007.

Wolf, J.; Burgard, W. & Burkhardt, H. (2005) Robust vision-based localization by combining an image-retrieval system with Monte Carlo localization, *IEEE Transactions on Robotics*, Vol. 21, Issue 2, April 2005, pp. 208-216

29

The Robotic Water Polo and Underwater Robot Cooperation Involved in the Game

Zhang Lee, Guangming Xie, Dandan Zhang and Jinyan Shao
Intelligent Control Laboratory, College of Engineering, Peking University
China

1. Introduction

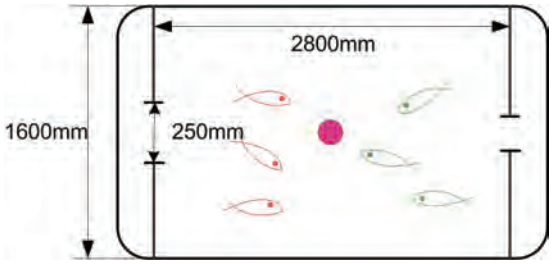
Multi-robot cooperation is a very important area in the field of robot. As we know, it is difficult to use a single robot to complete tasks independently. Often, people use multiple robots cooperation in order to compensate for the lack of individual ability. Multi-level robot can greatly improve the efficiency of the whole system. Compared to a single robot, a coordinated multi-robot system has many advantages : First, because of multi-robot systems' distribution in space, resources and functions, multi-robot systems have higher efficiency and wider scope of the application than single robot; Second, because multi-robot systems have high redundancy, So multi-robot systems have more fault-tolerant capabilities and higher robust than single robot; Also compared with the design of a very powerful independent robot, construction of a number of robots which have simple structure and function will be economical, easy, flexible, and can greatly reduce the costs. Due to the advantages of the multi-robot systems, Multi-robot cooperation is drawing increasing attention to the people, a lot of researches on multi-robot coordination and centralization, Load distribution, motion planning and other issues have been done all over the world.

By now, multi-robot cooperation achievements were mainly concentrated in the areas of robots on land, there has few achievements of underwater robot collaboration. The main reason is that the complexity and uncertainty of the underwater environment bring a lot of interference to the system, the efficiency and accuracy of the control are reduced to a large extent by these interfere. In addition, the underwater environment has a higher requirement that the robot should have a higher compressive strength, anti-jamming capabilities and more accurate and robust sensing system. In fact, along with the exploration of underwater resources, there are more and more underwater tasks, many underwater tasks need a number of robots to collaborate because of their high complexity. The research on underwater robot collaboration has become an urgent task.

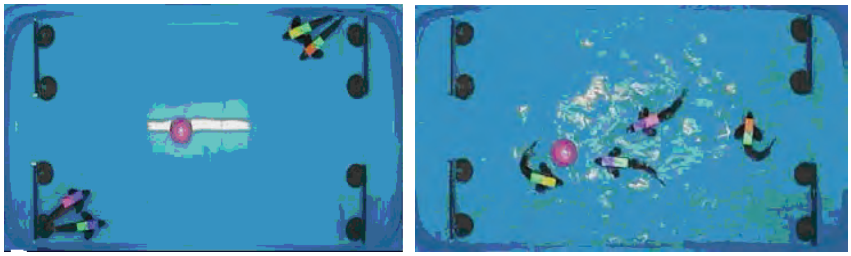
We propose a novel robotic soccer game called Robotic Water Polo(RWP) to promote the underwater robot technique and their combination, Fig.1 shows the filed of RWP, Similar to the popular robotic soccer games on land, the RWP is also designed as standard task for multiple swimming robots under a dynamic underwater environment. To play RWP, each team has three or two underwater robots, we called them robotic fish. They do their best to

push the ball to their opponent's gate, Fig.1(b) and (c)display the procedure of the competition. Because the special environment of the game, the RWP also involves hydrodynamic analysis, underwater communications, underwater image processing, anti-jamming technology, and other aspects of technology which is more difficult, more complicated than the similar game on land.

We focus this chapter on introducing the underwater robot cooperation involved in RWP and the basis of RWP, due to space limitation, we can't elaborate on all the cooperative technologies involved in RWP. We only introduce collision avoidance of robotic fish and the multiple robot fish cooperation system of we used in RWP now, in section 2, we first introduce the prototype of the underwater robot called robotic fish, in section 3 we introduce a general task-oriented platform which provides both hardware and software foundations for cooperation of underwater robots, in section 4 we propose a new reactive collision avoidance method for robotic fish navigation in RWP, in section 5 a situation based action selection mechanism is proposed for pushing ball to goal in RWP, and finally in section 6, there will be a conclusion.



(a) Field of RWP(Used by Peking university)



(b)Start of RWP (c) Fragment of RWP

Fig.1. Field and display of RWP

2. The Prototype of Robotic Fish

Before introducing the underwater robot cooperating technologies, we first present the robotic fish prototype developed for RWP in our laboratory, because it is the basis of RWP. Fig 2 shows the prototype of robot fish:

Fig. 2. Prototype of robot fish used in RWP

This multiple-mode robot-fish is mainly used to realize the cooperative control of multiple robots underwater. And the advantages of the fish are low cost, small size, and good agility, stability and accuracy. In the environment of lab or competition, the location information of the robot-fish can be gotten from the global vision, so we can conveniently control the cooperative movements of the multiple-robot fish.

The structure of robot-fish can mainly be divided into three parts: fish head, body, tail.The fish head portion is composed of the control circuit board, battery, communication module, power switch and pectoral fin.The fish body portion is composed of three in series swaying joints, and each joint is driven by the direct current servo motor, in order to simulate the wave curvilinear motion of the fish's body. The fish tail is tail fin of the robot-fish. The mechanical structure of the robot-fish is as the Fig 2, For technical details on design and implementation of the robotic fish. We can't elaborate on them here due to space limitation.

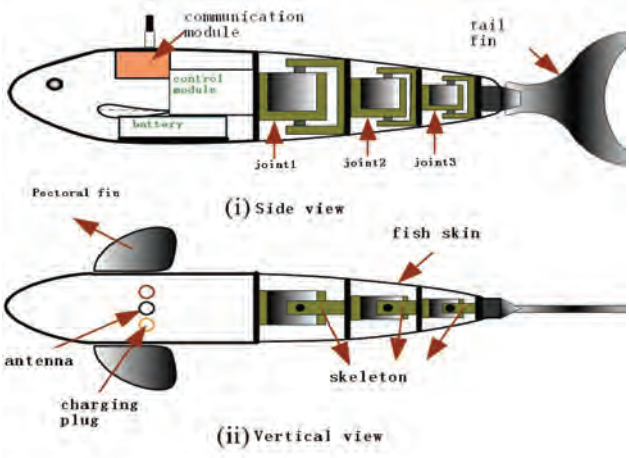


Fig. 3. The mechanical structure of the robot-fish

This type of multiple-mode robot-fish involved in the lab presently can complete precision control in the mission of multiple-robot-fish cooperative control, and also can ensure both the stability of the robot-fish's swimming and accuracy of the swimming performance. We have tested the robot-fish repeatedly through experimentation. It can satisfy the requirement of the RWP and other cooperative control.

3. The Multiple Robotic Fish Cooperation System

In recent years, underwater biomimetic robotics has emerged as a challenging new research topic, which combines bioscience engineering technology and underwater robotics, aiming at developing new classes of robots which will be substantially more compliant and stable than current robots. Taking advantage of new developments in materials, fabrication technologies, sensors and actuators, more and more biologically inspired robots have been developed. As one of the hot topics, robotic fish has received considerable attention during the last decade. Fish, after a long history's natural selection, have evolved to become the best swimmers in nature. They can achieve tremendous propulsive efficiency and excellent maneuverability with little loss of stability by coordinating their bodies, fins and tails properly. Researchers believe that these remarkable abilities of fish can inspire innovative designs to improve the performance, especially maneuverability and stabilization of underwater robots.

By now, the research on robotic fish mainly focuses on design and analysis on individual robot fish prototype, while seldom is concerned with the cooperation behaviors of the fish. As we know, fish in nature often swim in schools to strive against the atrocious circumstances in the sea. Similarly, in practice, the capability of a single robot fish is limited and it will be incompetent for achieving complex missions in dynamic environments. Thus, for real-world (ocean based) applications, a cooperative multiple robot fish system is required, which is the motivation of this work.

In this section, we propose the development of the Multiple Robot Fish cooperation System (MRFS), which is built on the basis of a series of radio-controlled, multi-link biomimetic robot fish. There are two features in MRFS: first, this cooperative platform is general and can be applied to different types of cooperative tasks; second, high-level tasks are eventually decomposed into two reactive motion controllers, which are designed under full consideration on the inertia of fish and the hydrodynamic forces of surrounding water.

The remaining of the section is organized as follows. In subsection 3.1 we present the establishment of the platform for MRFS. Based on the hardware and software platform, a four-level hierarchical control architecture is provided in subsection 3.2.

3.1 Design and Implementation of MRFS Platform

As mentioned above, a single fish is often limited both in capabilities and movement range. It will be incompetent for many complex tasks in dynamic environments. In this case, a multi-robot fish cooperative system becomes a desired solution. Inspired by the technology of multi-agent system and the approaches developed for cooperation of ground mobile robots, we establish the hardware platform of MRFS as depicted in Fig 4. The whole system can be decomposed into four subsystems: robot fish subsystem, image capturing subsystem,

decision making and control subsystem and wireless communication subsystem.

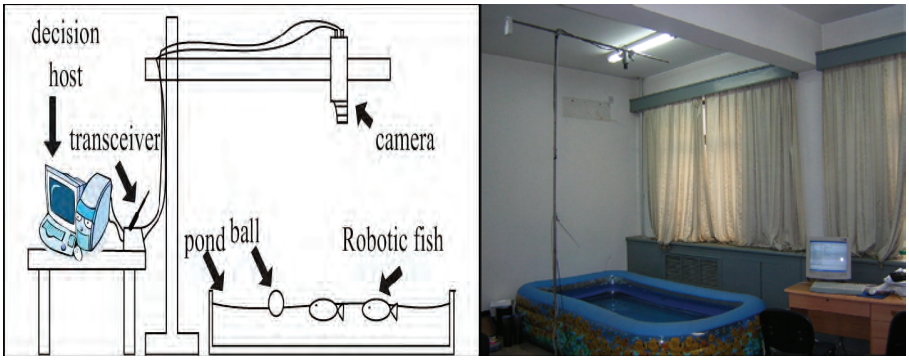


Fig. 4. Hardware platform for MRFS

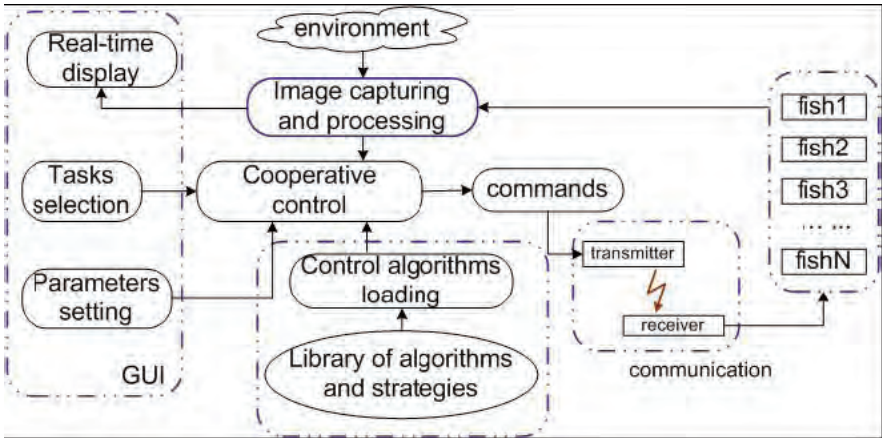


Fig. 5. Architecture of software platform

The information about fish and their surroundings are captured by an overhead camera and after being effectively processed, they are sent to the decision making and control subsystem as inputs. Then, based on input signals and specific control strategies for different tasks, the decision making subsystem produces corresponding control commands and transmits them to every robotic fish through the wireless communication subsystem. Since a global vision is adopted, MRFS should be basically categorized into centralized control system and so global planning and optimization can be obtained as a result. Based on the hardware architecture, we also develop a task- oriented software platform, on which we can implement various functions associated with cooperative task, such as task selection, environmental parameters setting, real-time display, image processing, control algorithm loading and commands executing. Fig 5 shows the schematic diagram of the software system architecture. It consists of GUI (Graphics User Interface), image processing module, algorithm module, communication module and fish module. Through GUI, users can choose different tasks, set

parameters of environment (goals, obstacles, etc.). In image processing module, a global image information is captured and processed. After that, useful information is abstracted and used for making decision. Algorithm module contains the algorithms and strategies, determining how the fished cooperate with each other. Communication module transmits every control command to the fish module which are the actuators of MRFS.

3.2 Hierarchical Control Algorithm for Cooperative Application

In this subsection, we propose a hierarchical control algorithm for cooperative application on MRFS. A four-level hierarchical architecture is developed: The first level is task planner level. In this level, the required task is decomposed into different roles. During the decomposition, it should be guaranteed that these roles are necessary and sufficient for achieving the task. After producing different roles we should select the most qualified candidate of robotic fish for each role according to some proper rules. Aiming at requirements of different tasks, we introduce both static and dynamic role assignments mechanism. In static assignments, once roles are determined at the beginning of the task, they will not change during the task; while in the dynamic mechanism, the robotic fish may exchange their roles according to the progress of the task. The third level is the action level. In this level, a sequence of actions are designed for each role. By action, we mean an intended movement of fish, such as turning, advancing, and so on. The fourth level, which is called the controller level, is the lowest one. In this controller level, we give a sufficient consideration to some unfavourable factors when control due to the speciality of fish.

– When the robotic fish swims, the interaction between it and its surrounding water will result in resonance at certain frequency. Moreover, the robotic fish can't stop immediately even if the oscillating frequency is set to zero. Hydrodynamic forces and the fish's inertia will make it drift a short distance along its advancing direction.

– In our design, the robotic fish's orientation is controlled by modulating the first two joints' deflection (φ1,φ2). However, it is quite difficult to adjust the deflection accurately, because the drag force produced by surrounding water is an unstructured disturbance and we can't get its precise model.

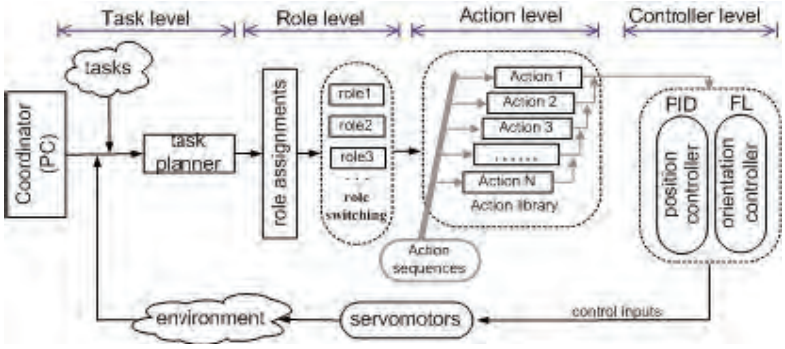
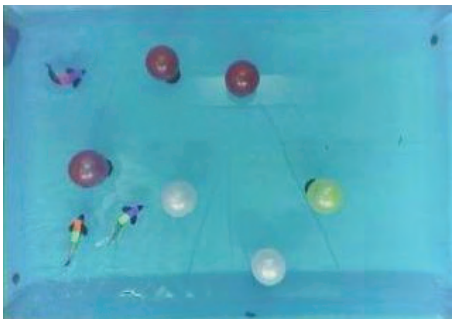


Fig. 6. The block diagram of the hierarchical architecture

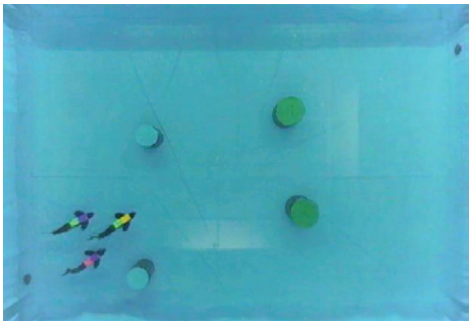
Based on the above conditions, we adopt a PID controller for piecewise speed control and a

fuzzy logic controller for orientation control, which are presented particularly in our previous work [3]. Figure 6 illustrates the block diagram of cooperative control architecture consisting of four levels.

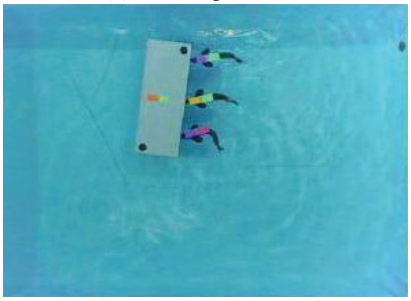
This section concentrates on the multi-robot fish cooperation problem. It describes the design and implementation of MRFS. It has been tested that MRFS provides a useful and effective platform to design and achieve cooperative tasks for multiple robot fish. Fig.7 shows the tasks designed on the basis of MRFS:



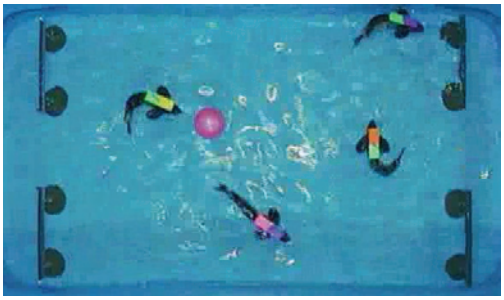
(a) Blasting collaboration



(b)Muti-fish Formation



(c)Pushin Box



(d)Robotic Water Polo

Fig. 7. The cooperating tasks on MRFS

4. The Collision Avoidance of Robotic Fish in RWP

As we know, robot navigation is concerned with driving a robot to the destination along a collision free path to perform a given task. In general, the navigation methods can be classified into two categories: global navigation, which is based on a prior known environment information; reactive navigation, which is based on real-time sensory information. Global navigation methods include roadmap approach, cell decomposition approach, artificial potential field approach, electrostatic potential field and the magnetic field method, etc. These methods can be understood well from the theoretical aspect, but the computational cost is expensive, so they are not applicable in dynamic environments. In reactive navigation, the robot makes decision on the real time sensory information. The low computational cost allows this approach suitable to be used in unknown or dynamic

environment. Examples of this approach include: Potential Fields [9], Vector Field Histogram [10], behavior-based method [11], fuzzy logic [12] and the Nearness Diagram Navigation [13]. Reactive navigation approach is effective in many dynamic scenarios, but most research on this method considers the robot as a point with holonomic properties. For the robotic fish we used in RWP, due to the particular propulsive mechanism, the translational velocity and the rotational velocity are not independent but coupled with each other. In addition, the robotic fish cannot move reversely or stop immediately in the water environment due to the inertial drift. Because of the particular kinematic characteristics of robotic fish and the complexity of the underwater environment, few navigation methods in the literature can be applied to robotic fish navigation directly. New navigation methods are required to be exploited for the robotic fish.

In this section, we propose a new reactive collision avoidance method for robotic fish navigation in the environment of RWP. Considering the nonholonomic properties and the inherent kinematic constraints of the robotic fish, limit cycle approach is proposed, with which the robotic fish can avoid one another smoothly and efficiently. Experiments performed by three robotic fish demonstrate the effectiveness of the proposed method.

The section is organized as follows. In subsection 2.1, the simplified propulsive model of the robotic fish is presented. In subsection 2.2, we describe our collision avoidance method in detail. Experimental results are given in subsection 2.3. subsection 2.4 concludes this section.

4.1 Simplified Propulsive Model of The Robotic Fish

Our designed robotic fish takes carangiform movement. Barrett et al. has presented a relative swimming model for RoboTuna (carangiform) in [6], and the undulatory motion is assumed to take the form of a propulsive travelling wave which is described by:

$$ybody(x;t) = [(c1x+c2x^2)][\sin(kx+wt)]: \quad (1)$$

In (1), *ybody* is the transverse displacement of the fish body, x the displacement along the main axis, k the body wave number ($k = 2\pi / \lambda$ □), λ the body wave length, $c1$ the linear wave amplitude envelope, $c2$ the quadratic wave amplitude envelope, and ω the body wave frequency ($\omega = 2\pi f$).

For simplification, we consider the discrete form of travelling wave (1), which is described by:

$$ybody(x;i) = [(c1x+c2x^2)][\sin(kx \pm \frac{2\pi}{M} i)] \quad (2)$$

where i denotes the index of the sequences, M is the body wave resolution, representing the discrete degree of the overall travelling wave.

4.2 Coordinated Collision Avoidance

(a) Kinematic Constraints of The Robotic Fish

Due to the particular propulsive mechanism, the translational velocity v and the rotational velocity of the robotic fish are not independent, but coupled with each other. Typical coupling relations of them under different oscillatory frequencies of the tail are shown in Fig. 8. We use the following equation to describe the relations:

$$w = F(v, frequency) \tag{3}$$

Equation (3) is called *inherent constraints* of the robotic fish. Any control input pair of $(v\ \omega)$ shall satisfy the *inherent constraints*.

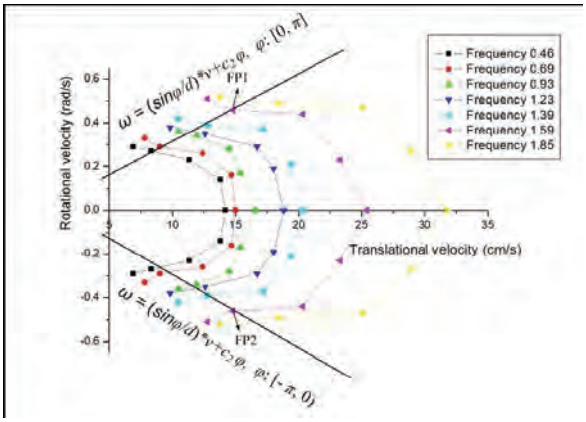


Fig. 8. Relations between the translational velocity and the rotational velocity under different oscillatory frequencies

(b) Limit Cycle Approach for Collision Avoidance

Next we discuss how collisions among multiple robotic fish are avoided. For simplification, The shape of the fish is described as an ellipse in the two dimensional plane. First we give one important lemma.Poincar´e-Bendixson Theory:

If D is an annulus-shape bounded absorbing region,, $D \subset R^2$ *, and contains no equilibria, then D contains at least one periodic orbit.* (A bounded region *D* on the phase plane is absorbing if no trajectories leave it.)

(c) Limit Cycle Approach

Considering the following nonlinear system in the fish frame:

$$\begin{aligned} \dot{x} &= \lambda(\mu y + \gamma x(r^2 - x^2 - \mu^2 y^2)) \\ \dot{y} &= \lambda(-\frac{1}{\mu} x + \gamma y(r^2 - x^2 - \mu^2 y^2)) \end{aligned} \tag{4}$$

where $\gamma\ \lambda\ \mu\ r$ are positive parameters. In order to prove the ellipse $x^2 + \mu^2 y^2 = r^2$ is the limit cycle of system (4), we use the following Lyapunov function:

$$V(x, y) = x^2 + \mu^2 y^2; \tag{5}$$

such that:

$$\begin{aligned} \dot{V}(x,y) &= 2\lambda\gamma(r^2 - x^2 - \mu^2 y^2)(x^2 + \mu^2 y^2) \\ &= 2\lambda\gamma(r^2 - V(x,y))V(x,y) \end{aligned} \tag{6}$$

We can see $\dot{V}(x,y) < 0$ when $V(x,y) > r^2$, while $\dot{V}(x,y) > 0$ when $V(x,y) < r^2$. This

shows the following annulus-shape region

$$B = \{a1 \le V(x, y) \le a2, |0 < a1 < r^2, a2 > r^2\} \tag{7}$$

is absorbing. It is also bounded and free of equilibrium points, since the unique equilibrium point is (0;0). According to Poincar´e-Bendixson Theory, *B* contains at least one periodic orbit. Since this argument is adaptive for any $0 < a1 < r^2$ and $a2 > r^2$, when □a1, □a2 get close to r^2, region *B* shrinks to the ellipse $V(x, y) = r^2$. Thus we get the limit cycle in the fish frame (shown in Fig. 9):

$$V(x, y) = r^2 \Rightarrow x^2 + \mu^2 y^2 = r^2 \tag{8}$$

The convergence speed of (*x*, *y*) toward the limit cycle can be tuned by the constant γ.Fig. 9 (a) shows the fast convergence condition with γ = 0.001, and Fig. 9 (b) shows the slow convergence condition with γ = 0.0001. From Fig. 9 we can see the trajectory from any point (*x*, *y*) moves toward and converges to the limit cycle clockwise when close. The counterclockwise condition can be derived by the following system (shown in Fig. 10):

$$\begin{aligned} \dot{x} &= \lambda(-\mu y + \gamma x(r^2 - x^2 - \mu^2 y^2)) \\ \dot{y} &= \lambda(\frac{1}{\mu} x + \gamma y(r^2 - x^2 - \mu^2 y^2)) \end{aligned} \tag{9}$$

Since the trajectory from any point (*x*, *y*) inside the limit cycle moves outward the cycle (thus away the center point of the cycle), and the trajectory from any point (*x*, *y*) outside the limit cycle approaches the cycle by aparting the center point distances (determined by the cycle), the limit cycle provides a method for collision avoidance among multiple robotic fish. Shown in Fig. 12, the shape of the obstacle fish body is described by an ellipse, and the safe region is defined by a concentric safe ellipse. When another fish is in the safe region, by the limit cycle approach, it will move away the obstacle fish toward the safe ellipse. Let θ denote the orientation of the fish, (*x*0, *y*0) the center point of the fish. With the following transformation we get the expression of system (4) in the original frame:

$$\begin{aligned} x &= \cos\theta(x + x0) - \sin\theta(y + y0) \\ y &= \sin\theta(x + x0) + \cos\theta(y + y0) \end{aligned} \tag{10}$$

such that:

$$\begin{aligned} \dot{x} &= \cos\theta\, \dot{x} - \sin\theta\, \dot{y} \\ \dot{y} &= \sin\theta\, \dot{x} + \cos\theta\, \dot{y} \end{aligned} \tag{11}$$

Let *v* denote the translational velocity of the fish in the original frame, a the direction of the motion. The kinematic model of the robotic fish is described by:

$$\begin{aligned} \dot{x} &= v \cos a \\ \dot{y} &= v \sin a \\ \frac{\dot{y}}{\dot{x}} &= \tan a \end{aligned} \tag{12}$$

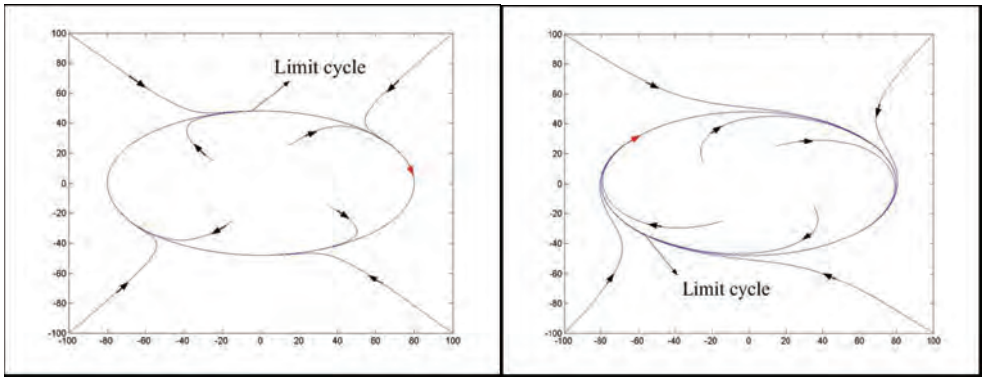
Substituting (11) into (12) we get:

$$\begin{aligned} v &= \sqrt{\dot{x}^2 + \dot{y}^2} \\ a &= \arctan \frac{\dot{y}}{\dot{x}} + \theta \end{aligned} \tag{13}$$

By tuning the value of λ, we can adjust the magnitude of v to get arbitrary speed values. Advantages of the limit cycle approach are listed below:

1) Since the control inputs of the robotic fish are the translational velocity v and the orientation angle a instead of the rotational velocity ω, we elimitate the trouble of treating the tackled coupling between v and ω in collision avoidance.

2) It is an efficient reactive collsion avoidance approach, and the fish can avoid the obstacle with the safety distances and appropriate direction without moving far away from the obstacle.



(a) Fast convergence with γ = 0.001 (b) Slow convergence with γ = 0.0001

Fig. 9. Phase portrait of limit cycle (clockwise)

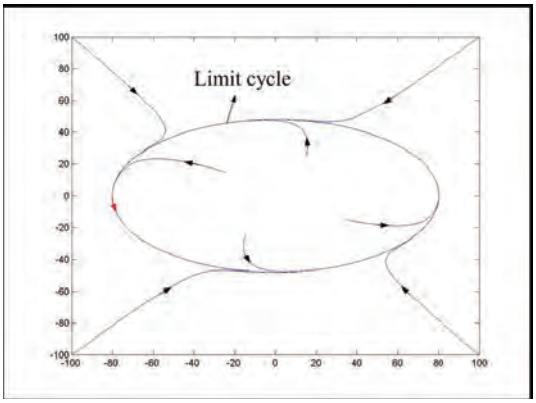


Fig. 10. Phase portrait of limit cycle (counterclockwise)

(d)Applying Limit Cycle Approach in Multiple Robotic Fish Collision Avoidance

Next we employ three robotic fish for discussion. To apply the limit cycle approach in the coordinated collision avoidance, we employ the .situated-behavior. method (similar to the *situated-activity* paradigm (see [14])) to divide the environment into a set of *exclusive* and *complete* situations, and for each situation, a behavior is elaborately designed to solve the situation associated problem individually. The advantage of employing this method is that it is a .divide and conquer. strategy, which reduces the task difficulty; in addition, the real-time *behavior coordination problem* need not to be taken into consideration, since the situations are complete and exclusive.

Situations Here we discuss only the situations for fish 1. The situations for other fish are similar. We use a decision tree to define the set of situations according to the relative locations of the robotic fish. The decision tree is traversed through binary decision rules according to several criteria. As shown in Fig. 11, the inputs of the decision tree are the goal location information and sensory information from the overhead camera, including the ID information, the location and orientation information of the fish. The current

situation is identified according to the input information. The decision tree is traversed through binary decision rules according to the following four criteria.

Criterion 1: Obstruction criterion. This criterion classifies the situations into the following two categories according to whether fish 1 is obstructed by another fish:

(1) Nobs (not obstructed) situation: Fish 1 is not obstructed by any other fish;

(2) OBS (obstructed) situation: otherwise.

Criterion 2: Obstacle fish number criterion. This criterion divides OBS situation into the following two situations:

(1) Sobs (single obstructed) situation: Fish 1 is obstructed by only one fish;

(2) Tavoid (trap avoiding) situation: Fish 1 is obstructed by other two fish simultaneously (shown in Fig. 13 (d)).

Criterion 3: Dual avoiding criterion. This criterion classifies Sobs situation into the following two situations according to whether fish 1 and the other fish are obstructed by each other:

(1) Savoid (single avoiding) situation: Fish 1 is obstructed by the other fish but not an obstacle for that fish (shown in Fig. 13 (a));

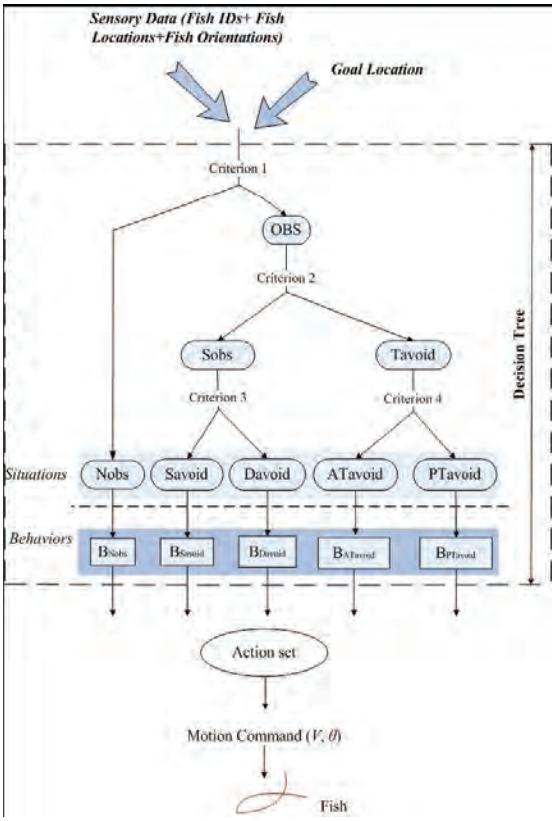


Fig. 11. Situations of coordinated collision avoidance

(2) Davoid (dual avoiding) situation: Fish 1 and the other fish are obstructed by each other (shown in Fig. 13 (b) (c)).

Criterion 4: Active avoiding criterion. According to whether fish 1 has the highest priority among fish in Tavoid situation (here the priorities of the fish are arranged according to their IDs), this criterion divides Tavoid situation into the following two situations:

(1) ATavoid (active trap avoiding) situation: Fish 1 has the highest priority;

(2) PTavoid (passive trap avoiding) situation: Otherwise.

We only care the leaf nodes of the decision tree: Nobs, Savoid, Davoid, ATavoid, PTavoid. Obtained through a binary decision tree, these five situations are exclusive and complete.

Behaviors associated with the situations

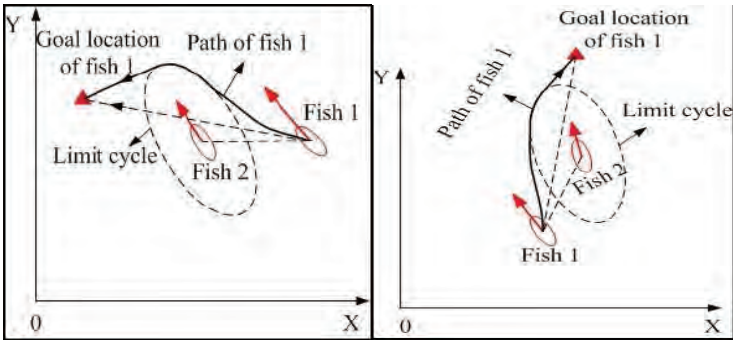
First we define a *normal behavior*.

normal behavior: If the goal direction is to the right of the obstacle direction, and the path of the fish to the goal location is obstructed by the safe ellipse of the obstacle fish, shown in Fig.

12 (a), the fish moves toward the limit cycle counterclockwise; while if the goal direction is to the left of the obstacle direction, and the path of the fish to the goal location is obstructed by an obstacle fish, the fish moves toward the limit cycle clockwise, shown in Fig. 12 (b).

1) *BNobs*: Fish 1 approaches the goal location directly.

2) *BSavoid* : Fish 1 avoids the obstacle fish with *normal behavior* (shown in Fig. 13(a)).



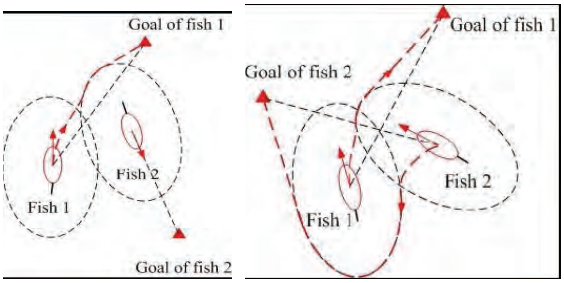
(a) Avoiding fish counterclockwise (b) Avoiding fish clockwise

Fig. 12. Normal behaviour

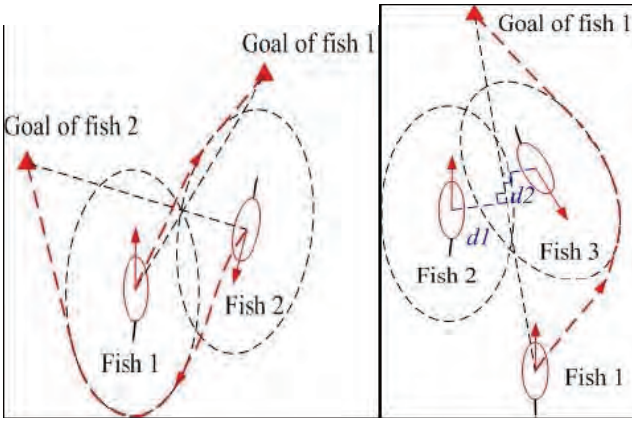
3) *BDavoid*: The two fish avoids each other with the same limit cycle direction (both clockwise, or both counterclockwise). If fish 1 is prior to the other fish, it avoids the other fish with *normal behavior*; otherwise it avoids the other fish with the direction opposite to *normal behavior* (shown in Fig. 13 (b) (c)).

4) *BATavoid* : In Tavoid situation, it is possible for fish 1 to get stuck in a local minima with *normal behavior*. For example, in Fig. 13 (d), fish 1 is avoiding fish 2 counterclockwise; however, since it is also obstructed by fish 3, then it will avoid fish 3 clockwise. Thus fish 1 will get stuck between fish 2 and fish 3. *BATavoid* is to avoid this situation. Let *l* denote the line from fish 1 to Goal of fish 1, and fish *i* (*i* 2 (2;3)) represent the fish with the shorter distance to line *l*. Fish 1 avoids fish *i* with direction opposite to *normal behavior*, until the local trap situation is eliminated (shown in Fig. 13 (d)).

BPTavoid : Fish 1 avoids the fish in ATavoid situation with the same limit cycle direction to overcome confiicts.



(a) Savoid situation example (b) Davoid situation example 1



(c) Davoid situation example 2 (d) ATavoid situation example

Fig. 13. Situations and associated behavior design

4.4 Conclusion for Collision Avoidance

We have presented a novel collision avoidance method for multiple robotic fish. Considering the inherent kinematic constraints of the robotic fish, we employ limit cycle approach for coordinated collision avoidance among the robotic fish. This approach allows the robotic fish to avoid one another smoothly and efficiently, and also eliminates the local minima problem. Experimental results demonstrate the effectiveness of the proposed method.

5. Situation Based Action Selection and Design for Pushing Ball in RWP

In this section, we will present the situation-base action selection mechanism to achieve ball pushing in RWP. It seems that pushing ball is quite simple, one robotic fish may be enough to finish it, in fact, the robotic fish's head is only 40mm wide and the ball drifts with the fluctuant water, it is not easy for one robotic fish to touch the ball exactly in the expected point and push it to the direction of the gate. Moreover, because of its inertia and hydrodynamic forces, the robotic fish can't stop immediately even if the oscillating frequency is set to zero. It will drift a short distance along its current direction and thus, overshot occurs. When this happens, the fish must turn back and re-adjust its relative position with respect to the ball, which will take a long time. In addition,when the fish swims back and adjusts its attitudes, it will inevitably disturb the surrounding water. As a result, the ball may float away before the fish approaches it, which will increase difficulty to the game.

Based on the above consideration, multiple fish are used to push the ball to goal cooperatively. For example, we can allow two or three to push the ball in different points, using composition of their forces to move the ball towards the gate. Or if a fish overshoot

the ball, it can hit the ball by its tail and make the ball fioat to a favorable position for another teammate.

5.1 Working Region, Situations and Rules for the Game

To describe the situations for the game, we first introduce three working regions for the robotic fish. As shown in Fig. 14, *l* is a line connecting the center of the ball to the goal. Let us draw a circle at the center of the ball with a radius *r* and then partition it into four vectors. There obtain five working regions: Pushing Region, Left Assistant Pushing Region, Right Assistant Pushing Region and Overshot Region. Other regions out of the circle is called Buffer Region.

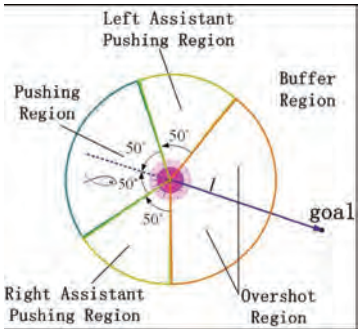


Fig. 14. Illustration of the region division

- the Pushing Region (PR): this is an effective working region, since in this region the fish-like robot can push the ball to the direction of the goal and so the pushing action is effective.
- the Left Assistant Pushing Region (LAPR): this region is a semi-effective working region. The fish in this region can not directly push the ball to the desired direction, but it can help a fish in PR prevent the ball floating away.
- the Right Assistant Pushing Region (RAPR): similar to LAPR, this is also a semi-effective working region
- the Overshot Region (OR): this region is a forbidden working region. When the fish swims into this region, it will be located between the ball and the goal. In this region, the fish are forbidden to touch the ball because it will push the ball to the opposite direction to the destination. In this case, the fish should first swim around the ball and enter a effective working region and then select suitable actions.
- the Buffer Region (BR): In this region, the fish is a little bit far from the ball, in will try to swim to an effective region (PR, LAPR or RAPR) first.

We use multiple fish in the RWP in order to enhance the efficiency of the game. However, when the number of the fish increase, more problems may occur. Because the space is limited, these fish may collide with each other. In order to deal with such circumstance, the collision avoiding strategies are designed, which will debase the performance of cooperation.Moreover, when multiple fish swim in the same place, the surrounding water

will be disturbed violently. The ball will float everywhere and become more difficult to track and push. Additionally, waves produces by fish and ball may cause uncertainty and inaccuracy to the image information captured by the overhead camera..Considering the characteristics of the fish's dynamics and the speciality of the hydro-environments, we propose the following rules for this game.

Rules 1: as shown in Fig. 15 (a), we divide the space (pond) into three parts in Y coordinates: Part A, Part B and Part C, representing the respective region each fish take charge of. We use such kind of region-responsibility strategy with main intent to avoid collisions between these fish. For each fish, if the ball is not in the region that it is responsible for. It will still swim restrictively within its region and cannot invade other regions to disturb its teammates. This rule is inspired by the human soccer match in which different roles take charge of different regions.

Rules 2: we prefer slowness than overshoot when controlling the fish. We intent to slow down the fish to some extent, especially when it approaches the pushing point. In addition, the distance from the pushing-point to the center of the ball is defined to be a value larger than theoretical value.

Rules 3: as we mentioned, the fish can't stop immediately even you send stop command. It will drift a short distance out of control. So, in practice, even when the fish is idle, we will let the fish wander or move very slowly instead of using stop command to halt it.

Obviously, based on the above rules and according to the number of fish entering the Pushing Region, we can define four primary situations for the task.

Situation 0 (S0): no fish is in the Pushing Region. In this situation, the game can not be implemented immediately. The fish should first swim entering the effective working region.

Situation 1 (S1): there is only one fish-like robot in the Pushing Region. In order to effectively push the ball, the fish should touch the ball exactly to the direction of the goal. In this situation, according to the number of fish in the Assistant Pushing Regions we can define three sub-Situations:

- S1----APR0: there is no fish in the Assistant Pushing Regions
- S1----LAPR1: there is one fish in the Left Assistant Pushing Regions, so this fish will assist the main pusher to transport the ball.
- S1----RAPR1: there is one fish in the Right Assistant Pushing Regions.
- S1----APR2: two fish are in the Assistant Pushing Regions, one is left and one is right.

Situation 2 (S2): there two fish are in the pushing region. At this circumstance, these two fish can push the ball cooperatively. They push the ball from different points and let the combination of their force move the ball to the destination. Similarly, we can define two sub-Situations:

- S2----APR0: there is no fish in the Assistant Pushing Regions
- S2----LAPR1: there is one fish in the Left Assistant Pushing Regions, so this fish act as an assistant in the game.
- S2----RAPR1: there is one fish in the Right Assistant Pushing Regions.

Situation 3 (S3): all the fish are in the pushing region. In this situation, these fish can have a work division. Some are responsible for pushing ball, others can assistant the pusher by preventing the ball from floating away with the fluctuant water.

Different situations will result in different task difficulty. As we mentioned above, when there is only one fish pushing the ball, it may easily miss the ball or push it to the wrong direction. While, as the pushers increase, the game will be easier due to cooperation and redundancy. So, in order to push the ball in a more stable and precise way, multiple fish are expected to help each other and work in a cooperative way, such as in S2 and S3.

5.2 Situated Actions Design

In contrast to controlling some ground mobile robots, the robotic fish's attitudes are more difficult to modulate, because it is very difficult to establish any precise dynamics models for the robotic fish. Hence, we designed the fish's primitive actions based on geometric approach:

Action 0: This action is designed for the fish to swim from a non-pushing region to the effective working region. Basically, this is a simple *Point----To----Point*(*PTP*) control.

Action 1: As depicted in Fig. 15 (b), the first action for the fish is to swim approaching the ball and hit the ball exactly along the direction from the ball to the gate. where (x_F , y_F , α □) denotes the pose of the fish, (x_D ; y_D) and (x_C ; y_C) stand for the center of the ball and the position of the gate, β is the expected direction from the ball to the gate, l_1 indicates the expected moving direction of the ball. Considering the fish's bodylength and its inertia, We choose a point *G* (x_G ; y_G)which located at the extended line of l_1 as the pushing-point. l_2 is the section connecting the fish to *G*, and l_3 represents the perpendicular of l_1 which pass through point *G*.

As illustrated in Fig. 15 (b), if the pushing-point *G* locates between the fish and the ball, that's $(x_D - x_F) \times (x_C - x_D) > 0$, we first define the perpendicular bisector l_4 for section l_2. Then using *r* as radius, we make a circle *C* which is tangental to l_1 at point *G*. If the circle intersects l_4 at one point, we chose this point as a temporary goal for the fish, or if they are two intersections, we chose the point with small x-coordinates, namely *T* as the temporary goal. While, if the fish is far away from the ball and there is no intersection between *C* and l_1, *G* will be the temporary goal point for the fish. As the fish moves, a series of temporary goals will be obtained, which will lead the fish swim gradually to the pushing-point.

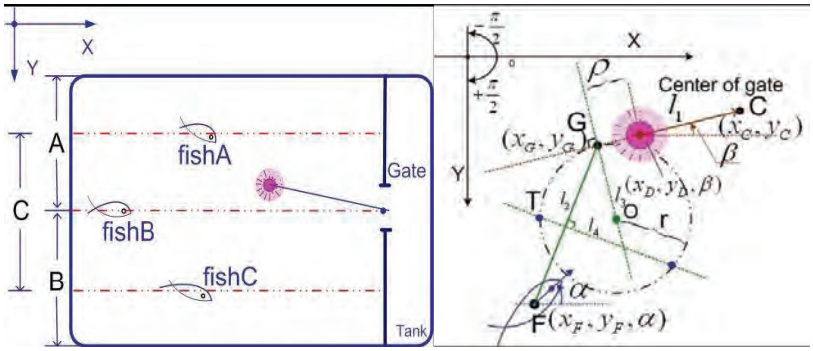


Fig. 15. (a)The experiment setting and Region division (b)The action of pushing ball along the exact direction pointing the gate

After simple geometrical analysis, The positions of intersection points can be calculated by the following equation(14):
equation:

$$\begin{aligned}&(x - x_D - \rho\cos\beta - r\sin a)^2 + (y - y_D - \rho\sin\beta + r\cos a)^2 = r^2\\ &y - \frac{1}{2}(y_G + y_F) = \frac{x_F - x_G}{y_G - y_F}(x - \frac{1}{2}(x_G + x_F))\end{aligned} \tag{14}$$

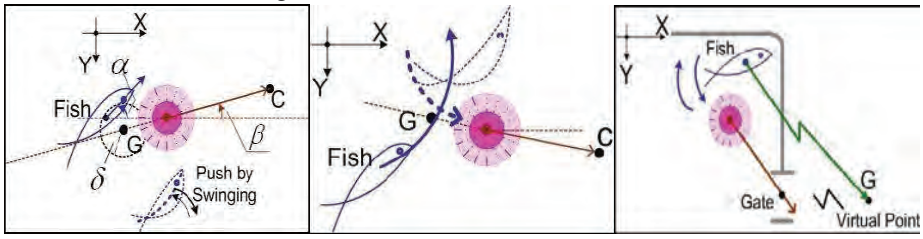
Action 2: Although when determining the pushing-point, we give sufficient consideration for the dynamics of the fish and the difficulty when controlling it, we still can't guarantee the fish will reach its destination in the expected attitudes, especially its orientation. Once it gets to the pushing-point with large orientation error, it may possible miss the ball. In this case, we design the following action which allow the fish to push the ball by swing.
As shown in Fig. 16 (a), if the fish approaches the pushing-point (in a small neighbor region) and its orientation satisfies the following condition, it will take a sharp turn to the direction of the ball.

$$\begin{cases}(x_F, y_F) \in \{\|(x_G, y_G) - (x_F, y_F)\| \le \delta\}\\ \alpha \in \{|\alpha - \beta| \ge \xi\}\end{cases} \tag{15}$$

where δ and ξ are the bounds for position error and orientation error, which are determined empirically experiments. In our experiment, we choose $\delta = 5cm$ and $\xi = \frac{\pi}{15}$

Action 3: This action is an assistant action, which is implemented in LAPR or RAPR. In particular, this action takes full advantage of the agility of fish's tail. Fig. 16 (b) indicates the fish pat the ball by its tail.

Action 4: Action 4 will be implemented when the ball fioats very close to the gate and to the edge of the pond, and the fish is in LAPR or RAPR. This action allows the fish to move the ball slowly towards the goal by oscillating. As shown in Fig. 16 (c), in this action, we design a point outside the pond as a *virtual pushing--- point*. Thus the fish will always try to swim to that virtual destination, although it can never reach it. During moving (or struggling), its body, especially posterior body, oscillates continually, which disturbs the water and makes the ball fioat towards the gate.



(a) (b) (c)

Fig. 16. The fish pushes ball by (a) shaking(throwing) head, (b)pushing the ball by tail and (c) swimming towards a virtual pushing point respectively

Next, we will investigate how the fish work cooperatively to push ball based on the above four basic actions. Three behaviors for cooperation are designed, two for two fish and one for three.

Action 5: This action is designed for S2. As shown in Fig. 17 (a), for a given pose of the ball, instead of

choosing one pushing-point, two goal points (PushPointL and PushPointR) are defined which locate at different sides of pushing-point with a defiection φ. Two fish swim towards different pushing points, and the combination of their pushing force will make the ball fioat towards the gate.

Action 6: When in **S3** we adopt the following cooperative action, as shown in Fig. 17 (b). That's three fish surround the ball and sent it into the gate. At this circumstance, FishC push the ball exactly to the direction of the goal, while FishL and FishR hit the ball from PushPointL and PushPointR respectively.

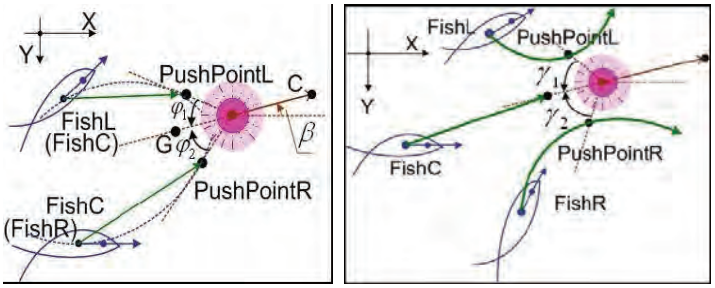


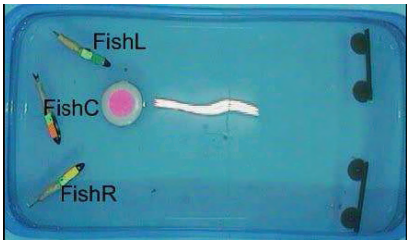
Fig.17. (a)Cooperative action for two fish. (b) Cooperative action for three fish

Table 1 sums up the corresponding strategies for each fish individual when in different situations:

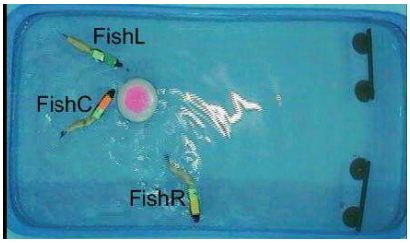
Situation	Multi-robot fish		
	FishL	FishC	FishR
S0	Action 0	Action 0	Action 0
S1-APR0	Action 1,2	Action 0	Action 0
	Action 0	Action 1,2	Action 0
	Action 0	Action 0	Action 1,2
S1-LAPR1	Action 3,4	Action 1,2	Action 0
	Action 0	Action 3,4	Action 1,2
S1-RAPR1	Action 0	Action 1,2	Action3,4
	Action 1,2	Action 3,4	Action 0
S1-APR2	Action3,4	Action1 ,2	Action 3,4
S2-APR0	Action 0	Action 1,2	Action 1,2
	Action 1,2	Action1,2	Action 0
S2-LAPR1	Action 3,4	Action 5	Action 5
S2-RAPR1	Action 5	Action 5	Action 3,4
S3	Action 6	Action 6	Action 6

Table 1. Situation Based Action Selection

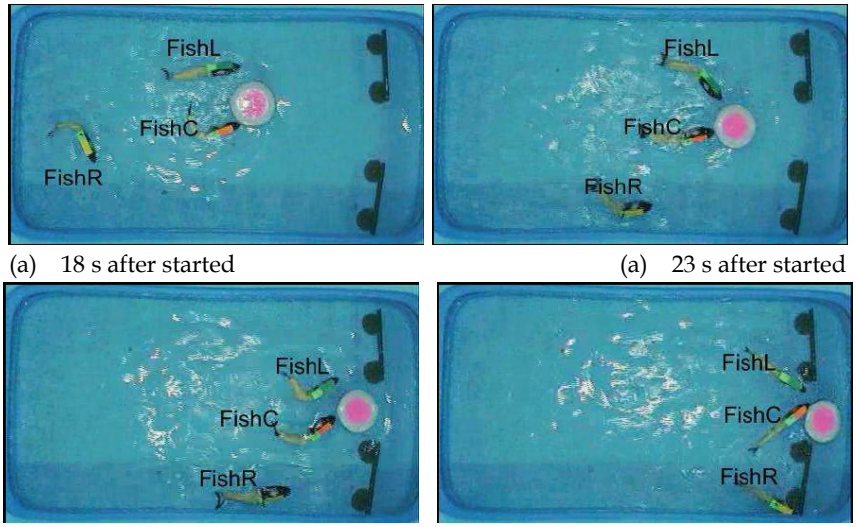
Fig. 18 shows the scenarios in one of the experiments. The experiment is quite successful and has high efficiency, since during the pushing, FishL and FishR cooperative very well and neither of them overshoots the ball. Moreover, the ball is relatively stable and there is little disturbance when it floats. In this sense, the hydro-environment is more complicated than general ground environment.



(a) 0 s after started



(a) 6 s after started



(a) 18 s after started (a) 23 s after started

(a) 24 s after started (a) 27 s after started

Fig. 18. Fragment of the experiment, Sequences of overhead images, order is left to right and top to down. (a) The initial state of the experiment. (b) FishL is in the LAPR, FishC in PR attempts to push the ball directly and FishR tries to approach the ball but turns when swims out of its charged region. (c) FishL takes Action 2 to pat the ball and at the same time FishC takes Action 1 to hit the ball. FishR still tries to approach the object. (d) Both FishL and FishC take Action 5 to push the ball cooperatively. FishR takes Action 0. (e) FishL takes Action 3, FishC takes Action 1 and FishR still takes Action 0. (f) FishL takes Action 3, FishC takes Action 1 and FishR takes Action 4

6. Conclusions and Foregrounds

We focus this chapter on the RWP and the underwater robot cooperation involved in the game, two useful methods are introduced in this chapter for the game. These two methods have been certified by the experiments.

Mankind invented the robot in the 20th century and realized the use of robots initially. In the 21st century, humans will coexist with robots. Robot Football is the ideal model to research the future sociality.

A very wide range of fields are involved in the RWP, including machinery and electronics, robotics and sensor information fusion, intelligent control, communications, computer vision, computer graphics, artificial intelligence and so on. What is more significant is that the RWP makes the research and education combined. The game combined ingeniously with the academic research will furthermore inspire the young students interesting strongly. It will train young students to have rigorous scientific attitude and good skills through the competition of RWP.

Though RWP has only just started, the significance of industry research and the huge potential has already been demonstrated:

(1) It provides a high-tech display of the outcome of the visualization window to promote scientific technology and social production and living closer integration. Soccer is not only one of the world's most popular sports, but also a huge industry. In the current world, the research on robots always consciously takes the initiative to integrate into the community's economic and cultural life.

(2) The RWP provides an approach of converting research results to industry. It can be expected that with the development of the RWP, variety of new underwater robots will be developed, and the efficiency will grow rapidly. Also, with other types of robot soccer team and the technical aspects mature gradually, the related products will grow rapidly.

(3) RWP provides a standard platform to research on underwater robot collaboration. Due to the special nature of the underwater environment, many ground collaboration theory and algorithms can not be applied directly in the underwater environment, there are great difficulties in collaborative research on underwater robot. By providing a standard platform, it will attract more people to conduct underwater robot for underwater robot collaborative research.

RWP can also reflect comprehensive strength of a national information technology and automation technology and at the same time multiple underwater robots cooperation have broad application prospects in a large of engineering field, for example, the cooperation control research of multiple underwater robot fish has a broad prospect in the military, the detection of underwater resources, on the sea detecting and rescue, marine multi-sensor sampling network, and other military tasks in which the cooperation control research plays an important role.

In short, the RWP not only is the high-tech competition, but also has the property of view and admire and entertainment like the football game. Therefore, there will surely attract a large number of "fans" .Currently, there are more and more schools invited to participate in RWP. We have the reason to believe that under the efforts of all colleges and universities, RWP will be promoted in the world to be the standards multi-robot competition.

7. References

J. Yu, S. Wang and M. Tan, .Basic motion control of a free-swimming biomimetic robot fish,. in Proc. IEEE Conf. Decision and Control, Maui, Hawaii USA, pp. 1268-1273, December 2003.

J. Yu, and L. Wang: Parameter optimization of simplified propulsive model for biomimeticrobot fish. In Proc. IEEE Int. Conf. Robotics and Automation. Barcelona, Spain (2005) 3317-3322

J. Yu, L. Wang, and M. Tan: A Framework for Biomimetic Robot Fish's Design and Its Realization. In Proc. of American Control Conference. Portland, USA (2005) 1593-1598

J. Yu, Y. Fang,W. Zhao, and L.Wang: Behavioral Design and Strategy for Cooperative Multiple Biomimetic Robot Fish System. In Proc. IEEE Int. Conf. Robotics and Biomimetics. Hongkong and Macau (2005) 472-477

Y. Fang, J. Yu, R. Fan, L. Wang, and G. Xie: Performance Optimization and CoordinatedControl of Multiple Biomimetic Robotic Fish. In Proc. IEEE Int. Conf. Robotics and Biomimetics. Hongkong and Macau (2005) 206-211

D. Barrett, M. Triantafyllou, D. K. P. Yue, M. A. Grosenbaugh, and M. J. Wolfgang: Dragreduction in fish-like locomotion. J. Fluid Mech. Vol. 392. (1999) 183-212

D. Barrett, M. Grosenbaugh, and M. Triantafyllou, .The optimal control of a fiexible hull robotic undersea vehicle propelled by an oscillating foil,. in Proc. IEEE AUV Symp. pp. 1-9. vol. 24 1999

M. Sfakiotakis, D. M. Lane, and J. B. C. Davies: Review of fish swimming modes for aquaticlocomotion. IEEE J. Oceanic Eng. Vol. 24. (1999) 237-252

N. C. Tsourveloudis, K. P. Valavanis and T. Hebert: Autonomous vehicle navigation utilizing electrostatic potential fields and fuzzy logic. IEEE Trans. Robotics and Automation. Vol. 17.(2001) 490-497

W. Tianmiao and Z. Bo: Time-varing potential field based .perception-action. behaviors of mobile robot. In Proc. IEEE Int. Conf. Robotics and Automation. (1992)2549-2554

J. Borenstein and Y. Koren: The vector field histogram-fast obstacle avoidance for mobile robots. IEEE Trans. Robotics and Automation. Vol. 7. (1991) 278-288 A. Taliansky and N. Shimkin: Behavior-based navigation for an indoor mobile robot. In The 21st IEEE Convention of Electrical and Electronic Engineers in Israel. (2000) 281-284

T.E. Mora and E.N. Sanchez: Fuzzy logic-based real-time navigation controller for a mobilerobot. In Proc. IEEE Int. Conf. Intelligent Robots and Systems. Vol. 1. (1998) 612-617

J. Minguez and L. Montano: Nearness Diagram (ND) Navigation: Collision Avoidance inTroublesome Scenarios. IEEE Trans. Robotics and Automation. Vol. 20. (2004) 45-59

R. C. Arkin: Behavior-Based Robotics. Cambridge, MA: MIT Press (1999)

Triantafyllou, M. S., Triantafyllou,G. S.: An efficient swimming machine. Sci. Amer. 272 (1995)64-70

Terada, Y.: A trial for animatronic system including aquatic robots, J. Robot. Soc. Jpn. 18(2000)195-197

Domenici, P., Blake, R. W.: The kinematics and performance of fish fast-start swimming,J. Exper. Biol., 200 (1997)1165-1178

M. S. Triantafyllou and G. S. Triantafyllou : An efficient swimmingmachine. Sci. Amer. vol. 272 pp. 64-70, 1995

M. J. Lighthill, .Note on the swimming of slender fish,. J. Fluid Mech., vol. 9, pp. 305-317, 1960.

D. Barrett, M. Triantafyllou, D. K. P. Yue, M. A. Grosenbaugh, and M. J. Wolfgang,.Drag reduction in fish-like locomotion,. J. Fluid Mech., vol. 392, pp. 183-212, 1999.

Y. Terada,. A trial for animatronic system including aquatic robots., J. Robot. Soc. Jpn. vol. 18 pp. 195-197 2000

T. Yuuzi and I. Yamamoto An Animatronic System Including Lifelike Robotic Fish pp. 1814-1820

P. Domenici and R. W. Blake, .The kinematics and performance of fish fast-start swimming,. J. Exper. Biol., vol. 200, pp. 1165C1178, 1997.