

Lecture Notes in Artificial Intelligence 1395

Subseries of Lecture Notes in Computer Science

Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Hiroaki Kitano (Ed.)

RoboCup-97: Robot Soccer World Cup I



Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editor

Hiroaki Kitano
Sony Computer Science Laboratory
3-14-13 Higashi-Gotanda, Shinagawa
Tokyo 141, Japan
E-mail: kitano@csl.sony.co.jp

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

RoboCup <1, 1997, Nagoya>:

Robot Soccer World Cup I / RoboCup-97. Hiroaki Kitano (ed.). -
Berlin ; Heidelberg ; New York ; Barcelona ; Budapest ; Hong Kong ;
London ; Milan ; Paris ; Santa Clara ; Singapore ; Tokyo : Springer,
1998

(Lecture notes in computer science ; Vol. 1395 : Lecture notes in
artificial intelligence)
ISBN 3-540-64473-3

CR Subject Classification (1991): I.2, C.2.4, D.2.7, H.5, I.5.4, I.6, J.4

ISBN 3-540-64473-3 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer -Verlag. Violations are liable for prosecution under the German Copyright Law.

© Springer-Verlag Berlin Heidelberg 1998
Printed in Germany

Typesetting: Camera ready by author
SPIN 10636984 06/3142 - 5 4 3 2 1 0 Printed on acid-free paper

Preface

In the history of artificial intelligence and robotics, the year 1997 will be remembered as a turning point. In May 1997, IBM Deep Blue defeated the human world champion in chess. Forty years of challenge in the AI community came to a successful conclusion. On July 4, 1997, NASA's pathfinder mission made a successful landing and the first autonomous robotics system, Sojourner, was deployed on the surface of Mars. Together with these accomplishments, RoboCup made its first steps toward the development of robotic soccer players which can beat a human World Cup champion team.

RoboCup is an international initiative devoted to advancing the state of the art in AI and robotics. The particular goals of the project and potential research directions are numerous. The most ambitious and long range goal can be stated as:

to build a team of robot soccer players, which can beat a human World Cup champion team.

The accomplishment of the goal requires decades of extensive efforts and series of innovative technologies must be developed. In addition, a broad range of technologies need to be integrated. However, because of the difficulties of the challenges, and the potential breadth of the technological domains affected, research toward RoboCup's ultimate goal is expected to generate numerous spin-off technologies. For details of the goals and technical issues of RoboCup, see the article "RoboCup: A Challenge AI Problem" in this volume.

This preface outlines the progression of robotic soccer over the course of the past 5 years from a basis for research in a few individual laboratories scattered around the world to a full-blown international initiative.

The idea of robots playing soccer was first mentioned by Professor Alan Mackworth (University of British Columbia, Canada) in a paper entitled "On Seeing Robots" presented at VI-92, 1992, and later published in a book *Computer Vision: System, Theory, and Applications*, pages 1-13, World Scientific Press, Singapore, 1993. A series of papers on the Dynamo robot soccer project was published by his group.

Independently, a group of Japanese researchers organized a Workshop on Grand Challenges in Artificial Intelligence in October, 1992 in Tokyo, discussing possible grand challenge problems. This workshop led to a serious discussions of using the game of soccer for promoting science and technology. A series of investigation were carried out, including a technology feasibility study, a social impact assessment, and a financial feasibility study. In addition, rules were drafted, as well as prototype development of soccer robots and simulator systems. As a result of these studies, we concluded that the project is feasible and desirable. In June 1993, a group of researchers, including Minoru Asada, Yasuo Kuniyoshi, and Hiroaki Kitano, decided to launch a robotic competition, tentatively named the Robot J-League (J-League is the name of the newly established Japanese Professional soccer league). Within a month, however, we received overwhelming reactions from researchers outside of Japan, requesting that the initiative be

extended as an international joint project. Accordingly, we renamed the project as the Robot World Cup Initiative, "RoboCup" for short.

Concurrent to this discussion, several researchers were already using the game of soccer as a domain for their research. For example, Itsuki Noda, at ElectroTechnical Laboratory (ETL), a government research center in Japan, was conducting multi-agent research using soccer, and started the development of a dedicated simulator for soccer games. This simulator later became the official soccer server of RoboCup. Independently, Professor Minoru Asada's Lab. at Osaka University, and Professor Manuela Veloso and her student Peter Stone at Carnegie Mellon University had been working on soccer playing robots. Without the participation of these early pioneers of the field, RoboCup could not have taken off.

In September 1993, the first public announcement of the initiative was made, and specific regulations were drafted. Accordingly, discussions on organizations and technical issues were held at numerous conferences and workshops, including AAAI-94, JSAI Symposium, and at various robotics society meetings

Meanwhile, Noda's team at ETL announced the Soccer Server version 0 (LISP version), the first open system simulator for the soccer domain enabling multi-agent systems research, followed by version 1.0 of Soccer Server (C++ Version) which was distributed via the RoboCup's World Wide Web home page. The first public demonstration of this simulator was made at IJCAI-95.

During the International Joint Conference on Artificial Intelligence (IJCAI-95) held at Montreal, Canada, August, 1995, the announcement was made to organize the First Robot World Cup Soccer Games and Conferences in conjunction with IJCAI-97 Nagoya. At the same time, the decision was made to organize Pre-RoboCup-96, in order to identify potential problems associated with organizing RoboCup on a large scale. The decision was made to provide two years of preparation and development time, so that initial group of researchers could start robot and simulation team development, as well as giving lead time for their funding schedules.

Pre-RoboCup-96 was held during International Conference on Intelligence Robotics and Systems (IROS-96), Osaka, November 4 – 8, 1996, with eight teams competing in a simulation league and demonstration of real robots for middle size league. While limited in scale, this competition was the first competition using soccer games for promotion of research and education.

The official first RoboCup games and conference was held in 1997 with great success. Over 40 teams participated (real and simulation combined), and over 5000 spectators attended. As results of the game, AT-Humboldt (Humboldt University, Germany) became the World Champion for the simulator league, runner-up was AndHill (Tokyo Institute of Technology, Japan), the third place was ISIS (Information Science Institute / University of Southern California, USA), and the fourth place was CMUnited (Carnegie Mellon University, USA). For the small-size real robot league, CMUnited (Carnegie Mellon University, USA) became the World Champion, by beating the NAIST (Nara Advanced Institute for Science and Technology, Japan). The World Champion for the middle-size

league was awarded to Deamteam (Information Science Institute / University of Southern California, USA) and Trackies (Osaka University, Japan) because both the final game and a preliminary game ended in draws.

Apart from the winners of the competition, RoboCup awarded two Engineering Challenge Awards and one Scientific Challenge Award. RMIT Raiders (Royal Melbourne Institute of Technology, Australia) and Uttori United (a joint team of Institute of Physical and Chemical Research (RIKEN), Toyo University, and Utsunomiya University, Japan) received the Engineering Challenge Award for their innovative design of omni-directional driving mechanisms. The Scientific Challenge Award was given to Sean Luke (University of Maryland) for demonstrating the utility of an evolutionary computing approach by evolving a soccer team using genetic programming. These challenge award were created in order to encourage scientific and engineering innovations. Often the use of new ideas and technologies work negatively for the results of the competition in the short run. Putting too much emphasis on winning or lost potentially hampers the incentives for using new and innovative ideas. The challenge award is created to solve this problem. We consider this award to be equally prestigious to the World Championship.

This book is the first official archival publication of the long-range international research initiative. It is composed of three parts: overview papers, technical papers on focused topic, and team descriptions. Overview papers provides overall perspectives on RoboCup. It also contains RoboCup Challenge papers, which are a method of evaluating short-range technical challenges. Apart from competitions, such a method of quantitative evaluation is necessary to measure scientifically the progress in the field. Technical papers on focused topics are largely based on long papers presented at the RoboCup-97 Workshop. Papers focus of specific technical aspects involved in robot players. In addition, several papers discuss infrastructures for RoboCup such as a three-dimensional visualization system, a Java-based soccer simulator, and an automated commentary generation system. An educational perspective is also included. The team descriptions are papers describing the technical and strategic aspects of participating teams. Authors were requested to include an analysis of their team based on the results of RoboCup-97. Some teams, however, decided to write longer technical papers instead of team descriptions papers. Also, for editing reasons, some papers originally presented as technical papers were located in the team description section. This is particularly the case for the winners of each league (refer to the corresponding papers in the description of these teams). I hope this volume contributes to the progress of the field, and accomplishment of our dream some day. This volume is the first archival publication recording our voyage.

Hiroaki Kitano
Chair, The RoboCup Federation
Senior Researcher, Sony Computer Science Laboratory, Inc.

RoboCup-97 Organization

RoboCup-97 Organizing Committee Members :

(Chairs and other members in alphabetical order)

Hitoshi Matsubara (Chair, Electrotechnical Laboratory)

Itsuki Noda (Simulation league chair, Electrotechnical Laboratory)

Sho'ji Suzuki (Real Robot league chair, Osaka University)

Minoru Asada (Osaka University)

Hajime Asama (RIKEN: The Institute of Physical and Chemical Research)

Hidegori Ishihara (Nagoya University)

Hiroaki Kitano (Sony Computer Science Laboratory)

Kenji Kimura (Chubu Bureau of International Trade and Industry)

Akihiro Matsumoto (Toyo University)

Akihiko Morita (NIHON Keizai Shinbun, INC.)

Tatsuya Narisawa (NIHON Keizai Shinbun, INC.)

Susumu Shimada (Chukyo University)

Atsushi Shinjo (IAMAS: International Academy
of Media Arts and Sciences)

Mutsumi Sugisaki (Fujita Corporation)

Yoichi Yazawa (NIHON Keizai Shinbun, INC.)

Organizer :

- RoboCup Japanese National Committee
- Nihon Keizai Shinbun Inc.

RoboCup-97 Sponsors and Supporting Organizations

Special Sponsors

- Namco Limited
- Sony Corporation

Sponsors

- Nihon Sun Microsystems K.K.
- Itochu Techno-Science Corporation

In Cooperation with

- Electrotechnical Laboratory
- Softopia JAPAN
- IAMAS
- Japan Society of Artificial Intelligence
- Robotics Society of Japan
- IEEE Robotics and Automation Society
- Nihon Silicon Graphics-Cray K.K.
- Fujita Corporation
- Japan Airlines
- Net One Systems Co. Ltd.
- SOUM Cooperaton

Supported by

- Chubu Bureau of International Trade and Industry
- Aichi Prefectural Government
- City of Nagoya
- New Technology Foundation

Table of Contents

Overview Papers

RoboCup: A Challenge Problem for AI and Robotics	1
<i>Hiroaki Kitano, Minoru Asada , Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara</i>	
Overview of RoboCup-97	20
<i>Itsuki Noda, Shoji Suzuki, Hitoshi Matsubara, Minoru Asada, and Hiroaki Kitano</i>	
The RoboCup Physical Agent Challenge: Goals and Protocols for Phase I	42
<i>Minoru Asada, Peter Stone, Hiroaki Kitano, Alexis Drogoul, Dominique Duhaut, Manuela Veloso, Hajime Asama, and Sho'ji Suzuki</i>	
The RoboCup Synthetic Agent Challenge 97	62
<i>Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada</i>	

Technical Papers

Playing Soccer by Modifying and Combining Primitive Reactions	74
<i>Jukka Riekki and Juha Röning</i>	
Learning, Deciding, Predicting: The Soccer Playing Mind	88
<i>Andrew Price, Andrew Jennings, John Kneen, and Elizabeth Kendall</i>	
Using Decision Tree Confidence Factors for Multiagent Control	99
<i>Peter Stone and Manuela Veloso</i>	
A Role-Based Decision-Mechanism for Teams of Reactive and Coordinating Agents	112
<i>Silvia Coradeschi and Lars Karlsson</i>	
Using an Explicit Model of Teamwork in RoboCup-97	123
<i>Milind Tambe, Jafar Adibi, Yaser Al-Onaizan, Ali Erdem, Gal A. Kaminka, Stacy C. Marsella, Ion Muslea, and Marcelo Tallis</i>	
Decision Making by the Characteristics and the Interaction in Multi-agent Robotics Soccer	132
<i>Akihiro Matsumoto and Hiromasa Nagai</i>	

Real-Time Vision Processing for a Soccer Playing Mobile Robot	144
<i>Gordon Cheng and Alexander Zelinsky</i>	
A Method Applied for Soccer's Behaviors Using Proper Feedback and Feedforward Control.....	156
<i>Hironori Mizuno, Masakatsu Kourogi, Yukihide Kawamoto, and Yoichi Muraoka</i>	
A Legged Robot for RoboCup Based on "OPENR"	168
<i>Masahiro Fujita, Hiroaki Kitano, and Koji Kageyama</i>	

JavaSoccer	181
<i>Tucker Balch</i>	

RoboCup-3D : The Construction of Intelligent Navigation System	188
<i>Atsushi Shinjoh</i>	

Generating Multimedia Presentations for RoboCup Soccer Games.....	200
<i>Elisabeth André, Gerd Herzog, and Thomas Rist</i>	

Football in Recent Times: What We Can Learn from the Newspapers.....	216
<i>Ian Frank</i>	

The Value of Project-Based Education in Robotics	231
<i>Igor M. Verner</i>	

Team Descriptions

Small-Size Robot Teams

The CMUnited-97 Small Robot Team	242
<i>Manuela Veloso, Peter Stone, Kwon Han, and Sorin Achim</i>	

Development of Self-Learning Vision-Based Mobile Robots for Acquiring Soccer Robots Behaviors	257
<i>Takayuki Nakamura</i>	

MICROB: The French Experiment in RoboCup	277
<i>Sébastien Rocher, Vincent Idasiak, Sébastien Doncker, Alexis Drogoul, and Dominique Duhaut</i>	

Description of Rogi-Team	286
<i>A. Oller, R. Garcia, J.A. Ramon, A. Figueras, S. Esteva, J.Ll. de la Rosa, J. Humet, and E. Del Acebo</i>	

Middle-Size Robot Teams

Autonomous Soccer Robots	295
<i>Wei-Min Shen, Jafar Adibi, Rogelio Adobatti, Bonghan Cho, Ali Erdem, Hadi Moradi, Behnam Salemi, and Sheila Tejada</i>	

Vision-Based Robot Learning Towards RoboCup: Osaka University "Trackies"	305
<i>S. Suzuki, Y. Takahashi, E. Uchibe, M. Nakamura, C. Mishima, H. Ishizuka, T. Kato, and M. Asada</i>	
RoboCup97: An Omnidirectional Perspective	320
<i>Andrew Price, Andrew Jennings, and John Kneen</i>	
Omni-directional Autonomous Robots Cooperating for Team Play	333
<i>K. Yokota, K. Ozaki, A. Matsumoto, K. Kawabata, H. Kaetsu, and H. Asama</i>	
The Spirit of Bolivia: Complex Behavior Through Minimal Control	348
<i>Barry Brian Werger with team members Pablo Funes, Miguel Schneider-Fontán, Randy Sargent, Carl Witty, and Tim Witty</i>	
Simulator Teams	
AT Humboldt - Development, Practice and Theory	357
<i>Hans-Dieter Burkhard, Makus Hannebauer, and Jan Wendler</i>	
Refinement of Soccer Agents' Positions Using Reinforcement Learning	373
<i>Tomohito Andou</i>	
The CMUnited-97 Simulator Team	389
<i>Peter Stone and Manuela Veloso</i>	
Co-evolving Soccer Softbot Team Coordintion with Genetic Programming .	398
<i>Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hender</i>	
Learning Cooperative Behaviors in RoboCup Agents	412
<i>Masayuki Ohta</i>	
Individual Tactical Play and Action Decision Based on a Short-Term Goal – Team Descriptions of Team Miya and Team Niken	420
<i>Harukazu Igarashi, Shougo Kosue, Masatoshi Miyahara, Toshiro Umaba</i>	
The Reactive Motion Planning in the Passive Situation	428
<i>Susumu Takaki</i>	
A Reactive Architecture for RoboCup Competition	434
<i>E. Pagello, F. Montesello, A. D'Angelo, and C. Ferrari</i>	
Team: <i>Kasuga-bitos</i> with Modulation of Playing	443
<i>Tomoo Inden and Tomoichi Takahashi</i>	

Team Sicily	450
<i>John Fry, Lyen Huang and Stanley Peters</i>	
Team Description: Building Teams Using Roles, Responsibilities, and Strategies	458
<i>Simon Ch'ng and Lin Padgham</i>	
A Multi-Layered Behavior Based System for Controlling RoboCup Agents	467
<i>Paul Scerri</i>	
Using <i>ABC</i> ² in the RoboCup Domain	475
<i>Vicente Matellán, Daniel Borrajo, and Camino Fernndez</i>	
Intergrating Learning with Motor Schema-Based Control for a Robot Soccer Team	483
<i>Tucker Balch</i>	
Describing Soccer Game in EAMMO	492
<i>Nobuhiro Ito, Takahiro Hotta, and Naohiro Ishii</i>	
Team GAMMA: Agent Programming on Gaea	500
<i>Itsuki Noda</i>	
Using Reactive Deliberation for Real-Time Control of Soccer-Playing Robots	508
<i>Yu Zhang and Alan K. Mackworth</i>	
A Multi-layered Planning Architecture for Soccer Agent	513
<i>Ransui Iso and Hiroshige Inazumi</i>	
Author Index	519

RoboCup: A Challenge Problem for AI and Robotics

Hiroaki Kitano¹, Minoru Asada², Yasuo Kuniyoshi³,
Itsuki Noda³, Eiichi Osawa¹, Hitoshi Matsubara³

¹ Sony Computer Science Laboratory,
3-14-13 Higashi-Gotanda, Shinagawa, Tokyo 141 Japan

² Dept. of Mechanical Engineering, Osaka University,
Suita, Osaka 565 Japan

³ Electrotechnical laboratory,
1-1-4 Umezono, Tsukuba, 305 Japan

Abstract. RoboCup 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. The first **RoboCup** competition was held at IJCAI-97, Nagoya. In order for a robot team to actually perform a soccer game, various technologies must be incorporated including: design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup is a task for a team of multiple fast-moving robots under a dynamic environment. Although RoboCup's final target is a world cup with real robots, RoboCup offers a software platform for research on the software aspects of RoboCup. This paper describes technical challenges involved in RoboCup, rules, and simulation environment.

1 Introduction

RoboCup (The World Cup Robot Soccer) is an attempt to promote AI and robotics research by providing a common task for evaluation of various theories, algorithms, and agent architectures. In order for the robot (physical robot and software agent) to play a soccer game reasonably well, wide range of technologies need to be integrated and numbers of technical breakthrough must be accomplished. The range of technologies spans both AI and robotics research, such as design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning and planning, intelligent robotics, sensor-fusion, and so forth. The first RoboCup, RoboCup-97, was held during IJCAI-97 at Nagoya, Japan, as a part of IJCAI-97's special program. Series of competitions are planned afterwards, just like the Formula One Championship. RoboCup consists of three competition tracks:

Real Robot League: Using physical robots to play soccer games. For RoboCup-97, there are two categories: small-size and middle size. A team for each category of league consists of up to five robots. The size of a robot for small size shall be within 180 cm² floor area and the maximum length of

longer dimension shall be within 18cm. A middle size robot shall be within 2000 cm². New categories will be created with technical needs and progress. Legged robot league and humanoid robot league, as well as wheel-based robots with 11 robots a team, are planned for future competitions.

Software Robot League: Using software agents to play soccer games on an official soccer server over the network.

Expert Robot Competition: Competition of robots which has special skills, but not able to play a game.

Although RoboCup's primary objective is a world cup with real robots, RoboCup offers a software platform for research on the software aspects of RoboCup. Software robot league, also called the simulator league, enables wider range of researchers to take part in this program. It also promotes research on network-based multi-agent interactions, computer graphics, and physically realistic animations — a set of technologies potentially promotes advanced use of internet. In addition, we intend to create an award for an expert robot, which demonstrates a high-level of competence for a specific task, such as shooting, intercepting, etc.

While it is so obvious that building robot to play soccer game is an immense challenge, readers might wonder why we propose RoboCup. It is our intentions to use RoboCup as a vehicle to revitalize AI research, by offering publicly appealing, but formidable challenge. One of the effective ways to promote engineering research, part from specific application developments, is to set a significant long term goal. When the accomplishment of such a goal has significant social impact, it is called the grand challenge project [Kitano et. al, 93]. Building a robot to play soccer game itself do not generate significant social and economic impact, but the accomplishment will certainly considered as a major achievement of the field. We call this kind of project as a landmark project. RoboCup is a landmark project as well as a standard problem.

The successful landmark project claims to accomplish a very attractive and broadly appealing goals. The most successful example is the Apollo space program. In case of the Apollo project, the U.S. committed the goal of “landing a man on the moon and returning him safely to earth.” [Kennedy 61] The accomplishment of the goal itself marks the history of the mankind. Although the direct economic impact of having someone landed on the moon is slim⁴, technologies developed to achieve this goal was so significant that it formed the powerful technological and human foundations to the American industries. The important issue for the landmark project is to set the goal high enough so that a series of technical breakthrough is necessary to accomplish the task, and the goal need to be widely appealing and exciting. In addition, a set of technologies necessary to accomplish the goal must be the technologies which can form the foundation of the next generation industries.

⁴ To be fair, the Apollo mission was planned to gain the “National Prestige” and to demonstrate technical superiority over the former Soviet Union. Even in this, aspect, no direct military advantage was gained by having few astronauts on the moon.

In case of RoboCup, the ultimate goal is to “develop a robot soccer team which beats Brazil world cup team.” (a more modest goal is “to develop a robot soccer team which play like a human players.”) Needless to say, the accomplishment of the ultimate goal will take decades of efforts, if not centuries. It is not feasible, with the current technologies to accomplish this goal in any near term. However, this goal can easily create a series of well directed subgoals. Such an approach is common in any ambitious, or overly ambitious, project. In case of the american space program, the Mercury project and the Gemini project, which manned orbital mission, were two precursors to the Apollo mission. The first subgoal to be accomplished in RoboCup is “to build a real and software robot soccer teams which plays reasonably well with modified rules.” Even to accomplish this goal will undoubtfully generates technologies which impacts broad range of industries.

One other aspect of RoboCup is a view that RoboCup is a standard problem so that various theories, algorithms, and architectures can be evaluated. Computer chess is a typical example of the standard problem. Various search algorithms were evaluated and developed using this domain. With the recent accomplishment by the Deep Blue team, which beat Kasparov, a human grand master, using the official rule, computer chess challenge is close to the finale. One of the major reasons for the success of computer chess as a standard problem is that the evaluation of the progress was clearly defined. The progress of the research can be evaluated as a strength of the system, which was indicated as the rating. However, as computer chess is about to complete its original goal, we need a new challenge. The challenge need to foster a set of technologies for the next generation industries. We consider that RoboCup fulfill such a demand. Table 1 illustrate difference of domain characteristics between computer chess and RoboCup.

	Chess	RoboCup
Environment	Static	Dynamic
State Change	Turn taking	Real time
Info. accessibility	Complete	Incomplete
Sensor Readings	Symbolic	Non-symbolic
Control	Central	Distributed

context recognition, vision, strategic decision-making, motor control, intelligent robot control, and many more.

2 Research Issues of RoboCup

In this section, we discuss several research issues involved in the development of real robots and software agents for RoboCup. One of the major reasons, why RoboCup attract so many researchers is that it requires integration of broad range of technologies into a team of complete agents, as opposed to a task-specific functional module. Following is a partial list of research areas involved in RoboCup:

- Agent Architecture in general
- Combining reactive approach and modeling/planning approach
- Real-time recognition, planning, and reasoning
- Reasoning and action in dynamics environment
- Sensor fusion
- Multi-agent systems in general
- Behavior learning for complex tasks
- Strategy acquisition
- Cognitive modeling in general

In addition to these technologies, providing network-based soccer server with high quality 3D graphics capability requires advancement of technologies for the real time animation of simulated soccer players and network-based interactive multi-user server system. In addition, numbers of natural language researchers are using RoboCup for the target domain of their automatic commentary generation systems, as seen in DFKI's Roccoco system. These are key technologies for network-based services in coming years. In this paper, we will analyse briefly on some of these issues.

2.1 Agent Architecture

The existing robot players have been designed to perform almost single behavior such as pushing/dribbling/rolling [Connel and Mahadevan 93a; Asada et. al, 95; Sahota 94], juggling [Rizzi and Koditschek 93; Schaal and Atkeson 94], or hitting [Watanabe et al. 94]. A RoboCup player should be designed so that it can perform multiple subtasks such as shooting (including kicking), dribbling (pushing), passing, heading, and throwing a ball which often involve a common behavior, avoiding the opponents. Roughly speaking, there are two ways to build up a RoboCup player. Design each component which is specialized for a single behavior and assemble them into one. The other approach is to design one or two components that can perform multiple subtasks. The former seems easier to design but difficult to assemble and *vice versa*. In addition, the problem of how to combine reactive approach and deliberative approach will be a major research

issue. To quickly react against the ball and move around the field, use of subsumption architecture [Brooks 86], or other reactive approach may be effective. However, soccer players need to have global strategy as well as local tactics. These cannot be accomplished by mere reactive systems. On the other hand, deliberation-based approach, which involves planning and reasoning, may be too slow to react a quickly moving ball and to cope with a dynamically changing environment. The agent architecture for RoboCup players need to address the issue of how to combine these approaches.

2.2 Physical Components

Since the RoboCup player should move around quickly it should be compact, therefore the development of the integrated multi-functional module should be a new target of mechanical design for the RoboCup player. We need compact and powerful actuators with wide dynamic ranges. Also, we have to develop sophisticated control techniques to realize multiple behaviors by components as few as possible with low energy consumption.

The ultimate goal of a RoboCup player is like a humanoid type that can run and kick or pass a ball by its legs and feet, can throw a ball by its arms and hands, and can do a heading by its head. Since to build up a team of the humanoid types seems impossible within the current technology, this is just for a demonstration track for now. However, we expect that sometime in future participants of RoboCup overcome technical difficulties and participate with humanoid robots.

In addition, an attempt is being made to provide a standard physical components for robots. We are currently discussing about a possibility of making a standard for autonomous robot. The standard OpenR is not necessarily designed for RoboCup, but RoboCup is one of its significant application areas [Fujita and Kageyama 97].

2.3 Vision and sensor fusion

The visual information is the richest source of information to perceive not only the external world but the effects of the robot actions as well. The Computer Vision researchers have been seeking for the accurate 3-D geometry reconstructed from 2-D visual information believing in that the 3-D geometry is the most powerful and general representation to be used in many applications such as view generation for video database and robot manipulation and navigation. However, the time-consuming 3-D reconstruction might not be necessary nor optimally encoded for the task given to the RoboCup player. In order to react to the situation in real time, the RoboCup player needs the information which behavior to select against which situation. This does not mean to build up a special-purpose vision system but to claim that vision is a part of complex system that interacts in specific ways with world [Aloimonos 94]. The RoboCup is one of such worlds which make clear the role of vision and evaluate the performance of the image processing that have been left ambiguous in the computer vision field.

In addition to vision, the RoboCup player might need other sensing such as sonar, touch, and force/torque to discriminate the situations that cannot be discriminated from only the visual information nor covered by the visual information. Again, the RoboCup player needs the real time processing for multi-sensor fusion and integration. Therefore, the deliberative approaches to obtain the robust estimation by multi-sensor system does not seem suitable. We should develop a method of sensor fusion/integration for the RoboCup.

2.4 Learning behaviors

The individual players has to perform several behaviors one of which is selected depending on the current situation. Since programming the robot behaviors against the all situations considering the uncertainties in sensory data processing and action execution is infeasible, robot learning methods seem promising. As a method for robot learning, reinforcement learning has recently been receiving increased attention with little or no *a priori* knowledge and higher capability of reactive and adaptive behaviors [Connel and Mahadevan 93b]. However, almost of the all existing applications have been done only with computer simulations in toy world, and real robot applications are very few [Asada et. al, 95; Connell and Mahadevan 93a]. Since the prominence of the role of the reinforcement learning is largely determined by the extent to which it can be scaled to larger and complex robot learning tasks, the RoboCup seems a very good platform.

One example of research on this issue, among other research such as [Stone and Veloso 96], is a project at Asada Lab at Osaka University. Here, we will only show some photo's on his robots in action, so that interested readers can access his papers for detail.

Figure 1 shows how a real robot shoots a ball into a goal by using the state and action map obtained by the method [Asada et. al, 96]. 16 images are shown in raster order from the top left to the bottom right in every 1.5 seconds, in which the robot tried to shoot a ball, but failed, then moved backward so as to find a position to shoot a ball, finally succeeded in shooting. Figure 2 shows a sequence of images taken by the robot during the task execution shown in Figure 1. Note that the backward motion for retry is just the result of learning and not hand-coded. The method used here is an off-line learning one. Currently, they used an on-line learning method [Uchibe et. al, 96a].

At the primary stage of the RoboCup tournament, one to one competition seems feasible. Since the player has to take the opponent motions into consideration, the complexity of the problem is much higher than that of simple shooting without an opponent. To reduce the complexity, the task decomposition is often used. Asada et al. [Asada et al 94b] proposed a method of learning a shooting behavior avoiding a goal keeper. The shooting and avoiding behaviors are independently acquired and they are coordinated through the learning. Their method still suffers from the huge state space and the perceptual aliasing problem [Whitehead and Ballard 90] due to the limited visual field.

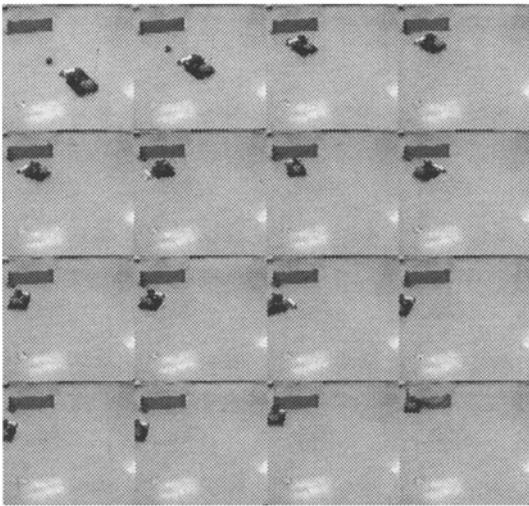


Fig. 1. A sequence of robot motion shooting a ball

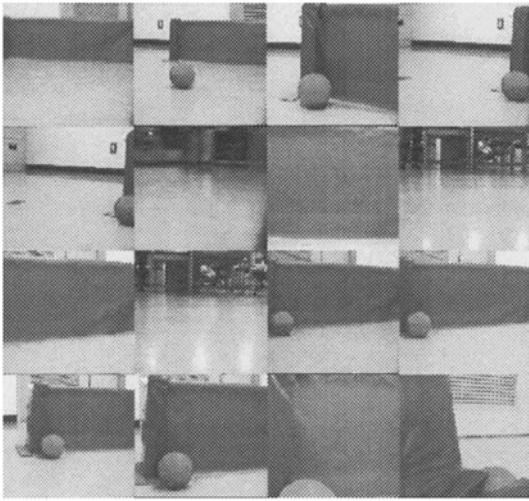


Fig. 2. A sequene of on-board camera images

Figure 3 shows a sequence of images in which the robot shoots a ball into a goal avoiding the opponent (a goal keeper) [Asada et al 94b].

2.5 Multi-Agent Collaboration

A soccer game is a specific but very attractive real-time multi-agent environment from the viewpoint of distributed artificial intelligence and multi-agent research. In a game, we have two competing teams. Each team has a team-wide common

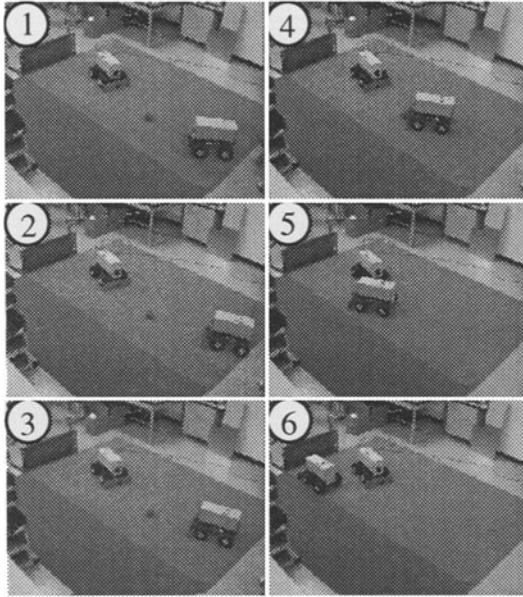


Fig. 3. A robot shooting a ball while avoiding an opponent robot

goal, namely to win the game. The goals of the two teams are incompatible. The opponent team can be seen as a dynamic and obstructive environment, which might disturb the achievement of the common team goal. To fulfill the common goal, each team needs to score, which can be seen as a subgoal. To achieve this subgoal, each team member is required to behave quickly, flexibly, and cooperatively; by taking local and global situations into account.

The team might have some sorts of global (team-wide) strategies to fulfill the common goal, and both local and global tactics to achieve subgoals. However, consider the following challenges:

1. the game environment, i.e. the movement of the team members and the opponent team, is highly dynamic.
2. the perception of each player could be locally limited.
3. the role of each player can be different.
4. communication among players is limited, therefore, each agent is required to behave very flexibly and autonomously in real-time under the resource bounded situation.

These restrictions are realistic and provides an interesting avenue of research for multi-agent systems. Let's us briefly look at multi-agent research which address cooperative planning under dynamics environment, where various resource and communication restrictions exists.

In cooperative distributed planning for common global goals, important tasks include the generation of promising local plans at each agent and coordination

of these local plans. When the dynamics of the problem space, e.g. the changing rate of goals compared with the performance of each planner, is relatively large, reactive planning that interleaves the plan generation and execution phases is known to be an effective methodology at least for a single agent [McDermott 78; Agre and Chapman 87; Maes 91; Ishida and Korf 91]. Whether this scheme extends naturally to the multi-agent environment is an interesting issue.

For cooperative plan schemes, there are frequent changes in the problem space or the observation of each agent is restricted locally. There is a trade-off between communication cost, which is necessary to coordinate the local plans of agents with a global plan, and the accuracy of the global plan (this is known as the predictability/responsiveness tradeoff). The study of the relationship between the communication cost and processing cost concerning the reliability of the hypotheses in FA/C [Lesser and Erman 80], and the relationship between the modification cost of local plans and the accuracy of a global plan in PGP [Durfee and Lesser 87] illustrate this fact. Also, Korf addressed it theoretically in [Korf 87]. Tambe specifically use RoboCup domain to test a scheme for joint intention generation[Tambe 96].

Schemes for reactive cooperative planning in dynamic problem spaces have been proposed and evaluated sometimes based on the pursuit game (predator-prey) [Benda *et al.* 85; Stephens and Merx 89; Gasser *et al.* 89; Levy and Rosen-schein 92; Korf 92; Osawa 95]. However, the pursuit game is a relatively simple game. Tileworld[Pollack and Ringuette 90] was also proposed and studied[Kinny and Georgeff 91; Ishida and Korf 91]. However, the environment is basically for the study of a single agent architecture.

As it is clear from these research, RoboCup directly address a critical issue in multi-agent systems research — generation and execution of cooperative plan under the dynamic environment. RoboCup provides an interesting and critically important task for multi-agent cooperative planning.

3 RoboCup Simulator

3.1 Soccer Server

In the simulation section, we will use Soccer Server, a simulator of **RoboCup** developed by Dr. Itsuki Noda, ETL, Japan, which is a network-based graphical simulation environment for multiple autonomous mobile robots in a 2D space. Using the soccer server, each client program can control each player on a soccer field via UDP/IP. This allows us to compare different types of multi-agent systems through the server, and test how well techniques of cooperation of agents work in dynamical varied situations.

The soccer server provides a virtual field where players of two teams play a soccer (association football) game. Each player is controlled by a client program via local area networks. Control protocols are simple in that it is easy to write client programs using any kind of programming system that supports UDP/IP sockets.

Control via Networks: A client can control a player via local area networks. The protocol of the communication between clients and the server is UDP/IP. When a client opens a UDP socket, the server assigns a player to a soccer field for the client. The client can control the player via the socket.

Physical Simulation: The soccer server has a physical simulator, which simulates movement of objects (ball and players) and collisions between them. The simulation is simplified so that it is easy to calculate the changes in real-time, but the essence of soccer is not lost.

The simulator works independently of communications with clients. Therefore, clients should assume that situations on the field change dynamically.

Referee: The server has a referee module, which controls each game according to a number of rules. In the current implementation, the rules are: (1) Check goals; (2) Check whether the ball is out of play; (3) Control positions of players for kick-offs, throw-ins and corner-kicks, so that players on the defending team keep a minimum distance from the ball.

Judgments by the referee are announced to all clients as an auditory message.

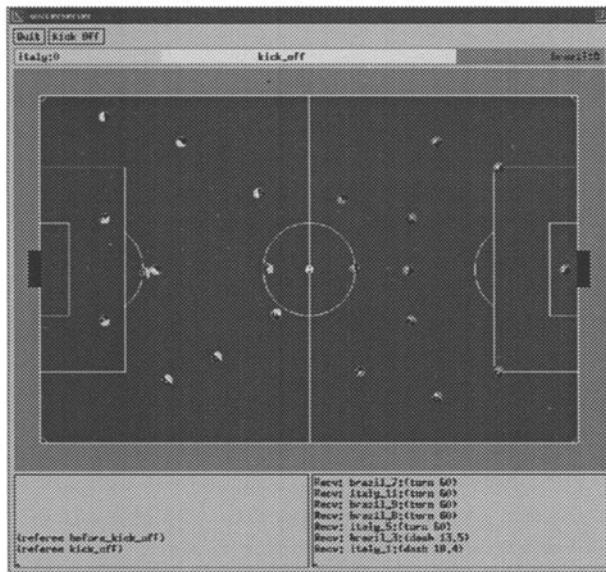


Fig. 4. Screen of Soccer Server

Although current version of the soccer server do not implements detailed physical and visual simulations, as seem in [Tu and Terzopoulos 94], we are planning to incorporate more realistic simulation in the future version.

3.2 Visualization

While the current version of the soccer server provides a top-down two-dimensional visualization, an independent server and browser systems provide a three dimensional visualization. A group of researchers at International Academy of Media Arts and Science (IAMAS) and Softopia, located in Gifu, Japan, is developing a three-dimensional visualization system. The first version of the system was used at RoboCup-97. A series of development project are planned for high-quality visualization, associated with automated camera-switching and commentary generation systems.

4 Conclusion

As it is clear by now, RoboCup provides fertile ground for AI and robotics research. The ultimate goal of RoboCup is so difficult that any near term accomplishment is not feasible. There is a clear paths to the stated goal, and each step toward the goal will generate a set of technologies which impacts industries for the next generation. Apart from these impact assessment, we believe that RoboCup contribute to AI and robotics community by providing exciting and publicly appealing goals, so that researchers in various field can collaborate for the common goal. We wish RoboCup offers an opportunities for AI and robotics community to revitalize their activities. "Let's get AI moving again!"

APPENDIX: Regulations for RoboCup

Regulations for RoboCup Real Robot Session (Summary)

General Policy

'Real worldness' in RoboCup mainly arises from the vast complexity of the overall situation due to interactions between behaviors and strategies of the ball and the players which cannot be fully predicted or controlled.

In the real robot session, we expect to have significantly greater complexity and hence much stronger reality than the simulation session. This is introduced by the uncertainty and uncontrollability in the structures and functions of the real robots along with real physical phenomena.

Therefore, we lean toward the least commitment policy in the game regulations, so that they do not obstruct surprises and creativity.

Due to the technical difficulty and unpredictability, the regulations can be adjusted to the overall situation of participating teams in each contest. However, the modifications must maintain the fairness to all the participants and must be announced in advance of the contest with an approval by the RoboCup technical committee.

The following sections summarize the regulations as of July 1997 very briefly. The most recent version can be obtained from the RoboCup web site. As rules

being modified to cope with various technical and management issues, please obtain the up-to-date rules from the web site. The rule have undergone two major changes since the first announcement in 1995. Also, prior to the announcement, several changes have been made since 1993, when we first drafted the RoboCup rule. The recent major changes include the size of the field, the size of robot, and the creation of defense zone. The field was defined based on a ping pong table so that most people can purchase it at low cost anywhere in the world. It is important to consider logistics of the material supply. The field, balls, and other materials should be so chosen that widest possible researchers can easily access and purchase with low cost. After a hearing period and discussions, the international committee for RoboCup have finalized the regulations for RoboCup-97. Further modifications will be made reflecting progress of research of participants. The RoboCup real robot league basically have two different classes based on the size of robots and the field — small size robot and medium size robot. Other classes, such as legged robots and humanoid robots may be created after the discussion by the committee.

The regulation for small robot league (excerpt)

Field Size: A ping pong table (a table tennis table) is used for the official match. The size and color of the table is officially determined as the international standard for ping pong. It is 152.5cm by 274cm, and color is green. Details shall be given in the figure 5.

Four small panels are attached to the corner to avoid ball to stuck. As shown in the figure below, it should be located 3 cm from the corner for each axis. Green strips of width 1cm shall be painted to identify the edge of the panel.

Robot: The maximum diameter of a circular robot shall be 15cm, while the maximum length of a rectangular robot shall be 18cm with a width of 10cm. These provide for the same size of robot in terms of surface area. This is approximately 1/10 of the length of the shorter end of the field.

(At the end of 1997, this rule was rewritten as:

For the small-size, each robot shall be within a floor area of 180 square centimeter, and the maximum length of the robot body shall be within 18cm. This is approximately equivalent to the previous definition that maximum diameter of a circular robot shall be 15cm. These provide for the same size of robot in terms of surface area. Height of the robot shall be within 15cm when the team uses the global vision system. The allowance was made for those who only uses the on-board vision systems (cameras are on-board, but not necessary all processors.) that up to 22.5cm height is permitted for 1998. This will be reviewed during RoboCup-98 Paris and the decision will be made whether this allowance shall be abolished in future, and if so when it shall be abolished.)

Team: A team should consist of no more than 5 robots.

Goals: The width of the goal is 50 cm, which is approximately 1/3 of the length of the shorter end of the field.

Ball: Orange golf ball shall be used.

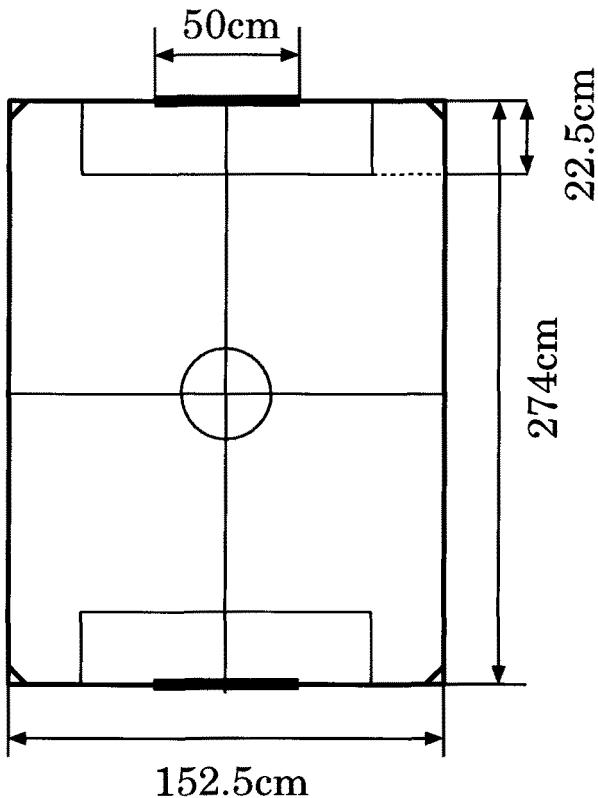


Fig. 5. Top view of the field for small robots

Colorings: Colors of each part of the field are as follows:

- Field shall be green.
- Wall shall be white.
- Ball shall be orange.
- Lines are drawn in white.
- Some markers on corners and goals are in green.

Length of the game: The games consists of the first half, break, and the second half. Each of them is 10 minutes.

Wall: A wall which is the same height as the golf ball shall be placed all around the field, except in goals. The wall shall be painted in white.

Defense Zone: Defense zone will be created in surrounding the goal of each side. It is 22.5 cm from the goal line, and width of 100 cm. The boarder of the defense zone will be painted in white, with the width of 1cm. Only one defense robot can enter this area. A brief passing and accidental entry of other robots are permitted, but intensional entry and stay is prohibited.

Global Vision System / External Distributed Vision System: The use

of a global vision system and an external distributed vision system is permitted, not required, to identify and track the position of robots and balls. The rule explicitly allow the use of multiple cameras for tracking. The use of the global vision system shall be notified at the time of register, and detailed arrangements shall be discussed with the RoboCup organizing committee. (This rule was added at the end of December 1997. The rule make it explicit that for small-size, multiple camera and distributed vision is permitted to foster distributed vision research.)

Robot marking: Each robot should put at least one colored ping pong ball on top of their body, approximately between 15 cm to 20 cm in height. Teams may, at their discretion, use two pingpong balls (of differing colours) per robot to determine the orientation of the robot as well as its postion. The color(s) of the ping pong ball(s) will be used to identify friend and enemy, as well as positions using the global vision system.

Goal keepers: Goal keeper can hold and manipulate a ball for up to 10 seconds within its penalty area. After releasing the ball, the keeper must not hold the ball until it touches any opponent, or an alley outside the penalty area. If the ball released by the keeper reaches the other half end of the court without touching any other player, the opponent is given an indirect free kick positioned anywhere along the half way line (borrowed from Futsal rule).

Fouls: Following fouls are defined:

Multiple Defense: When more than one defense robots enter the defense zone to substantially affects the game. The foul will be called, and the penalty kick will be declared.

Ball Holding: A player cannot 'hold' a ball unless it is a goal keeper in its penalty area. Holding a ball means taking a full control of the ball by removing its entire degrees of freedom; typically, fixing a ball to the body or surrounding a ball using the body to prevent accesses by others. A free kick will be decleared. If this happens in the defense zone by the denfense team, a penalty kick will be declared.

Court Modification: Modification or damage to the court and the ball is forbidden. Should this occur, the game is suspended and the appropriate restoration is done immediately before the game resumes.

Robot Halting: All the players must be halted prior to kick-off or restarting of the game. The judges check or adjust the placements of the players and declares the completion of adjustment by 5 seconds before cueing a kick-off or a restart action. During this 5 seconds, the players can move.

Offside Offside rule is not adopted.

Charging: Unless during striving for a ball, a player must not attack another. In case the umpire clearly observes such an act, it is regarded as a violent action. Then the umpire presents a red card to the responsible player ordering it to leave the game. The judgment is done based on an external appearance.

Throughout the game, if a player utilizes a device or an action which continuously exerts, or whose primal purpose appears to be, serious damages to other robot's functions, the umpire can present a yellow card as a warning to the responsible player, and order it to go outside the court and correct the problem. Once the correction is made, the robot can resume to the game under an approval by the umpire. In case the problem is repeated, the umpire presents a red card to the responsible player telling it to leave the game.

Aside from the above items, no regulations are placed against possible body contacts, charging, dangerous plays, obstructions etc.

The Regulations for Medium Size Robots (excerpt)

The regulations for medium size robots basically applies the same rule as the rule for small robot. All sizes are multiplied by 3. This means that:

Field: 457.5cm by 822cm

Goals: The width of the goal is 1500mm and the height is 500mm. Each goal is coloured in one colour (defined in Colouring) and has different colour from the other.

Corner: Four small panels are attached to the corner to avoid sticking the ball.

As shown in the figure below, it should be located 200mm from the corner for each axis. Green strips of width 30mm shall be painted to identify the edge of the panel.

Penalty Area: Penalty area will be created in surrounding the goal of each side. It is 675mm from the goal line, and width of 3000mm. The area is shown by white lines with the width of 35mm.

Robot: The size of the robot should be within a circle of 500mm diameter or 450mm square.

Team: A team should consist of no more than 5 robots.

Ball: FIFA Size 4 Futsal ball painted red.

Colourings: Following parts are painted in different colour:

- A field is green
- Walls are white
- Lines drawn on the field is white
- Markers on corners is green (see the photo above)
- A ball is red
- Goals are blue and yellow

Regulations of Simulation Track (excerpt)

1. Field of Play

The field of play is provided by Soccer Server, a simulator of a soccer field. A match is carried out in a server-client style: The server, Soccer Server, provides a virtual field and simulates all movements of a ball and players.

Clients become brains of players and control their movements. Communication between a server and each client is done using UDP/IP sockets via local area networks.

2. Players and Teams

The simulation track of PreRoboCup consists of ‘small track’ and ‘standard track’. In the small track, each team has 1 ~ 5 players. In the standard track, each team has 6 ~ 11 players. There is no goalkeeper because players have no hands. Even a team consists of fewer players than another team, a match is carried out without any penalties.

Client programs can be written by any programming systems. with the following restrictions.

- (a) A client controls only a player. Or if a client controls multiple players, the different control modules of players are separated logically from each other.
- (b) Clients may not communicate directly with each other. Communication between clients must be done by facilities provided by Soccer Server.

3. Rules

The referee module in Soccer Server controls a match according to 3 rules: goal, out-of-field, clearance. Moreover, a human referee also controls a match. When he/she judges player’ action is too un-gentle, for example, surrounding the ball, he/she suspends the match and restarts by a free kick of the opposite team.

4. Format of the Competition

The competition shall be played in two rounds. In the first round, teams shall be divided into several groups of 4 teams. The system of play shall be the league system, each team playing one match against each of the other teams in the same group. The two teams coming first and second in each group shall qualify for the second round.

The second round shall be played by a system of elimination (cup system).

For more detail, please refer the following WWW homepage and FTP cite.

```
http://ci.etc.go.jp/~noda/soccer/regulations/regulations.html
http://ci.etc.go.jp/~noda/soccer/manual.newest/main.html
ftp://ci.etc.go.jp/pub/soccer/server/
```

References

- [Agre and Chapman 87] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp. 268–272, 1987.
- [Aloimonos 94] Y. Aloimonos. “Rply: What i have learned”. *CVGIP: Image Understanding*, 60:1:74–85, 1994.
- [Asada et. al, 96] M. Asada, S. Noda, and K. Hosoda. Action-based sensor space categorization for robot learning. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, pages ???–???, 1996.

- [Asada et. al, 95] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based reinforcement learning for purposive behavior acquisition. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 146–153, 1995.
- [Asada et al 94a] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. “Purposive behavior acquisition on a real robot by vision-based reinforcement learning”. In *Proc. of MLC-COLT (Machine Learning Conference and Computer Learning Theory) Workshop on Robot Learning*, pages 1–9, 1994.
- [Asada et al 94b] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. “coordination of multiple behaviors acquired by vision-based reinforcement learning ”. In *Proc. of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems 1994 (IROS '94)*, pages 917–924, 1994.
- [Benda et al. 85] M. Benda, V. Jagannathan, and R. Dodhiawalla. On Optimal Cooperation of Knowledge Sources. Technical Report BCS-G2010-28, Boeing AI Center, 1985.
- [Brooks 86] R. Brooks, “A robust layered control system for a mobile robot,” *IEEE J. Robotics and Automation*, RA-2, 14-23, April, 1986.
- [Connel and Mahadevan 93a] J. H. Connel and S. Mahadevan. “Rapid task learning for real robot”. In J. H. Connel and S. Mahadevan, editors, *Robot Learning*, chapter 5. Kluwer Academic Publishers, 1993.
- [Connel and Mahadevan 93b] J. H. Connel and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic Publishers, 1993.
- [Tu and Terzopoulos 94] X. Tu, and D. Terzopoulos, “Artificial Fishes: Physics, Locomotion, Perception, Behavior,” *Proc. of SIG GRAPH-94*, Orlando, Florida, July 24-29, pp43-49, 1994.
- [Durfee and Lesser 87] E. Durfee and V. Lesser. Using Partial Global Plans to Coordinate Distributed Problem Solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, 1987.
- [Fujita and Kageyama 97] M. Fujita and K. Kageyama, An Open Architecture for Entertainment Robot. In *Proceedings of the First International Conference on Autonomous Agents*, 1997.
- [Gasser et al. 89] L. Gasser, N. Rouquette, R. Hill, and J. Lieb. Representing and Using Organizational Knowledge in Distributed AI Systems. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pp. 55–78. Morgan Kaufmann Publishers, Inc., 1989.
- [Ishida and Korf 91] T. Ishida and R. Korf. Moving Target Search. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 204–210, 1991.
- [Kennedy 61] Kennedy, J. F., “Urgent National Needs,” *Speech to a Joint Session of Congress*, 25 May 1961, Congressional Record – House (25 May 1961) p.8276.
- [Kinny and Georgeff 91] D. Kinny and M. Georgeff. Commitment and Effectiveness of Situated Agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 82–88, 1991.
- [Kitano et. al, 93] H. Kitano, B. Wah, L. Hunter, R. Oka, T. Yokoi, and W. Hahn, “Grand Challenge AI Applications,” In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, 1993.
- [Korf 87] R. Korf. Planning as Search: A Quantitative Approach. *Artificial Intelligence*, Vol. 33, No. 1, pp.65–88, 1987.
- [Korf 92] R. Korf. A Simple Solution to Pursuit Games. In *Proceedings of the Eleventh International Workshop on Distributed Artificial Intelligence*, 1992.

- [Lesser and Erman 80] V. Lesser and L. Erman. Distributed Interpretation: A Model and Experiment. *IEEE Transactions on Computers*, Vol. 29, No. 12, pp.1144–1163, 1980.
- [Levy and Rosenschein 92] R. Levy and J. Rosenschein. A Game Theoretic Approach to Distributed Artificial Intelligence. In Eric Werner and Yves Demazeau, editors, *Decentralized A.I. 3*. Elsevier/North Holland, 1992.
- [Maes 91] P. Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pp. 49–70. The MIT Press, 1991.
- [Mataric 94] M. J. Mataric. Learning to behave socially. In *Proc. of the 3rd Int. Conf. on Simulation and Adaptive Behaviors – From animals to animats 3* –, pages 453–462, 1994.
- [McDermott 78] D. McDermott. Planning and Action. *Cognitive Science*, Vol. 2, pp.71–110, 1978.
- [Nourbakhsh et al 93] I. Nourbakhsh, et al., The Winning Robots from the 1993 Robot Competition. *The AI Magazine*, Vol. 14, No. 4, 51-62, The AAAI Press, 1993.
- [Osawa 95] E. Osawa. A Metalevel Coordination Strategy for Reactive Cooperative Planning. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.
- [Pollack and Ringuette 90] M. Pollack and M. Ringuette. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 183–189, 1990.
- [Rizzi and Koditschek 93] A. A. Rizzi and D. E. Koditschek. Further progress in robot juggling: The spatial two-juggle. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 919–924, 1993.
- [Sahota 94] M. K. Sahota. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In *Proc. of AAAI-94*, pages 1303–1308, 1994.
- [Schaal and Atkeson 94] S. Schaal and C. G. Atkeson. Robot learning by nonparametric regression. In *Proc. of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems 1994 (IROS '94)*, pages 478–485, 1994.
- [Stephens and Merx 89] L. Stephens and M. Merx. Agent Organization as an Effector of DAI System Performance. In *Proceedings of the Ninth Workshop on Distributed Artificial Intelligence*, pp. 263–292, 1989.
- [Stone and Veloso 96] P. Stone, and M. Veloso, “Beating a defender in robotic soccer: Memory-based learning of a continuous function,” In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, Cambridge, MA, MIT Press, 1996.
- [Tambe 96] M. Tambe, “Tracking Dynamic Team Activity ” In *Proceedings of AAAI-96*, Portland, 1996.
- [Uchibe et. al, 96a] E. Uchibe, M. Asada, and K. Hosoda. 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 (IROS96)*, pages ???–???, 1996.
- [Uchibe et. al, 96b] E. Uchibe, M. Asada, and K. Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, pages ???–???, 1996.
- [Watanabe et al. 94] H. Watanabe, Y. Nihna, y. Masutani, and F. Miyazaki. Vision-based motion control for a hitting task - hanetsuki -. In *Proc. of IEEE/RSJ/GI*

International Conference on Intelligent Robots and Systems 1994 (IROS '94), pages 910–916, 1994.

[Whitehead and Ballard 90] S. D. Whitehead and D. H. Ballard. “Active perception and reinforcement learning”. In *Proc. of Workshop on Machine Learning-1990*, pages 179–188, 1990.

Overview of RoboCup-97

Itsuki Noda¹ noda@etl.go.jp

Shoji Suzuki² ssuzuki@ams.eng.osaka-u.ac.jp

Hitoshi Matsubara¹ matsubar@etl.go.jp

Minoru Asada² asada@ams.eng.osaka-u.ac.jp

and Hiroaki Kitano³ kitano@cs1.sony.co.jp

¹ Electrotechnical Laboratory, Umezono 1-1-4, Tsukuba, Ibaraki 305, JAPAN

² Osaka University, Yamadaoka 2-1, Suita, Osaka 565, JAPAN

³ Sony, Computer Science Laboratory,
Higashi-Gotanda 3-14-13, Shinagawa, Tokyo 141 JAPAN

Abstract. RoboCup-97, The First Robot World Cup Soccer Games and Conferences, was held at the Fifteenth International Joint Conference on Artificial Intelligence. There were two leagues: real robot and simulation. 10 teams participated in the real robot league and 29 teams did in the simulation league. The World Champions are CMUnited (CMU, U.S.A.) for the Small-Size league, Dreamteam (Univ. of Southern California, U.S.A.) and Trackies (Osaka Univ., Japan) for the Middle-Size league, and At-Humboldt (Humboldt Univ., Germany) for the Simulation league. The Scientific Challenge Award was given to Sean Luke (Univ. of Maryland, U.S.A.) for his genetic programming based simulation team and the Engineering Challenge Awards was given to UttoriUnited (Utsunomiya Univ., Toyo Univ. and RIKEN, JAPAN) and RMIT (Royal Melbourne Institute of Technology, Australia) for designing novel omni-directional driving mechanisms. RoboCup-98, the Second Robot World Cup Soccer Games and Conferences, will be held at the Third International Conference on Multi-Agent Systems at Paris, France in July 1998.

1 Introduction

RoboCup is an attempt to promote AI and robotics research by providing a common task, Soccer, for evaluation of various theories, algorithms, and agent architectures[Kitano *et al.* 1997a]. RoboCup-97, The First Robot World Cup Soccer Games and Conferences, was held in August 22-28 1997 at Nagoya Congress Center, Japan. RoboCup-97 was the official event of IJCAI-97, The Fifteenth International Joint Conference on Artificial Intelligence.

RoboCup-97 consists of competitions, exhibitions, and a technical workshop. Workshop was attended by over 150 researchers, and organized as a part of IJCAI-97 workshop program. Competitions were organized into the simulator league, the small-size robot league, and the middle-size robot league. Overall, 38 teams participated in RoboCup-97 from 10 countries. There were 29 teams for the simulation league, 4 teams for the small-size robot league, and 5 teams

for middle-size robot league. During RoboCup-97, Over five thousand people watched the games and over one hundred international media reported the games and workshop.

This article reports RoboCup-97. RoboCup-97 had two leagues: real robot league and simulation league. Real robot league had tow leagues: small-size and middle-size. Aside from the world championship awards, RoboCup created the RoboCup Scientific Challenge award and Engineering Challenge Award, to be equally prestigious award as the world champion. The detailed information about RoboCup is available from:

<http://www.robocup.org/RoboCup>

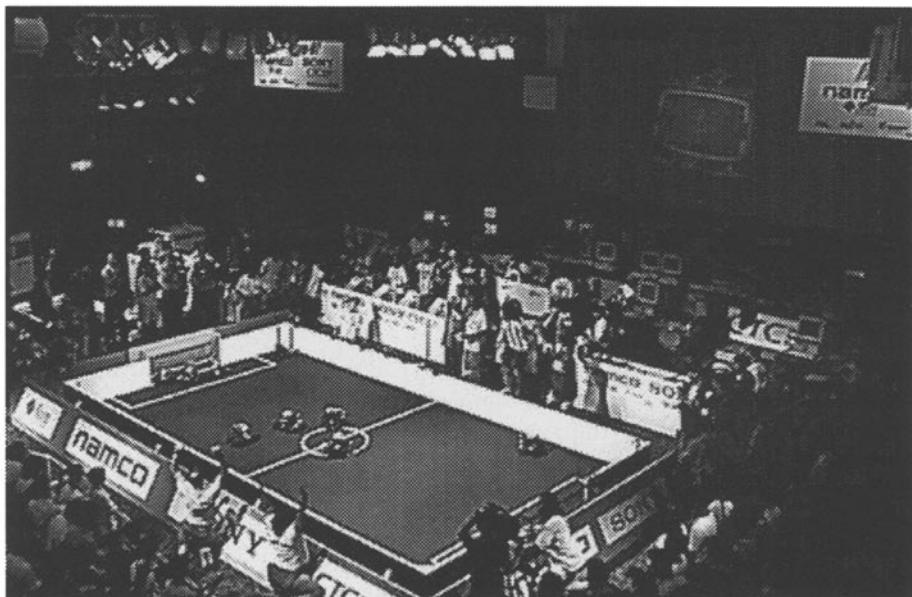


Fig. 1. RoboCup-97 (Copyright, 1997 P. Plailly/Eurclios. All right reserved.)

2 Real Robot League

The Real Robot League which uses physical robots to play soccer games consists of several categories. In RoboCup-97, there were two categories for game competition and one for skill competition.

The Small-Size League: A team consists of five robots and plays on a field equivalent to one ping-pong table. Each robot is about 15cm in diameter, or under 180cm² and the maximum length must be less than 18cm. An orange golf ball is used.

The Middle-Size League: Here, there are five robots per team, and each robot must be less than 50cm in diameter, or 2,000cm². A FIFA size 4 Futsal ball is used. The field of play is equivalent to 9 ping-pong tables (3 by 3).

The Expert Robot League: Competition between robots having special skills. These skills will concentrate on isolated aspects of the game of soccer.

For the real robot league in the first RoboCup-97, about 10 teams throughout the world participated in the competition: four teams (Carnegie Mellon University (USA), Paris-VI (France), University of Girona (Spain), and Nara Advanced Institute of Science and Technology (Japan)) in the small-size robot league, five teams (ISI/USC (USA), Osaka University (Japan), Ullanta Performance Robotics (USA), RMIT (Australia), Uttoni United - A joint team of Riken, Toyo Univ., and Utsunomiya Univ. (Japan)) participated in the middle size league, and two teams (RMIT (Australia) and Colorado School of Mines (USA)) in the expert robot league.

2.1 The Small-Size League

For the small robot league, the use of global vision system is permitted, which enables the team to plot absolute position information for each robot and the ball.

A ping pong table was selected as the basic field size because it is a low cost standardized material which can be purchased throughout the world. We initially defined a field as 1/100 of the FIFA world cup field which is 120m by 90m. However, in this case, researchers would have had to build everything from scratch. Considering the amount of work that has to be done in building robots, field construction really presents no major efforts, nevertheless, it seemed important that necessary materials be a widely available. This is particularly important as the RoboCup becomes widely used for educational purposes. Figure 2 shows a scene of a match of the small robot league.

Global vision is permitted in this league for two reasons. First, it is rather difficult for this size of robot to have an on-board camera system. If they did, it would inevitably be very expensive so that many under-funded research groups would not be able to participate in the initiative. Second, the use of global vision allows us to investigate issues of distributed vision systems expected to cooperate with each other for video surveillance, monitoring, and guidance.

Actually, three of four teams adopted the global vision system, and only one team, NAIST (Japan) adopted an on-board vision system for each of their two robots. Figure 3 shows the robot of each team. Six games were scheduled for the preliminary rounds so that every team may have games with all teams. However, due to conflict in radio frequency for robot control, there was no game between MICROB and Girona. Table 1 shows the result of the preliminary games for the small-size league.

1. MICROB Universite Paris VI, France (Dominique Duhaut, et al.)

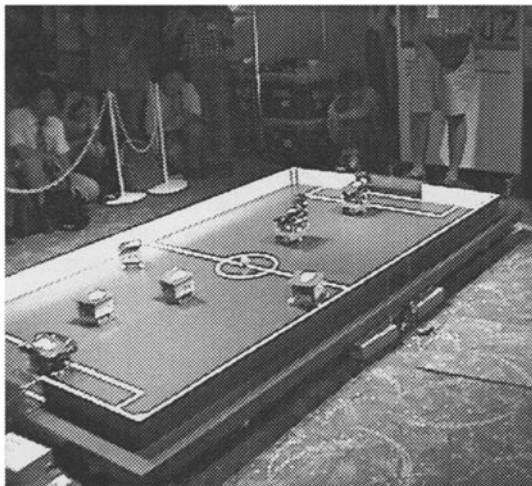


Fig. 2. A match of the small robot league

2. University of Girona, Spain (Josep Li De la Rosa, et al.)
3. CMUnited Carnegie Mellon University, USA (Manuela Veloso, et al.)
4. Nara Institute of Science and Technology, Japan (Takayuki Nakamura, et al.)

vs	Team 1	Team 2	Team 3	Team 4	Points	Goal Diff	Goals Scored	Rank
Team 1:MICROB	-	0-0	1-3	0-0	2	-2	1	3
Team 2:Girona U.	0-0	-	0-2	0-1	1	-3	0	4
Team 3:CMUnited	3-1	2-0	-	5-0	6	9	10	1
Team 4:NAIST	0-0	1-0	0-5	-	3	-4	1	2

Table 1. Result of the small-size league Preliminary

According to the rank of the preliminary, CMUnited and NAIST were selected to the final, and the Small-size league World Champion was awarded to CMUnited from Carnegie Mellon University which won over Nara Advanced Institute of Science and Technology, Japan by a 3-0 score in the final.

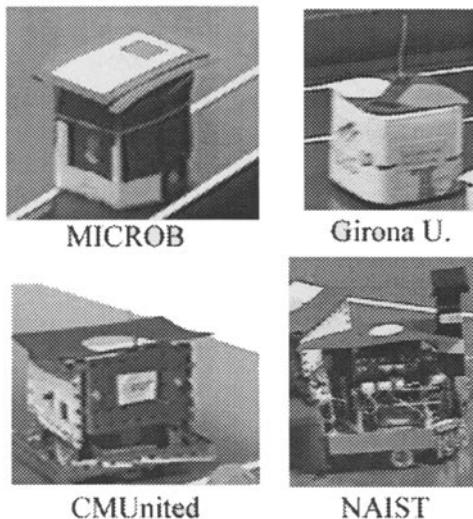


Fig. 3. Participated robots

2.2 The Middle-Size League

In this league, a FIFA, size 4, futsal ball is used. The size of the field is 8.2m × 4.5m which is equivalent to nine ping pong tables. Figure 4 shows a scene of a match. This year, we had five teams in the middle-size league each of which has its own features in several aspects:

1. Trackies, Osaka Univ. Japan (Asada Lab.) : remote brain systems, non-holonomic vehicles, four attackers and one goalee with omnidirectional vision system. Learning techniques to obtain basic skills such as shooting and avoiding.
2. RMIT Raiders: Royal Melbourne Institute of Technology (RMIT), The Department of Computer Systems Engineering, Research Center, Australia (Andrew Price, et al.): special mechanical design for omnidirectional motion of the round-shape robots, global vision system was used to control their four robots.
3. Ullanta Performance Robotics, U.S.A. (Barry Werger, et al.): three robot platforms provided by Real World Interface Corp., which has vision, sonar, and bumper sensors.
4. Utton United: Utsunomiya Uni., Toyo Univ., and The Institute of Physical and Chemical Research (Riken), Japan (Kazutaka Yokota, Hajime Asama, et al.): special mechanical design of omnidirectional motions different from RMIT team. Explicit communication through infra-red, completely on-board system

5. Dreamteam University of Southern California, Information Science Institute, USA (Wei-Min Shen, et al.) : completely on-board system. The same body as Trackies are used.

Figure 5 shows the robot of each team.

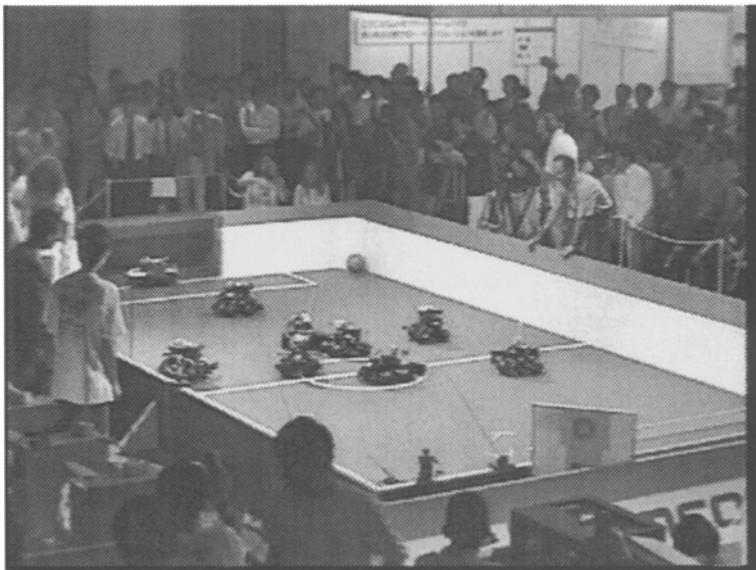


Fig. 4. A match of the middle robot league

Figure 6 shows visual images from four out of five robots being used by Osaka University Trackies. It should be noted that one of the robots uses an omni-vision camera system which can view 360 degrees. Typically, a goalee robot will be equipped with such a camera system, as it must respond quickly to opponent's shots from any direction. This set up focuses on a situation where all robots have their own sensing systems and are not backed up by any global sensing systems.

Totally, 5 games were held in the preliminary rounds (see Table 2), and the Middle-size league final resulted in a draw (0-0) between the Dreamteam of ISI/USC and the Trackies of Osaka University. Their scores in the preliminary round was also a 2-2 draw. The committee decided, in this case, to award the World Champion to both teams.

The Engineering Challenge Award was given to both UttoriUnited and RMIT for designing novel omni-directional driving mechanisms. These teams designed new robot driving mechanisms that used special wheels (Uttori) and balls (RMIT) to enable their respective robots to move to any direction without rotation (see Figure 7). Such mechanisms significantly improve a robot's maneuverability, and their potential impact is far reaching.

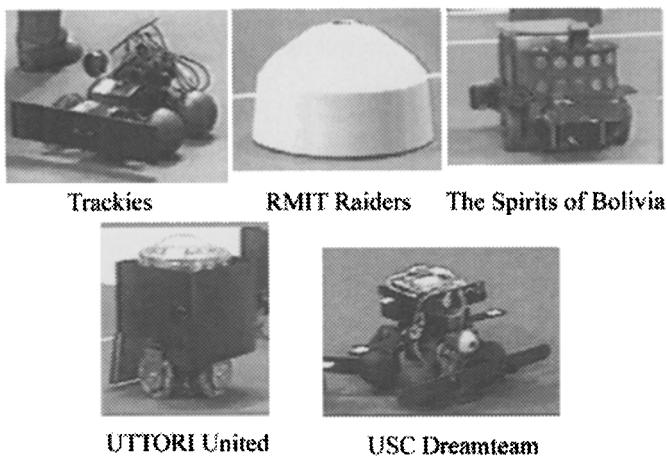


Fig. 5. Participated robots

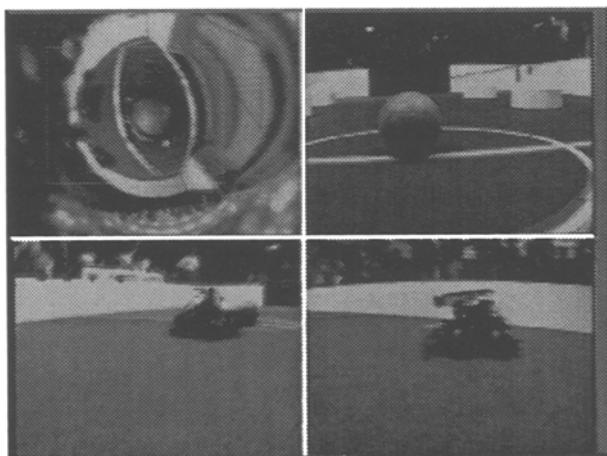
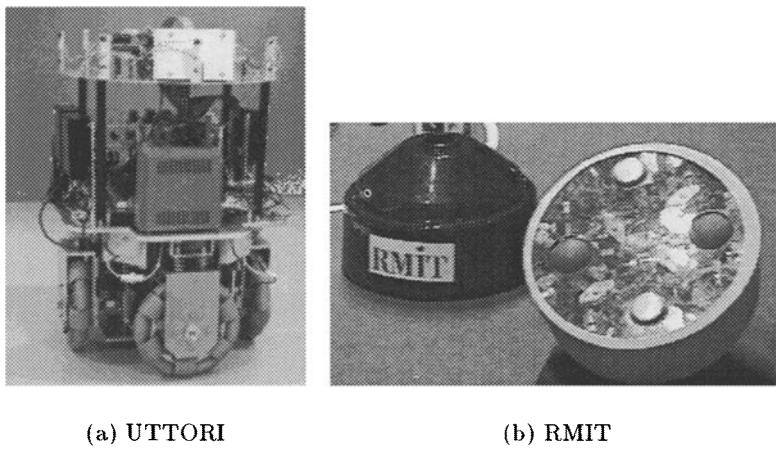


Fig. 6. Visual Image from four robots in the field

2.3 The Expert Robot League

Two teams made their entries for the expert robot league: Wolf, Colorado School of Mine (USA) and Raiders, RMIT (Australia). The Wolf used a robot platform by Nomadic Corp. with stereo vision system, and tried to chase a red ball (see Figure 8). However, the vision system did not work well, and therefore, the

vs	Team 1	Team 2	Team 3	Team 4	Team 5	Points	Goal Diff	Goals Scored	Rank
Team 1:Trackies	-	1-0	*	*	2-2	3	1	3	2
Team 2:Raiders	0-1	-	0-0	*	*	1	-1	0	4
Team 3:Ullanta	*	0-0	-	0-0	*	2	0	0	3
Team 4:Uttori	*	*	0-0	-	0-4	1	-4	0	5
Team 5:Dreamteam	2-2	*	*	4-0	-	3	4	6	1

Table 2. Result of the middle-size league Preliminary**Fig. 7.** The Omni-directional driving mechanism

Wolf did not make it. The Raiders demonstrated their omnidirectional motion mechanism that is specially designed for RoboCup-97 (see Figure 7(b)).

The committee decided that no team was awarded in the expert league since the both team did not make their achievements expected.

2.4 Issues in Future

Since RoboCup-97 was the first event of RoboCup initiative, we faced many unexpected problems. One of the most serious problem was radio link between robot bodies and their brains (off-board processors). All four teams in the small-size league, and two more teams in the middle-size league adopted the remote brain system, and they were suffering from serious radio noise in the competition site. Specially, Osaka Trackies suffered from noises in both ways, that is, transmitting video image to the host computer and sending motion command through radio



Fig. 8. The Robot of Colorado School of Mine

control unit. Therefore, the possibility of normal communication for five robots was very low. The set up to prevent such a problem should be designed in the future RoboCups. Also, each team shall design robust wireless communication systems, or to make it fully autonomous without relying on external processing in any essential aspects of control.

Related to the first issue, the second one was rules during a game. Many robots lost the red ball very frequently due to radio noise and/or some problems in image processing, and often the game was stacked. Therefore, the judge changed the ball position according a rule fixed in the competition site by all team representatives, and restarted the game. However, the positions of the ball were sometimes very easy for some robots to make a goal. Some systematic ways should be developed to avoid such trivial situations since the rule had been fixed after short discussions.

3 Simulation League

3.1 Overview

RoboCup simulation league is a common testbed of various technologies in the field of Artificial Intelligence [Noda and Matsubara 1996, Kitano *et al.* 1997b]. There are following major issue:

- Real World Problem: real-time processing, robustness, reactive system, learning
- Multi-agent System: agent programming, teamwork of agents, communication, resource sharing, agent modeling

Many researchers working on these fields participated in the simulation league.

First, we will describe rule and the simulator used in the simulation league in section 3.2, and show results of the competition and overview of team-features in section 3.3 and 3.4.

3.2 Rule and Simulator

Rule In the simulation league, each team must build 1 ~ 11 player programs. Of course they can use the same program for all players. Each player program connects with **Soccer Server** as a client, which simulates movements of objects (players and ball) on the soccer field. The player program control only a player. It receives visual and verbal sensor information from the server and sends control commands ('turn', 'dash', 'kick' and 'say') to the server. Ability of a player to sense and to act is decided by the server. All players have the same ability.

Sensor information tells situation of the field from the player's viewpoint. Because the sensing ability is limited, the player program should make decisions using these partial and incomplete information. The player can execute only one command in a simulation step, so that the player program should select the best command in the moment. Limited verbal communication is also available, by which the player can communicate with each other to decide team strategy.

The programs can be written any kind of programming language. The restriction is not to share any information among players except by verbal communication via the server. For example, player programs never communicate by shared memory or direct communication.

Simulator Soccer Server enables a soccer match of two teams of player-programs written by different programming systems. A match using **Soccer Server** is carried out by a client-server system: **Soccer Server** provides a virtual soccer field (Fig. 9) and simulates movements of players and a ball. Clients (player-programs) are brains of players and control their actions via a computer network.

A client connects with the server by a UDP/IP socket. Using the socket, the client sends commands to control an assigned player and receives information from player's sensors. Each client can control only one player, so a team consists of a maximum of 11 client programs⁴.

In the competition, Soccer Server version 3.28 was used the official simulator.

Protocol All communication between the server and each client is done by using ASCII strings. The protocol of the communication consists of *control commands* and *sensor information*:

Control commands are messages sent from a client to the server to control actions of the client's player. There are six kinds of control commands, `turn`, `dash`, `kick`, `say`, `change_view` and `move`. `Turn`, `dash`, and `kick` are used for

⁴ In order to test various kind of systems, we permit a client to control multiple players when control modules are separated logically from each other in the client. This is also a gentleman's agreement.

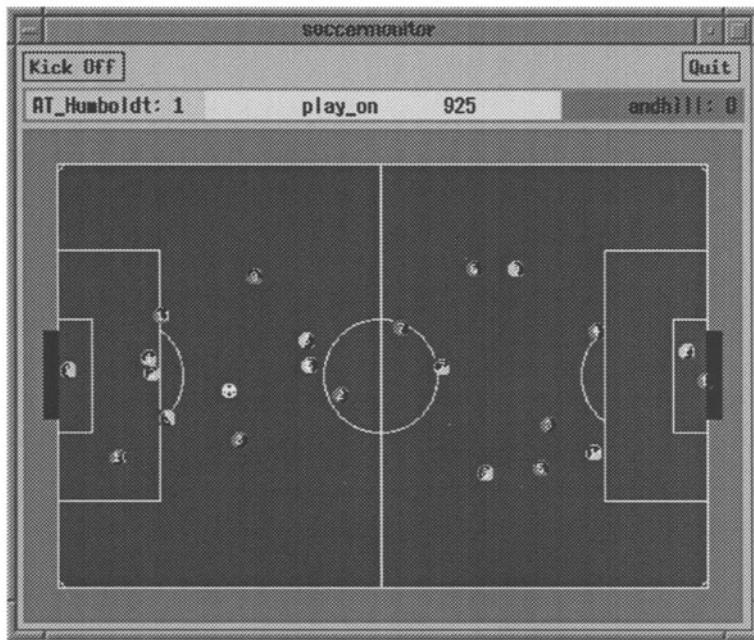


Fig. 9. Soccer Server (A scene of the final match of RoboCup-97)

controlling the client's player during a match. **Say** is used for communication. **Change_view** is used for changing the mode of the visual sensor. **Move** is used to move the player to a certain position before kick-off.

(move *X* *Y*) Move the player to the position (X, Y) . The origin is the center mark, and the X-axis and Y-axis are toward the opponent's goal and the right side respectively. So, usually *X* is negative to locate the player in its own side of the field.

This command is available only in the **before_kick_off** mode.

(turn *Moment*) Change the direction of the player according to *Moment*. *Moment* should be $-180 \sim 180$. Actual change of the direction is reduced when the player is moving quickly.

(dash *Power*) Increase the velocity of the player in the direction it's facing by *Power*. *Power* should be $-30 \sim 100$. (The range is modifiable by the parameter file.)

(kick *Power* *Direction*) Kick the ball by *Power* to the *Direction* if the ball is near enough (the distance to the ball is less than $1.0 + \text{ball_size} + \text{player_size}$). *Power* should be $-30 \sim 100$, and *Direction* should be $-180 \sim 180$. (The ranges are modifiable by the parameter file.)

(say *Message*) Broadcast *Message* to all players. *Message* is informed immediately to clients using a **(near ...)** format described below. *Message* must be

a string whose length is less than 512, and consist of alphanumeric characters and the symbols “+-*/_.() ”.

(change_view ANGLE_WIDTH QUALITY) Change angle of view cone and quality of visual information. *ANGLE_WIDTH* must be one of **wide** (=180 degrees), **normal** (= 90 degrees) and **narrow** (=45 degrees). *QUALITY* must be one of **high** and **low**. In the case of **high** quality, the server begins to send detailed information about positions of objects to the client. In the case of **low** quality, the server begins to send reduced information about positions (only directions) of objects to the client. Default values of angle and quality are **normal** and **high** respectively.

On the other hand, the frequency of visual information sent by the server changes according to the angle and the quality: In the case that the angle is normal and the quality is high, the information is sent every 300 milli-sec. (The interval is modifiable by specifying **send_step** in the parameter file.) When the angle changes to wide the frequency is halved, and when the angle changes to narrow the frequency is doubled. When the quality is low, the frequency is doubled. For example, when the angle is narrow and the quality is low, the information is sent every 75 milli-sec.

Sensor information are messages sent from the server to a client to inform the situation of the field from the viewpoint of the player. There are two kinds of sensor information, **see** (visual information) and **hear** (verbal information).

(see Time ObjInfo ObjInfo ...) Transmit visual information. *Time* indicates the current time. *ObjInfo* is information about a visible object, whose format is:

```
(ObjName Distance Direction DistChng DirChng)
ObjName ::= (player Teamname UNum)
          | (goal Side)
          | (ball)
          | (flag [l|c|r] [t|b])
          | (flag p [l|r] [t|c|b])
          | (line [l|r|t|b])
          | (Player) | (Goal) | (Ball) | (Flag) |
```

where *Distance* and *Direction* are relative distance and direction of the object respectively, and *DistChng* and *DirChng* are respectively the radius- and angle-components of change of the relative position of the object. *DistChng* and *DirChng* are not exact values but are sketchy values. If the object is not movable, *DistChng* and *DirChng* are omitted.

The characters “**l**, **c**, **r**, **t**, **b**” in flags and lines mean left, center, right, top and bottom respectively, that is positions of flags. So, **(flag l t)** means the flag on upper-left corner, and **(flag c b)** means the flag on the lower end of the center line. **(flag p ...)** is virtual flags on penalty area. So, **(flag p l t)** means the position of the upper-right corner of the left penalty area.

In the case of **(line ...)**, *Distance* is the distance to the point there the center line of the player’s view crosses the line, and *Direction* is the direction of line.

As the distance to a player is larger and larger, more and more information about the player is lost. Specifically, *UNum* is lost in the case of farther than a certain distance, and *Teamname* is lost from players who are very far. The quality of information is also changed by `change_view` commands. The frequency of sending this information depends on the quality of information transmitted. See the description of `change_view`.

(**hear Time Direction Message**) Inform verbal information. *Direction* is the direction of the sender. In the case that the sender is the client itself, *Direction* is ‘self’. *Time* indicates the current time. This message is sent immediately when a client sends (**say Message**) command. Judgments of the referee are also broadcast using this form. In this case, *Direction* is ‘referee’.

For more detail, see the manual of **Soccer Server [SSe]**.

Performance of Player Each player receives visual information every 300ms, and acts one of turn, dash and kick actions every 100ms. This sensing/action ratio was an important point of discussion in deciding the regulation.

There were two opinions about it:

- **more frequent sensing:** In order to react changes of the environment reasonably, the player require more sensing information. Because there are 21 other players on the field, the player should receive more frequent sensing information rather than action.
- **less frequent sensing:** Compared with action commands, sensing information is highly abstracted. Therefore sensing information should be come less frequently. For example, a ‘dash’ command is usually used with a ‘turn’ command. On the other hand, ‘see’ information includes all information about object in the visible area.

We decided the ratio as 300ms/100ms for this competition. This ratio will be adjusted for future competitions based on results of RoboCup-97.

In visual sensing, a player can see only objects in front of itself. Width of visual cone is 90 degree. The width can be changed by `change_view` commands, but it depends on the frequency and quality of the visual information.

Verbal information also has limitation. A message said by a player reaches only 50m, so that players far from the player can not hear it. Moreover, a player has a capacity of hearing messages. The player can hear only a message from teammates and a message from opponents every 200ms. Therefore, communication among players should be done effectively.

Stamina is another factor of Player’s performance. Each player has a certain stamina that decreases when the player dashes, and increases constantly until it reaches the maximum. For example, when a client sends a command (`dash 80`), the player’s stamina decreases 80 points. If the stamina is less than 80 points,

the power of dash is reduced to the same value as the stamina. The stamina increases 20 points in every simulation cycle until the stamina is 2000 points.

Refereeing A match were controlled by a referee module in **Soccer Server** according to the following rules:

- **Goal:** A team get a goal, then the referee suspend the match 5 seconds, and restart it from the center mark.
- **Out of Field:** When the ball goes out of the field, the referee restart the match from the suitable points (touch lines, corner area or goal area).
- **Halftime and Time-Up:** The referee suspends a match when the first or the second half finishes. The length of each half is about 5 minutes (3000 simulation-cycles). If the match is draw after the second half, the match is extended until another goal (golden-goal style).
- **Clearance:** At restart of the match, the referee control the defending players not to enter the area of 9.15m from the ball.

These judgments were done automatically.

A human referee also controlled the match, because the following rules requires subjected judgement.

- If many players rush into the ball and the match *seems* to stagnate, then the referee restarts the match by a drop ball.
- If a team *seem* not to be able to restart the match by kick-off or free-kick, then the referee restarts the match by a drop ball.
- If a team *seem* to try to cover their own goal by players *intensively*, then the referee restarts the match in front of the goal.

3.3 Conditions of the Competition

Participants Finally, 29 teams from 10 country participated in the simulation league. Table 3 is a list of teams.

The competition was carried out in two stages: In the first stage, 29 teams were divided into 8 groups. Each group consisted of 3 or 4 teams. In a group, a team had a match with each other team in the same group (round-robin system). The first and second places in each group were qualified to go up to the second stages. The second stage was single elimination system of 16 teams.

We had three days for test run before the formal competition. During these days, every team did final tune-up and had test-matches with each other. By these test-matches participants exchange the information about their teams and discussed about technology.

Equipments For the competition, the following infrastructure were prepared.

- 36 sets of Sun Ultra Sparc 1 for running client (player) programs
- 4 sets of Sun Ultra Sparc 2 for running **Soccer Server**

Group A	Group E
LAI (Univ. Carlos III De Madrid), Spain	Pagello (Univ. of Padua), Italy
FC Mellon (CMU), USA	HAARLEM (Chukyo Univ.), Japan
RM Knights (RMIT), Australia	Orient (Toyo Univ.), Japan
Ichimura (Kinki Univ.), Japan	
Group B	Group F
Rieksi (Univ. of Oulu), Finland	UBC Dynamo (Univ. Britich Columbia), Canada
CMUnited (CMU), USA	Luke (Univ. of Maryland), USA
The Headless Chickens (RMIT), Australia	Ogalets (Univ. of Tokyo), Japan
NIT-Stones (Nagoya Inst. of Tech.), Japan	TUT (Toyohashi Univ. of Tech.), Japan
Group C	Group G
MICROB (Univ. PARIS 6), France	Christensen (Charlmers Univ. Technology), Sweden
Balch (Georgia Inst. of Tech.), USA	Team GAMMA (ETL), Japan
Project MAGI (Aoyama Univ.), Japan	Kosue (Kinki Univ.), Japan
Ohta (Tokyo Inst. of Tech.), Japan	
Group D	Group H
AT Humboldt (Humboldt Univ.), Germany	ISIS (USC/ISI), USA
Team Sicily (Stanford Univ.), USA	TEAM Garbage Collectors (private), Japan
Kasuga-bit (Chubu Univ.), Japan	I&W (Waseda Univ.), Japan
AndHill (Tokyo Inst. of Tech.), Japan	

Table 3. Team List

- 4 sets of SGI Indigo 2 for running 3D monitor
- Fast Ether Switch to connect all machines

Each team could use 4 sets of Ultra Sparc 1 in the first stage, and 11 sets in the second stage. But, most teams needed only 4 machines to run 11 clients, because their programs were written in C and C++ directly so that they required small CPU power. We expect that more CPU power will be required in future RoboCup, because participants will use more complicated strategies.

3.4 Results and Team Features

Results The champion of RoboCup-97 simulation is

World champion: AT-Humboldt, Humboldt University, Germany.

The Second Place: AndHill, Tokyo Institute of Technology, Japan,

The Third Place: ISIS, Information Science Institute / University of Southern California, USA,

The Fourth Place: CMUnited, Carnegie Mellon University, USA.

Preliminary Games Results of preliminary rounds are as follows:

Group A:

A-1: LAI: Universidad Carlos III De Madrid, Spain

A-2: FC Mellon: Carnegie Mellon University, U.S.A.

A-3: RM Knights: Royal Merbolune Institute of Technology, Australia

A-4: Ichimura: Kinki University, Japan

vs.	A-1	A-2	A-3	A-4	Win/Lost	GF - GA	Total Goals	Rank
A-1	—	1 - 9	10 - 0	0 - 8	1/2	-6	11	3
A-2	9 - 1	—	16 - 0	6 - 5	3/0	9	15	1
A-3	0 - 10	0 - 16	—	0 - 12	0/3	-22	0	4
A-4	8 - 0	5 - 6	12 - 0	—	2/1	19	25	2

Group B:

B-1: Riekki: University of Oulu, Finland

B-2: CMUnited: Carnegie Mellon University, U.S.A.

B-3: The Headless Chickens: Royal Merbolune Institute of Technology, Australia

B-4: NIT-Stones: Nagoya Institute of Technology, Japan

vs.	B-1	B-2	B-3	B-4	Win/Lost	GF - GA	Total Goals	Rank
B-1	—	3 - 4	1 - 0	7 - 0	2/1	7	11	2
B-2	4 - 3	—	13 - 0	20 - 0	3/0	34	37	1
B-3	0 - 1	0 - 13	—	8 - 1	1/2	-7	8	3
B-4	0 - 7	0 - 20	1 - 8	—	0/3	-34	1	4

Group C:

C-1: MICROB: Universite Paris-6, France

C-2: Balch: Georgia Institute of Technology, U.S.A.

C-3: Project MAGI: Aoyama Gakuin University, Japan

C-4: Ohta: Tokyo Institute of Technology, Japan

vs	C-1	C-2	C-3	C-4	Win/Lost	GF - GA	Total Goals	Rank
C-1	---	7 - 1	7 - 9	7 - 14	1/2	-3	21	3
C-2	1 - 7	—	2 - 11	0 - 15	0/3	-30	3	4
C-3	9 - 7	11 - 2	—	1 - 20	2/1	-8	21	2
C-4	14 - 7	15 - 0	20 - 1	—	3/0	41	49	1

Group D:

D-1: AT-Humboldt: Humboldt University, Germany

D-2: Team Sicily: Stanford University, U.S.A.

D-3: Kasuga-bit: Chubu University, Japan

D-4: AndoHill: Tokyo Institute of Technology, Japan

vs	D-1	D-2	D-3	D-4	Win/Lost	GF - GA	Total Goals	Rank
D-1	—	25 - 0	23 - 0	7 - 4	3/0	51	55	1
D-2	0 - 25	—	1 - 4	0 - 23	0/3	-51	1	4
D-3	0 - 23	4 - 1	—	0 - 23	1/2	-43	4	3
D-4	4 - 7	23 - 0	23 - 0	—	2/1	43	50	2

Group E:

- E-1:** Pagello: University of Padua, Italy
E-2: (Cancelled)
E-3: HAARLEM: Chukyo University, Japan
E-4: Orient: Toyo University, Japan

vs	E-1	E-2	E-3	E-4	Win/Lost	GF - GA	Total Goals	Rank
E-1	—	-	7 - 2	4 - 1	2/0	8	11	1
E-2	-	—	-	-				
E-3	2 - 7	-	—	4 - 2	1/1	-3	6	2
E-4	1 - 4	-	2 - 4	—	0/2	-5	3	3

Group F:

- F-1:** UBC Dynamo: University of British Columbia, Canada
F-2: Luke: University of Maryland, U.S.A.
F-3: Ogalets: University of Tokyo, Japan
F-4: TUT: Toyohashi Science and Technology University, Japan

vs	F-1	F-2	F-3	F-4	Win/Lost	GF - GA	Total Goals	Rank
F-1	—	2 - 5	1 - 7	9 - 0	1/2	0	12	3
F-2	5 - 2	—	1 - 6	17 - 0	2/1	15	23	2
F-3	7 - 1	6 - 1	—	18 - 0	3/0	29	31	1
F-4	0 - 9	0 - 17	0 - 18	—	0/3	-44	0	4

Group G:

- G-1:** Christensen: Chalmers University of Technology, Sweden
G-2: (Cancelled)
G-3: Team GAMMA: ElectroTechnical Laboratory, Japan
G-4: Kosue: Kinki University, Japan

vs	G-1	G-2	G-3	G-4	Win/Lost	GF - GA	Total Goals	Rank
G-1	—	-	9 - 0	2 - 10	1/1	1	11	2
G-2	-	—	-	-				
G-3	0 - 9	-	—	0 - 16	0/2	-25	0	3
G-4	10 - 2	-	16 - 0	—	2/0	24	26	1

Group H:

- H-1:** (Cancelled)
H-2: ISIS: ISI/University of Southern California, U.S.A.
H-3: Team-GC, Private Entry, Japan
H-4: Inoe&Wilkin: Waseda University, Japan

vs	H-1	H-2	H-3	H-4	Win/Lost	GF - GA	Total Goals	Rank
H-1	—	-	-	-				
H-2	-	—	17 - 0	10 - 0	2/0	27	27	1
H-3	-	0 - 17	—	4 - 0	1/1	-13	4	2
H-4	-	0 - 10	0 - 4	—	0/2	-14	0	3

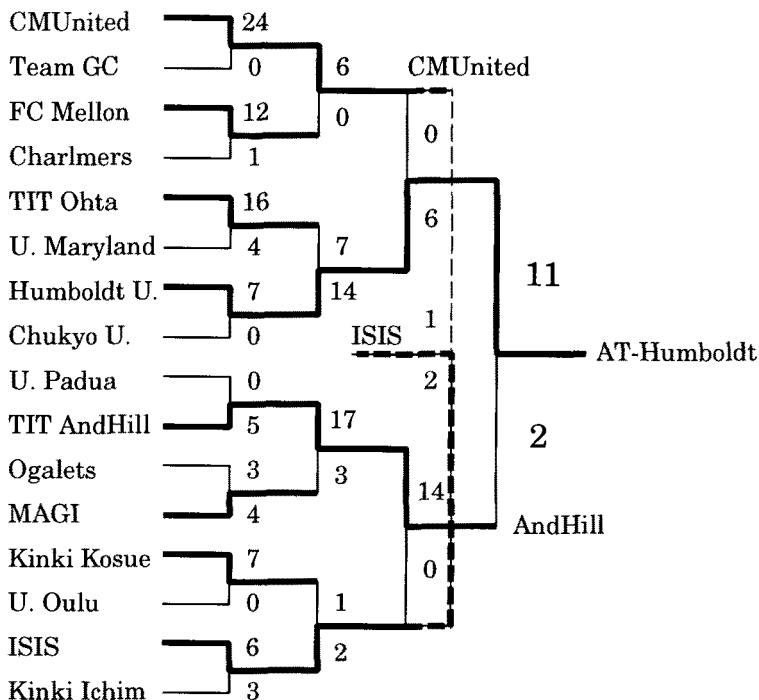


Fig. 10. Results of Finals Games

Finals

General Analysis General impression of the competition were as follows:

- Most of matches in the first round were one-side games. In a typical case, the score was 23–0. The main reason was that there is big difference in player's individual skill. In the next RoboCup, such difference will become small, because most of teams of RoboCup-97 agreed with it to make their programs in public, so that we can share the know-how of the good technique.
- The tactics of defense was not so good as one of offense in most teams. In order to defense opponent attack effectively, a team must have the model of tactics of opponent offense. But, because RoboCup-97 was the first competition, every team did not know what kind of tactics were possible. Now, we can see replay of matches, so that we can build models of opponent. Using such models, defense will become more effective in the next RoboCup.

Both finalists were in the same group (Group D) in the first round. At their first match in Group D, each team already recognized another team as a rival in the final, so that AndHill used their weaker program for the first match. Then they used stronger one for the final match. Moreover, AndHill tried to change

their formation in the second half of the final match. Their efforts improve the performance of their team, so that the score of the second half was closer than the first half. But finally AndHill defeated against AT Humboldt.

Of course, these strategies (changing tactics between matches and during a match) were done by human. It may be challenging topic to realize these strategies by program.

A process of each match was recorded in a log file. You can see replay of matches by the log files using `logplayer` included in Soccer Server distribution. The log files of all matches are FTP available from the following site:

`ftp://ci.etl.go.jp/pub/soccer/log/RoboCup97/`

also, detailed result of the competition is shown in the following homepage:

`http://ci.etl.go.jp/~noda/soccer/RoboCup97/result.html`

Features of Teams There were three types of approaches to build player programs: agent programming, multi-agent system, and learning.

Agent Programming Because situations of the field simulated by Soccer Server changes dynamically, each player should be programmed as an agent in a dynamic environment. A main issue in this programming is realtime-ness. A player should perform a suitable play for the current situation as soon as possible. For example, when the ball is in front of the own goal and the opponent player is coming, then the goalie should kick the ball away as soon as possible before searching a teammate who is the best receiver of pass.

Many participants used reactive systems to realize the above behavior. These reactive systems are based on or inspired by Brooks' subsumption architecture. They also used planning systems to make high-level decision of play like selection of pass-receivers. For example, AT Humboldt, Univ. of Padua, Univ. of Oulu, Univ. of Carlos III, etc. used the combination of reactive systems and planning systems.

Multi-Agent System Because soccer is a team game, a couple of teams were programmed based on multi-agent systems to describe team-plays. USC-ISI, CMU, RMIT, Stanford Univ, etc. attacked this topic. There are following issues in this topic:

- How to define team-plays and decompose them into roles.
- How to assign the roles to players.
- How and what to communicate effectively among players.

USC-ISI is the most successful team in the above teams. They built their programs based on STEAM, which they developed to describe teamwork in a party of helicopter. Remarkable things is that 35% of codes were shared between soccer players and the party of helicopter even though domains of these two application were quite different from each other.

Machine Learning Machine learning was also a major approach for programming player clients. CMU, Tokyo Inst. of Tech., Georgia Inst. of Tech., etc. used machine learning technique. There are couple of points machine learning should be used as follows:

- To improve player's skill of handling the ball, for example, approaching moving ball, or kicking with suitable power.
- To organize formation of players.
- To acquire suitable conditions a certain play, for example, which teammate to pass to.

Peter Stone and Manuela Veloso from CMU used a couple of learning techniques for various levels of play. They used neural network learning to improve basic skill, while they used decision trees learning for high-level collaboration.

Genetic Programming The most impressive technology used in the simulation league was Genetic Programming (GP). Sean Luke and his colleague from University Maryland used GP to improve the behavior of players.

At first, they prepared pieces of program like (`kick POWER DIR`), (`turn POWER`), (`plus X Y`), (`if COND THEN ELSE`), and so on. They combined these pieces randomly and built programs to control players. Then they made teams of the players and made them compete with each other. Using the result of the competition, they selected teams that got good score, and generated new programs by the GP method from the selected teams. They repeated this cycle, and evolved player programs. Of course, this evolution was done before the RoboCup competition, and they participated the competition using programs in the final generation.

They also reported interesting feature of changes of behavior during the evolution. In the first generation of the programs, most of players did not see the ball. After some alternations of generations, all players became to chase the ball. Then, some players became to defense their own goals after more alternations. Finally, all player in a team spread on the field and passed the ball smoothly. Interesting thing is that these changes are quite similar to old history of human soccer.

The Scientific Challenge Award was given to Sean Luke for demonstrating the utility of this genetic programming approach.

3.5 Computer vs. Human

An extra exhibition match, the champion team (AT Humboldt) vs. human team, took place after the final match.⁵ The human team consisted of 11 person selected from participants. Each person controlled a player from the console by a simple interface.

The result was that AT Humboldt won in 8 – 1. But this result does not means that computer is strong enough. Because it was the first time for members of the

⁵ This exhibition was organized by Sean Luke and Tucker Balch.

human team to control players directly using the interface, AT Humboldt got goals one-sidedly, so that the score of the first half was 6 – 0. Then human team improved themselves in controlling players in the second half. As a result, the score of the second half is 2 – 1. AT Humboldt was still stronger than human team, but it was close. Every member of the human team said that human would win in the next time. There are a couple of reasons why the human team became nice so quickly.

- The ability of human to learn skills to control players is very high. They improved their skill only 5 minutes.
- The human is very good at modeling opponent tactics. AT Humboldt (and also other computer teams) has few tactics to get goals, so that it was easy to cut passes.
- The human team used communication quite effectively. We used voice for the communication in order to exchange the plan of defense and offense.

These points means that there remain many issues of research on leaning, opponent modeling, and multi-agent systems.

3.6 Open Issues for Future RoboCup Simulation League

Because RoboCup-97 was the first competition, most of works of participants were in progress. There remain following open issues on these works:

- **Complex, Large Scale Teamwork:** Teamwork realized in RoboCup-97 was relatively simple and small. Most of them were done among only 2 or 3 players. On the other hand, human soccer team can perform more complex and large scale teamwork. For example, off-side trap requires synchronized movements of more than 4 players. In such case, it becomes more difficult to make a consensus during players.
- **On-line Learning and Opponent Modeling:** Most of learning techniques used in RoboCup-97 were off-line learning, so that the behaviors of each teams did not change drastically during a match. On the other hands, a team should acquire opponent's model and adapt it during a match, because various types of teams will gather at the RoboCup competition. Therefore on-line learning will become necessary for the next RoboCup.
- **Easily Modifiable Parameter:** The rule of the simulation league permits for participants to change their program during a half-time. In order to use this rule effectively, a player program should be built to be able to be modified easily. However, most of programs in RoboCup-97 were written in C, and various parameters were written in the source code directly. Therefore it was hard to tune parameters during the half-time. For future competition, more flexible systems of programming player will be required.
- **Evolution of Verbal Communication:** Though Sean's GP approach evolved remarkable team-play, it did not evolve language for the communication, because they did not prepare program peaces for the communication in the initial setup of GP. Other researchers using machine learning also did not

try to train players to acquire how to use communication for team-play. This will be the most challenging topic in the future RoboCup.

4 Future

Overall RoboCup-97 was very successful. It made clear scientific issues in developing multiple robots to work collaboratively in real world, and help us identifying technical challenges in bringing multiple robots in the-outside-of-lab set up. Once the goal is set, we expect the progress of technology will be accelerated. Many problems we have recognized during RoboCup-97 will be solved quickly. This is particularly true for real robot league, where uncontrollable physical interaction is the central issue. As we have expected even before RoboCup-97, we reconfirmed the problems and challenges in front of us. On the other hand, results of simulator league clearly demonstrate the strength of AI approach over hard-coded programming. It was very encouraging for the AI community that for the game of this level of complexity, AI-based approach was proven to be more effective than hard-coded hacking.

RoboCup will continue in future until a robot soccer team will beat the human World Cup Soccer champion team. RoboCup-98, The Second Robot World Cup Soccer, will be held in July 1998 during The Third International Conference on Multi-Agent Systems(ICMAS-98) in Paris, France. RoboCup-99 will be held during the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) in Stockholm, Sweden as part of IJCAI-99's special program. We hope RoboCup offers a good target for the 21st century's AI.

References

- [Kitano *et al.*1997a] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa and Hitoshi Matsubara: RoboCup: a challenge problem for AI, AAAI AI magazine Spring 1997, pp.73-85,1997.
- [Noda and Matsubara1996] Itsuki Noda and Hitoshi Matsubara. Soccer server and researches on multi-agent systems. In Hiroaki Kitano, editor, *Proceedings of IROS-96 Workshop on RoboCup*, pages 1-7, Nov. 1996.
- [Kitano *et al.*1997b] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The robocup synthetic agent challenge 97. In *Proc. of IJCAI-97*, pages 24-29, Aug. 1997.
- [SSe] Soccer Server Homepage. <http://ci.etl.go.jp/~noda/soccer/server.html>.

The RoboCup Physical Agent Challenge: Goals and Protocols for Phase I

Minoru Asada¹, Peter Stone², Hiroaki Kitano³, Alexis Drogoul⁴,
Dominique Duhaut⁵, Manuela Veloso², Hajime Asama⁶, and Sho'ji Suzuki¹

¹ Department of Adaptive Machine Systems, Osaka University,
Suita, Osaka, 565 Japan

² School of Computer Science, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, Pennsylvania 15213 USA

³ Sony Computer Science Laboratory,
3-14-13 Higashi-Gotanda, Shinagawa, Tokyo, 141 Japan

⁴ LIP6, Universite Paris 6,
Case 169, 4 Place Jussieu 75252 Paris CEDEX 05, France

⁵ Laboratoire de Robotique de Paris, U.V.S.Q.
10-12, avenue de l'Europe 78140 Vélizy, France

⁶ Institute for Chemical and Physical Research (Riken),
2-1 Hirosawa, Wako, Saitama, 351-01 Japan

Abstract. Traditional AI research has not given due attention to the important role that physical bodies play for agents as their interactions produce complex emergent behaviors to achieve goals in the dynamic real world. The RoboCup Physical Agent Challenge provides a good test-bed for studying how physical bodies play a significant role in realizing intelligent behaviors using the RoboCup framework [Kitano, *et al.*, 95]. In order for the robots to play a soccer game reasonably well, a wide range of technologies needs to be integrated and a number of technical breakthroughs must be made. In this paper, we present three challenging tasks as the RoboCup Physical Agent Challenge Phase I: (1) moving the ball to the specified area (shooting, passing, and dribbling) with no, stationary, or moving obstacles, (2) catching the ball from an opponent or a teammate (receiving, goal-keeping, and intercepting), and (3) passing the ball between two players. The first two are concerned with single agent skills while the third one is related to a simple cooperative behavior. Motivation for these challenges and evaluation methodology are given.

1 Introduction

The ultimate goal in AI, and probably in robotics, is to build intelligent systems capable of displaying complex behaviors to accomplish the given tasks through interactions with a dynamically changing physical world. Traditional AI research has been mainly pursuing the methodology of symbol manipulations to be used in knowledge acquisition and representation and reasoning about it with little attention to intelligent behavior in dynamic real worlds [Brooks, 1991]. On the other hand, in robotics much more emphasis has been put on the issues

of designing and building hardware systems and their controls. However, recent topics spread over the two areas include design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning and planning, intelligent robotics, sensor-fusion, and behavior learning. These topics expose new aspects with which traditional approaches seem unable to cope.

In coping with these issues and finally achieving the ultimate goal, physical bodies play the important role of bringing the system into *meaningful* interaction with the physical environment – complex and uncertain, but with an automatically consistent set of natural constraints. This facilitates the correct agent design, learning from the environment, and rich meaningful agent interaction. The meanings of “having a physical body” can be summarized as follows:

1. Sensing and acting capabilities are not separable, but tightly coupled.
2. In order to accomplish the given tasks, the sensor and actuator spaces should be abstracted under resource-bounded conditions (memory, processing power, controller etc.).
3. The abstraction of the sensor and actuator spaces depends on both the fundamental embodiments inside the agents and the experiences (interactions with their environments).
4. The consequence of the abstraction is the agent-based subjective representation of the environment, and it can be evaluated by the consequences of behaviors.
5. In the real world, both inter-agent and agent-environment interactions are asynchronous, parallel and arbitrarily complex. There is no justification for adopting a particular top-down abstraction level for simulation such as a global clock ‘tick’, observable information about other agents, modes of interaction among agents, or even physical phenomena like slippage, as any seemingly insignificant parameter can sometimes take over and affect the global multi-agent behavior.
6. Natural complexity of physical interaction automatically generates reliable sample distributions of input data for learning, rather than from an *a priori* Gaussian distribution in simulations which does not always correctly capture the characteristics of the system.

Even though we should advocate the importance of “having a physical body,” it seems required to show that the system performs well coping with new issues in a concrete task domain. In other words, we need a standard problem which people regard as a new one that expose various various aspects of intelligent behaviors in real worlds.

RoboCup (The World Cup Robot Soccer Games:[Kitano, *et al.*, 95; Kitano, *et al.*, 97]) is an attempt to promote AI and robotics research by providing a common task for evaluation of various theories, algorithms, and agent architectures, and was proposed as a new standard problem. Not only the integration of a wide range of technologies but also accomplishment of technical breakthroughs are required for the robots to play a soccer game reasonably well. RoboCup consists of three competition tracks: (1) **Real Robot League:** using physical robots

to play soccer games, (2) **Software Robot League:** using software agents to play soccer games on an official soccer server over the network, and (3) **Expert Robot Competition:** competition of robots which have special skills, but are not able to play a game.

In this paper, we propose the RoboCup Physical Agent Challenges as new research issues of physical agents. First, we show how the challenge is significant for physical agent research with long range issues. Then, we show mid- and short-term issues which spans from simple skill acquisition to a simple teamwork behavior. In particular, we pick two single agent skills (ball moving and ball catching) and one cooperative skill (passing the ball between two players) with different situations (no obstacles, stationary or moving obstacles) as the RoboCup physical agent challenge Phase I. We describe how these techniques can be evaluated in terms of various kinds of design issues.

2 Research Issues of RoboCup Physical Agent Track

RoboCup physical agent challenges are summarized as follows:

- **Perception:** The player should observe in real-time the behaviors of other objects which it cannot completely predict or control in order to take an action to deal with them. Such objects include a ball, an opponent, and in some sense, a common-side player, and further a judge. Capabilities of wide range perception, discrimination of other agents, and estimation of their locations and motions coping with occlusions are needed. Such perception is a basic technology to expand robotic applications.
- **Actions:** Controlling a ball in a multi-agent situation introduces a new set of problems even in robotics, where traditionally the manipulated objects are mostly stationary or follow predictable trajectories. Some of the immediate challenges are discussed in depth in the following sections. Long term challenges including: a) controlling a 3D (bouncing or flying) motion of a ball, b) soccer playing with limbed (quadruped, hexapods, or even bipeds) robots, and c) generating "faint" actions assuming that the opponent has a capability of action observation and prediction.
- **Situation and Behavior:** The task domain itself is simple, but infinitely many situations can occur in accordance with dynamic changes of the relationships in position and relative motion between the ball, the goal, and the players, and the context of the game. The optimal behavior changes from one situation to another. Since our goal is a soccer playing team, we need abilities beyond simple reflexive behaviors. For example, we need situation understanding, tactics selection and modification, minimum communication with team mates, teamwork behaviors acquired through practical training. These issues are closely related to the cognitive issues such as organization of spatio-temporal memory of the world and categorization of sets of motor behaviors into skills (symbols) grounded by the physical bodies.
- **Real-Time:** Since the situation rapidly changes according to motions of the ball and other players, there is no time to carefully analyze the situation and

deliberate a plan. Therefore, the player should take an action such as kicking a ball immediately or dribbling it if surrounded by no opponents in order to wait for a better situation, or moving to a certain position in order to facilitate future or collaborative actions.

- **Platform:** The initial barrier for physical agent track challengers would be to build a team of robotic platforms for soccer playing. A platform should have a power source, high speed and quick response mobility, a ball kicking capability, sensors for a ball, the goal, the soccer court, self position, and other players' position and movements, on-board and/or remote computers, and a wireless communication for at least accepting the judge's commands. Integrating all these into a size-constrained physical structure while achieving entire real-time performance is already a challenge which requires constrained optimization of a large number of design parameters. Participating in RoboCup requires producing and maintaining multiple such copies. A valuable short range challenge would be to propose a standard robotic platform design for robot soccer. Although this is not exactly an AI issue, such a standard platform will surely facilitate many AI researchers potentially wanting to test their ideas on real robots.

These challenges described above are significantly long-term ones along the way to realizing a good soccer-playing robot team. Some of them will take a few decades to meet. However, due to the clarity of the final target, several subgoals can be derived, which define mid term and short term challenges. One of the major reasons that RoboCup is attractive to so many researchers is that it requires the integration of a broad range of technologies into a team of complete agents, as opposed to a task-specific functional module. The long term research issues are too broad to compile as a list of specific items. Nevertheless, the challenges involve a broad range of technological issues ranging from the development of physical components, such as high performance batteries and motors, to highly intelligent real-time perception and control software.

The mid-term technical challenges, which are the target for the next 10 years, can be made more concrete, and a partial list of specific topics can be compiled. Following is a partial list of research areas involved in RoboCup physical agent track, mainly targeted for the mid term time span: (1) agent architecture in general, (2) implementation of real-time and robust sensing, (3) realization of stable and high-speed robot control, (4) sensor fusion, (5) behavior learning for multi agent environments, and (6) cooperation in dynamic environments.

The RoboCup Physical Agent Challenge should be understood in the context of larger and longer range challenges, rather than as a one-shot challenge. Thus, we wish to provide a series of short-term challenges, which naturally leads to the accomplishment of the mid- term and long-term challenges.

3 Overview of The RoboCup Physical Agent Challenge Phase I

For the RoboCup Physical Agent Challenge Phase I, we offer three specific challenges, essential not only for RoboCup but also for general mobile robotics research.

The fundamental issue for researchers who wish to build real robot systems to play a soccer game in RoboCup is how to obtain basic skills to control a ball in various kinds of situations. Typical examples are to shoot the ball into the goal, to intercept the ball from an opponent, and to pass the ball to a teammate. These skills are needed to realize cooperative behaviors with team mates and competitive ones against opponents in soccer game. Among basic skills to control the ball, we selected three challenges as the RoboCup Physical Agent Challenge Phase I:

1. moving the ball to the specified area with no, stationary, or moving obstacles,
2. catching the ball from an opponent or a teammate, and
3. passing the ball between two players.

These three challenges have many variations in different kinds of situations such as passing, shooting, dribbling, receiving, and intercepting the ball with/without opponents whose defensive skills vary from amateur to professional levels. Although they seem very specific to RoboCup, these challenges can be regarded as very general tasks in the field of mobile robotics research in a flat terrain environment. Since target reaching, obstacle avoidance, and their coordination are basic tasks in the area, the task of shooting the ball while avoiding opponents that try to block the player should be ranked as one of the most difficult challenges in the area. Once the robot succeeds in acquiring these skills, it can move anything to anywhere.

In another aspect, these three challenges can be regarded as a sequence of one task which leads to an increase of the complexity of the internal representation according to the complexity of the environment [Asada, 1996].

In the case of visual sensing, the agent can discriminate the static environment (and its own body if observed) from others by directly correlating the motor commands the agent sent and the visual information observed during the motor command executions. In other words, such observation can be classified as a self and stationary environment. In contrast, other active agents do not have a simple and straightforward relationship with the self motions. In the early stage, they are treated as noise or disturbance because of not having direct visual correlation with the self motor commands. Later, they can be found as having more complicated and higher correlation (cooperation, competition, and others). As a result, the complexity is drastically increased especially since between the two there is a ball which can be stationary or moving as a result of self or other agent motions.

The complexities of both the environment and the internal representation of the agent can be categorized as a cognitive issue in general, and such an issue is

naturally involved in this challenge. In the following, we describe the challenges more concretely.

4 Task-I: Ball Moving

4.1 Objectives

The objective of this challenge is to check how the most fundamental skill of moving a ball to the specified area under several conditions with no (**Level I**), stationary (**Level II**), or moving (**Level III**) obstacles in the field can be acquired in various kinds of agent architectures, and to evaluate merits and demerit of realized skills using the standard tasks.

The specifications of the ball and the surface of the field is an issue common to all the challenges in the physical agent track. In order to emerge various behaviors, the field surface should not be so rough as to prevent the ball from rolling, but not so smooth that there is no friction. The former would rule out kicking or passing, and the latter, dribbling.

Since there are few variations of the task environment in **Level I**, agent architecture, and sensing in particular, is a major focus. In **Level II** motion control is the central issue, and in **Level III** prediction of the motion of obstacles is the key issue.

4.2 Technical Issues

Vision General computer vision and robot vision issues are too broad to deal with here. Finding and tracking independently moving objects (ball, players, judges) and estimating their motion parameters (2-D and further 3-D) from complicated background (field lines, goals, corner poles, flags waved by the supporters in the stadium) is too difficult for the current computer and robot vision technologies to completely perform in real-time.

In order to focus on skill acquisition, visual image processing should be drastically simplified. Discrimination by color information such as a red ball, a blue goal, a yellow opponent makes it easy to find and track objects in real-time [Asada *et al.*, 1996b]. Nevertheless, robust color discrimination is a tough problem because digitized signals are so naive against the slight changes of lighting conditions. In the case of remote (wireless) processing, increased noise due to environmental factors causes fatal errors in image processing. Currently, human programmers adjust key parameters used in discriminating colored objects on site. Self calibration methods should be developed, which will be able to expand the general scope of image processing applications.

Visual tracking hardware based on image intensity correlation inside a window region can be used to find and track objects from the complicated background by setting the initial windows [Inoue, *et al.*, 92]. Currently, a color tracking version is commercially available. As long as the initialized color pattern

inside each window does not change much, tracking is almost successful. Coping with pattern changes due to lighting conditions and occlusions is one of the central issues in applying this type of vision hardware.

As long as the vision system can cope with the above issues, and capture the images of both the specified area (the target) and the ball, there might be no problem [Nakamura and Asada, 1995; Nakamura and Asada, 1996]. To prevent the agent from losing the target, and/or the ball (in **Level II** and **III**, obstacles, too), an active vision system with panning and tilting motions seems preferable, but this makes the control system more complicated and introduces the spatial memory organization problem for keeping track of lost objects. A more practical way is to use a wider-angle lens. One extreme of this sort is to use the omnidirectional vision system to capture the image all around the agent. This sort of lens seems very useful not only for acquiring the basic skills but also for realizing cooperative behaviors in multi agent environments. Currently this type of lens is commercially available as spherical and hyperboloidal ones [Ishiguro, 96].

Other perception In the case of other sensing strategies, the agent should find the ball, (in **Level II and III**, obstacles, too) and know what the target is. Beside vision, typical sensors used in mobile robot research are range finders (e.g., sonar) and contact sensors (e.g., bumpers). However, it seems difficult for each or any combination among them to discriminate the ball (obstacles, too in higher levels) and the target unless special equipment such as transmitter is positioned inside the ball or the target, or a global positioning system besides on-board sensing and communication lines are used to inform the positions of all agents. The simplest case is no on-board sensing but only a global positioning system, which is adopted in the small robot league in the physical agent track because on-board sensing facilities are limited due to its size regulation.

In **Level II** and **III**, requirements include an obstacle avoidance behavior and the coordination of this behavior with a ball-carrying (or passing/shooting) behavior. One good strategy is assign the sensor roles in advance. For example, sonar and bumper sensors are used for obstacle avoidance while vision sensor is used for the target reaching. One can make the robot learn to assign the sensor roles [Nakamura *et al.*, 1996].

Action As described in section 2, total balance of the whole system is a key issue to the robot design. In order for the system to facilitate various kinds of behaviors, a more complicated mechanical system and its sophisticated control techniques are necessary. We should start with a simpler one and then step up. The simplest case is to use just a car-like vehicle which has only two DOFs (degrees of freedom, for example forward and turn), and pushes the ball to the target (dribbling).

The target can be just a location, the goal (shooting), and one of the team mates (passing). In the case of location, a dribbling skill to carry the ball to the location might be sufficient. In the latter case, the task is to kick the ball into the desired direction without caring about final position of the ball. To discriminate

it from a simple dribbling skill, we may need more DOFs to realize a kick motion with one foot (or what we may call arm). In the case of passing, velocity control of the ball might be a technical issue because one of the team mates to be passed to is not stationary but moving.

In **Levels II** and **III**, requirements include an obstacle avoidance behavior and the coordination of this behavior with a ball-carrying (or passing/shooting) behavior. To smoothly switch two behaviors, the robot should slow down, which increases the possibility that the opponent will take the ball. To avoid these situations, the robot can quickly switch behaviors, which causes instability of the robot motion. One might use the omni-directionally movable vehicle based on a sophisticated mechanism [Asama *et al.*, 1996]. For example, the vehicle could move to any direction anytime. In addition to the motion control problem, there are more issues to be considered such as how to coordinate these two behaviors (switching conditions) [Uchibe *et al.*, 1996].

Mapping from perception to action There are several approaches to implementing the control mechanisms which perform the given task. A conventional approach is first to reconstruct the geometrical model of the environment (ball, goal, other agents etc.), then deliberate a plan, and finally execute the plan. However, this sort of approach is not suitable for the dynamically changing game environment due to its time-consuming reconstruction process.

A look-up table (LUT) indicating the mapping from perception to action by whatever method seems suitable for quick action selection. One can make such an LUT by hand-coding given *a priori*, precise knowledge of the environment (the ball, the goals, and other agents) and the agent model (kinematics/dynamics). In a simple task domain, a human programmer can do that to some extent, but seems difficult to cope with all possible situations completely. An opposite approach is learning to decide action selection given almost no *a priori* knowledge. Between exist several variations with more or less knowledge. The approaches are summarized as follows: (1) complete hand-coding (no learning), (2) parameter tuning given the structural (qualitative) knowledge (self calibration), (3) Subtask learning fit together in a layered fashion [Stone and Veloso, 1997] (4) typical reinforcement learning such as Q-learning with almost no *a priori* knowledge, but given the state and action spaces, (5) action selection from the state and action space construction [Asada *et al.*, 1996a; Takahashi *et al.*, 1996], and (6) tabula rasa learning. These approaches should be evaluated in various kinds of viewpoints.

4.3 Evaluation

In order to evaluate the achieved skills, we set up the following standard tasks with some variations.

Level-I: With no obstacle

Field Definition: Positions of a robot, a ball, and target zones are defined as shown in figure 1. Exact coordinates for each position will be defined and announced. Positions denoted by alphabetical characters A to F are robot and ball positions, and X, Y, and Z are target zone.

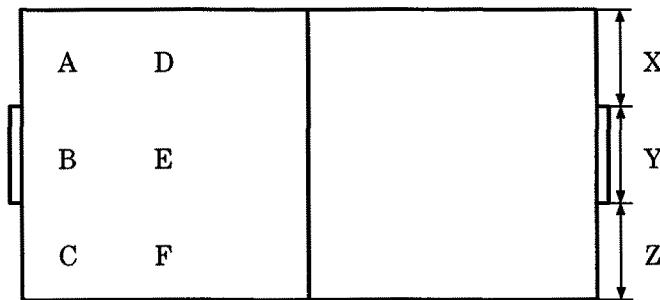


Fig. 1. Field Definition for Level-I

Robot Direction: The direction of the robot is denoted as shown in figure 2. The direction of the robot is defined as the direction that the robot move at normal forward moving mode. For robots with omni-directional driving mechanism, the direction can be the direction of default on-board camera direction. When omni-vision system is used for such a robot, arbitrary direction will be used.

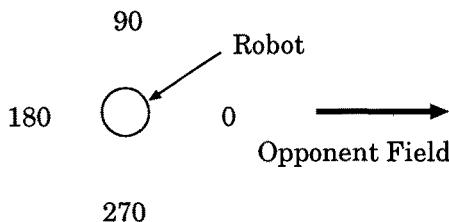


Fig. 2. Robot Direction

Session Sequence: Each session consists of 10 trials. For each trial do followings:

1. One Robot is randomly placed one of A - F position with random direction.
2. Ball is randomly placed one of A - F position, where robot is not already placed.
3. Target Zone is randomly selected from T1, T2, T3.
4. With the start signal, the robot is supposed to approach the ball and move it to the assigned target zone or hit the wall in the opponent side.

There are several restrictions in the robot operations and sequence set up. Current restrictions are as follows:

1. Instructions given to the robot before the each trial shall be limited to the assigned target zone.
2. At least three trial should include robot position in D, E, F, while a ball is in A, B, C.
3. At least one trial should be robot at position E and a ball at B.

The first restriction is defined to avoid an operator to instruct robot's motion path manually for each trial. The robot is supposed to find appropriate path by itself.

The second and third restrictions are imposed to examine robot's path planning capability. For these set up, the robot need to move around the ball to place itself to the position behind the ball from the opponent field. A simple reactive robot which rush to the ball would not able to cope with such situations.

Score: Scores will be rated as follows:

- 10 points for correctly moving ball to defined target.
- 5 points for moving a ball to adjacent zone.
- 2 points for moving a ball to opponent base line.
- No point when time out after defined time.
- Total time is counted.
- Total point / 100 = Accuracy

For example, when a robot position, a ball position, and a target zone are assigned D, B, and X, respectively, the robot must move toward it's own goal to place itself between the ball and the goal, but, perhaps, slightly to the right. The robot shall hit the ball to roll it toward zone X. If the ball hit the wall in zone X, it will get 10 points. If it hits wall in zone Y (in reality, this is a goal area), it will get 5 point.

Level-II: with Stationary Obstacle

Field: The field is defined as shown in Figure 3. Three obstacle positions are added.

Session Sequence: Session procedure is similar to Level-I, but randomly place a obstacle in either one of O1, O2, O3. The size of the obstacle is equal to the maximum size of one robot is the corresponding league, and use the color marking for the opponent robot.

1. One Robot is randomly placed one of A - F position with random direction.
2. A ball is randomly placed one of A - F position, where robot is not already placed.

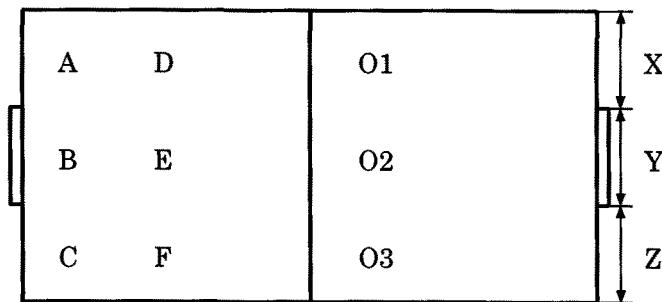


Fig. 3. Field with Obstacle

3. An obstacle is randomly placed one of O1, O2, or O3 position.
4. Target Zone is randomly selected from T1, T2, T3.
5. With the start signal, the robot is supposed to approach the ball and move it to the assigned target zone or hit the wall in the opponent side.

1. Instructions given to the robot before the each trial shall be limited to the assigned target zone.
2. At least three trial should include robot position in D, E, F, while a ball is in A, B, C.
3. At least one trial should be robot at position E and a ball at B.
4. At least one trial should be robot at position B, a ball at E, and a obstacle at O2.

Score: Scores will be rated as follows:

- 10 points for correctly moving ball to defined target.
- 5 points for moving a ball to adjacent zone.
- 2 points for moving a ball to opponent base line.
- No point when time out after defined time.
- -5 points when a ball hit the obstacle.
- Total time is counted.
- Total point / 100 = Accuracy

Level-III: With Moving Obstacle

Field: The field definition is same as Level-II, shown as Figure 3, but obstacle is moving with predefined programs.

Session Sequence: In order to predict the motion of the obstacle, which is the virtual opponents, a limited number of practice sequence is allowed.

Practice: Pre-defined number of practice sequence is allowed. Within the define numbers of trial, an operator is allowed to train the robot with initial robot, ball, and obstacle positions of their choice.

- Evaluation:**
1. One Robot is randomly placed one of A - F position with random direction.
 2. A ball is randomly placed one of A - F position, where robot is not already placed.
 3. An obstacle is randomly placed one of O1, O2, or O3 position.
 4. Target Zone is randomly selected from T1, T2, T3.
 5. With the start signal, the robot is supposed to approach the ball and move it to the assigned target zone or hit the wall in the opponent side.

Score: Scoring scheme is same as Level-II.

5 Task-II: The Ball Catching

5.1 Objectives

The objective of this challenge is to check how the most fundamental skill of catching a ball under several conditions such as pass receiving (**Situation A**), goal keeping (**Situation B**), or intercepting (**Situation C**) can be acquired, and to evaluate merits and demerit of realized skills using the standard tasks.

5.2 Technical Issues

In addition to issues in the task-I, several other issues remain:

- **Situation A:** Prediction of the ball speed and direction is a key issue to receive the ball. To receive the passed ball while moving, the relationship between the moving ball and the self motion should be made clear [Stone and Veloso, 1997].
- **Situation B:** In addition to the above issue, goal protection is important. To estimate the goal position, the agent may have to watch the goal area lines and the penalty area line. Again, the omni directional lens is much better to see the coming ball and the goal position simultaneously. In the goal area line, the agent can receive and keep the ball while outside this line it may have to kick the ball (not receiving but just protecting the goal). Discrimination of these lines might cause the vision to be much more complicated.
- **Situation C:** It seems similar to **Situation A**, but the main difference is to get the ball from the pass between opponents. This requires more accurate prediction of motions of not only the ball but also opponents (both passer and receiver). Also, an additional load in perception module.

5.3 Evaluation for Situation A

Level-I: With No obstacle

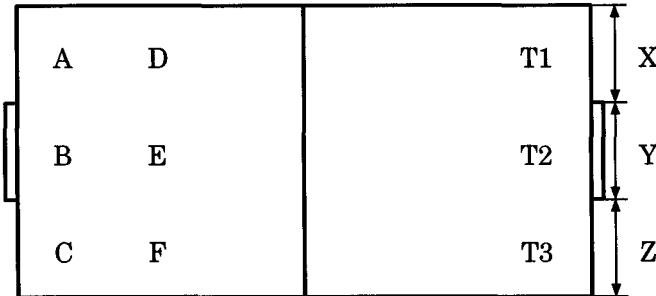


Fig. 4. Field with Obstacle

Field: Field is defined as Figure 4. T1, T2, and T3 are initial points of ball trajectory.

Session Sequence:

- Robot is randomly placed in one of A - F.
- Facing direction of the opponent goal.
- A ball is rolled from one of randomly selected position of T1, T2, T3.
- To maintain consistency of the speed, a ball is rolled from special equipment (a slope), with two different heights (change speed).
- Robot is supposed to hit the ball before the ball stops.

Score:

- 10 points: Stopped the ball before the ball touch the wall.
- 5 points: Contact the ball before the ball touch the wall, but ball hit the wall after the contact. If ball stops before hitting the wall, it will be no point even if the robot contact the ball.
- 3 Points: Contact the ball after the ball hit the wall.
- No point: Did not contact the ball until the ball stops.
- -3 point: Did not contact the ball, but it entered its own goal.
- -5 point: Contact the ball and it entered its own goal.

Level-II: With A Stationary Obstacle

Field: O1, O2, and O3 are possible positions of a stationary obstacle.

Session Sequence:

1. Robot is randomly placed in one of A - F.
2. Facing direction of the opponent goal.
3. A ball is rolled from one of randomly selected position of T1, T2, T3.
4. To maintain consistency of the speed, a ball is rolled from special equipment (a slope), with two different heights (change speed).

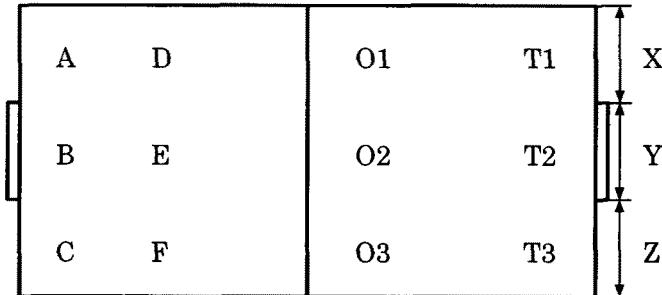


Fig. 5. Field with Obstacle

5. Robot is supposed to hit the ball before the ball stops.
6. Stationary obstacle is placed on O1, O2, or O3 randomly.

Score: Same as Level-I.

Level-III: With A Moving Obstacle

Field: Same as Level-II.

Session Sequence:

1. Robot is randomly placed in one of A - F.
2. Facing direction of the opponent goal.
3. A ball is rolled from one of randomly selected position of T1, T2, T3.
4. To maintain consistency of the speed, a ball is rolled from special equipment (a slope), with two different heights (change speed).
5. Robot is supposed to hit the ball before the ball stops.
6. Obstacle is moving with predefined programs

Score: Same as Level-II.

5.4 Situation B

Field: Set up conditions and evaluation sequences are similar to other sessions discussed already. But, robot and obstacle positions are concentrated near the goal area. From G1 to G5 are possible goal keeper robot positions (figure 6 and 7), and from O1 to O5 are possible obstacle positions (figure 7).

Session Sequence: A session sequence is as follows:

1. Robot is randomly placed in one of G1 - G5.
2. In case of Level-II, -III where obstacle exists, an obstacle is place randomly at any of position from O1 to O5.

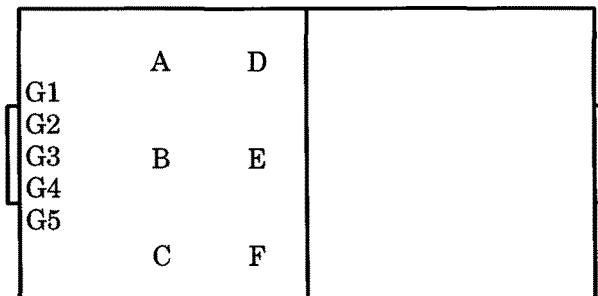


Fig. 6. Field Definition for Goal Keeping Situation

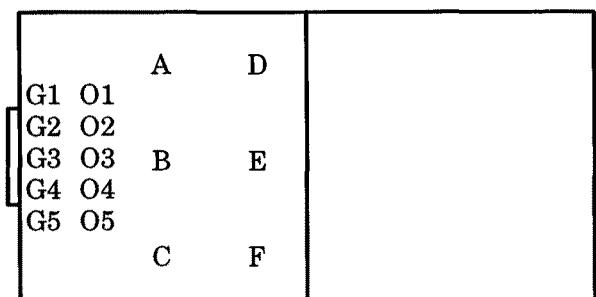


Fig. 7. Field Definition for Goal Keeping with Obstacle

3. A ball is rolled from one of randomly selected position of A - F.
4. To maintain consistency of the speed, a ball is rolled from special equipment (a slope), with two different heights (change speed).
5. Robot is supposed to defend the goal.

Score: Following score scheme is used.

- 10 points: Received the ball and move it quickly to the opponent area (beyond the center line).
- 5 points: Received the ball and move it quickly to the area beyond the obstacle positions.
- 3 Points: Received the ball and clear the ball to the side wall.
- 0 point: Received the ball, but it contacted the obstacle and move to elsewhere (Not entered the goal).
- -3 points: Contact the ball, but the ball entered the goal.
- -5 points: Did not contact the ball and the ball entered the goal.

5.5 Situation C

Evaluation sequence of the intercept requires opponent robots which actually moves by pre-programmed sequence. Definition of specific sequence largely de-

pends on the speed and accuracy of prediction and recognition of robot used in the community. A concrete evaluation sequence will be defined in future.

6 Task-III: The Cooperative Behavior (Passing the Ball)

6.1 Objectives

The objective of this challenge is to check how the most fundamental cooperative skill (passing a ball between two players) can be acquired, and to evaluate merits and demerit of realized skills using the standard tasks.

This task focuses on a basic skill of cooperative behavior between two agents while task-I and -II focus on the basic skills of one single agent even though the environment includes other agents (possible opponents). If task-I and -II are successfully achieved, passing the ball between two players might become easy. That would mean a combination of passing and receiving skills. However, from a viewpoint of cooperative behaviors, there might be more issues.

6.2 Technical Issues

In addition to issue in the task-I and -II, three issues are left.

- Since the control architecture is not centralized but decentralized, each agent should know capabilities in passing and receiving skills of not only itself but the other. That is, the agent should estimate the level of the other agent skills. This means agent modeling. The problem includes partial observability due to perceptual limitations.
- Even though both agents have reasonable passing and receiving skills, the timing of passing and receiving must be coordinated. If both agents try to learn to improve their own behavior independently, the learning may not converge because the policy of the other changes simultaneously. To prevent this situation, one of the agents should be a coach (fixed policy) and the other, a learner. In this case, modeling of the learner is another issue for good teaching.
- In all these behaviors, and especially in a pass interception, the success rate will drastically increase if the agent can predict the ball holder's ball controlling behavior, specifically, when and in which direction it will kick. For this, the agent should first find a ball holder, track its motion, and predict the oncoming event based on the relative positions of the ball and the surrounding agents, such as the potential pass receivers. A primitive example of vision based interception of a static obstacle from another robot's trajectory has been demonstrated [Kuniyoshi, 95]. However, a general interception in fully dynamic situations like soccer playing is an open problem.
- Selection of passing direction depends on the motions of opponents. This introduces the opponent modeling issue which makes the cooperative behavior much harder to realize.

6.3 Evaluation

Since the challenge with many issues is very hard in the current stage, the Phase I challenge will only check cooperative behavior in a benign environment: two players with equal skills and with no opponents.

Field: Field is defined as Figure 8. From D1 to D6 are possible defender robot positions, from O1 to O6 are possible opponents positions, and from F1 to F6 are possible forward robot positions. A ball is placed on one of possible defender positions.

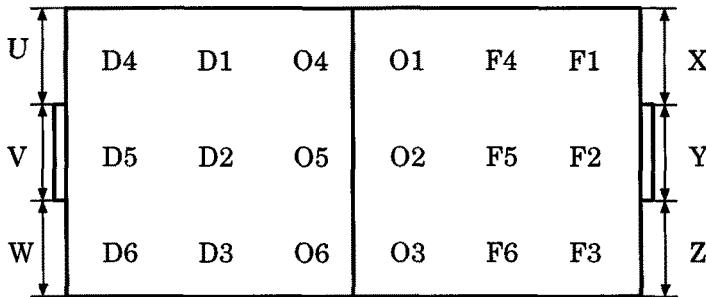


Fig. 8. Field Definition for Ball Passing

Session Sequence

1. One robot is randomly placed in one of D1 - D6.
2. The other robot is randomly place in one of F1 - F6.
3. Direction of two robots are defined at random.
4. A ball is randomly placed in one of D1 - D6, where robot is not already occupied.
5. For Level-II and -III sequence, an obstacle is randomly placed in one of O1 - O6.
6. A robot is supposed to hit the ball and the other robot shall receive it.

Score:

- 10 points: Move the ball and pass it to the other robot without hitting the obstacle nor the wall. The robot in F1 - F6 area must stop the ball or goal it after contacting the ball.
- 5 points: Move the ball and pass it to the other robot without hitting the obstacle. The ball may hit the wall during the pass. The robot in F1 - F6 area must stop the ball or goal it after contacting the ball.
- 3 points: Move the ball and pass it to the other robot without hitting the obstacle nor the wall. The robot in F1 - F6 area contacted the ball, but ball hit the side wall.

- No point: Move the ball beyond the center line without hitting the obstacle, but the other robot did not stop the ball nor goal the ball.
- -3 point: The ball hit the obstacle.
- -5 point: Fail to move the ball beyond the centerline within given time.
- -10 points: Fail to contact the ball within given time.
- -10 points: Own Goal.

7 Discussion

7.1 Managing the Challenge

Committee: The RoboCup Physical Agent Challenge Committee will be formed to execute the challenge initiative. The committee will include members of the international executive committee for RoboCup and distinguished robotics researchers not directly involved in the RoboCup. The committee will create specific tasks and criteria for evaluation, as well as provide technical advises for the challengers.

Web Site: On the RoboCup home page (<http://www.robocup.org/RoboCup/>), we are planning to show how to design a basic robot platform and technical information. The home page also provides a set of papers and technical documents related to RoboCup.

Competitions and Conferences: A series of RoboCup competition is planned to provide opportunities to test their ideas in conjunction with workshops at major international conferences, as well as local workshops, in order to facilitate exchange of information, discussions, and to feedback the status of the challengers to the overall framework of the challenge. As international events, we are planning to have RoboCup-98 Paris (with ICMAS-98 conference), RoboCup-98 Victoria (as a part of IROS-98 conference), and RoboCup-98 Singapore (as a part of PRICAI-98 Conference). Several local competitions will be organized by local committee in each region.

7.2 Infrastructure Issues

One of the most important issues in the physical agent challenge is the design of the standard platform and their supply because the range of the physical agent designs is too wide to evaluate the skills achieved in the same level. If we had a precisely specified platform for RoboCup, the situation would be similar to the current simulation track because the design issues of mechanical structure would disappear. On the other hand, completely free design makes it hard to evaluate the skills achieved on equal terms. As a middle ground, we would like to have a platform that allows us to easily reconfigure the physical structure by which we expect physical agents to develop complex behaviors.

Recently, an open architecture called OPENR has been proposed as such a platform [Fujita and Kageyama, 1997]. OPENR has 1) standard interfaces

between physical and software components and a programming framework, 2) configurable physical components with a common interface and information exchanger of their function and configurations, and 3) is constructed as a layered architecture based on object-oriented robot OS.

Expecting that an open architecture such as OPENR will be available for the RoboCup context in the very near future, we will offer basic resources and opportunities in order to facilitate technical progress based on the RoboCup physical agent challenge.

8 Conclusion

The RoboCup Physical Agent Challenge Phase I offers a set of three fundamental challenges, focused on ball moving, ball catching, and ball passing. These were chosen as the first set of challenges because they are essential technical issues for RoboCup, as well as for general robot system control and in particular for multi-agent control.

References

- [Asada *et al.*, 1996a] M. Asada, S. Noda, and K. Hosoda. Action-based sensor space categorization for robot learning. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS '96)*, pages 1502–1509, 1996.
- [Asada *et al.*, 1996b] M. Asada, S. Noda, S. Tawaratumida, and K. Hosoda. Purposeful behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.
- [Asada, 1996] Minoru Asada. An agent and an environment: A view on “having bodies” - a case study on behavior learning for vision-based mobile robot -. In *Proceedings of 1996 IROS Workshop on Towards Real Autonomy*, pages 19–24, 1996.
- [Asama *et al.*, 1996] H. Asama, M. Sato, N. Goto, H. Kaetsu, A. Matsumoto, and I. Endo. Mutual transportation of cooperative mobile robots using forklift mechanisms. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 1754–1759, 1996.
- [Brooks, 1991] R. A. Brooks. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*, pages 3–15. MIT/Elsevier, 1991.
- [Inoue, *et al.*, 92] H. Inoue and T. Tachikawa and M. Inaba. Robot vision system with a correlation chip for real-time tracking, optical flow and depth map generation. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 1621–1626, 1992.
- [Ishiguro, 96] H. Ishiguro. <http://www.lab7.kuis.kyoto-u.ac.jp/vision/omni-sensor/omni-sensor.htm>.
- [Kitano, *et al.*, 95] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. and Osawa, E., "RoboCup: The Robot World Cup Initiative", *IJCAI-95 Workshop on Entertainment and AI/Aife*, 1995
- [Kitano, *et al.*, 97] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., and Matsubara, H., "RoboCup: A Challenge AI Problem", *AI Magazine*, Spring, 1997.
- [Kuniyoshi, 95] Y. Kuniyoshi. Behavior Matching by Observation for Multi-Robot Cooperation. In G. Giralt and G. Hirzinger (eds.) *Robotics Research - The Seventh International Symposium*, pages 343–352, Springer, 1996.

- [Nakamura and Asada, 1995] T. Nakamura and M. Asada. Motion sketch: Acquisition of visual motion guided behaviors. In *Proc. of International Joint Conference on Artificial Intelligence*, pages 126–132, 1995.
- [Nakamura and Asada, 1996] T. Nakamura and M. Asada. Stereo sketch: Stereo vision-based target reaching behavior acquisition with occlusion detection and avoidance. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 1314–1319, 1996.
- [Nakamura *et al.*, 1996] T. Nakamura, J. Morimoto, and M. Asada. Direct coupling of multisensor information and actions for mobile robot behavior acquisition. In *Proc. of 1996 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration*, pages 139–144, 1996.
- [Stone and Veloso, 1997] P. Stone, M. Veloso. Towards Collaborative and adversarial learning: A case study in robotic soccer . In *International Journal of Human-Computer Systems (IJHCS)*, 1997.
- [Stone and Veloso, 1997] P. Stone, M. Veloso. A Layered Approach to Learning Client Behaviors in the RoboCup Soccer Server In *Applied Artificial Intelligence (AAI) Journal*, 1997.
- [Takahashi *et al.*, 1996] Yasutake Takahashi, Minoru Asada, and Koh Hosoda. 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, 1996.
- [Uchibe *et al.*, 1996] Eiji Uchibe, Minoru Asada, and Koh Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS '96)*, pages 1329–1336, 1996.
- [Fujita and Kageyama, 1997] Masahiro Fujita and Koji Kageyama. An Open Architecture for Robot Entertainment In *Proc. of First International Conference on Autonomous Agents*, pages 435–442, 1997.

The RoboCup Synthetic Agent Challenge 97

Hiroaki Kitano¹, Milind Tambe², Peter Stone³, Manuela Veloso³,
Silvia Coradeschi⁴, Eiichi Osawa¹, Hitoshi Matsubara⁵,
Itsuki Noda⁵, and Minoru Asada⁶

¹ Sony Computer Science Laboratory,

3-14-13 Higashi-Gotanda, Shinagawa, Tokyo 141 Japan

² Information Science Institute/University of Southern California,
Marina del Ray, California, USA

³ Shool of Computer Science, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, Pennsylvania, 15213 USA

⁴ Linkoeping University, Linkoeping, Sweden

⁵ ElectroTechnical Laboratory, 1-1-4 Umezono, Ibaraki, 305 Japan

⁶ Department of Adaptive Machine Systems, Osaka University,
Suita, Osaka, 565 Japan

Abstract. RoboCup Challenge offers a set of challenges for intelligent agent researchers using a friendly competition in a dynamic, real-time, multi-agent domain. While RoboCup in general envisions longer range challenges over the next few decades, RoboCup Challenge presents three specific challenges for the next two years: (i) learning of individual agents and teams; (ii) multi-agent team planning and plan-execution in service of teamwork; and (iii) opponent modeling. RoboCup Challenge provides a novel opportunity for machine learning, planning, and multi-agent researchers — it not only supplies a concrete domain to evaluate their techniques, but also challenges researchers to evolve these techniques to face key constraints fundamental to this domain: real-time, uncertainty, and teamwork.

1 Introduction

RoboCup (The World Cup Robot Soccer) is an attempt to promote AI and robotics research by providing a common task, Soccer, for evaluation of various theories, algorithms, and agent architectures [Kitano, et al., 1995; Kitano et al., 1997a; Kitano et al., 1997b]. Defining a standard problem in which various approaches can be compared and progress can be measured provides fertile grounds for engineering research. Computer chess has been a symbolic example of the standard challenge problems. A salient feature of computer chess is that progress can be measured via actual games against human players.

For an agent (a physical robot or a synthetic agent) to play soccer reasonably well, a wide range of technologies need to be integrated and a number of technical breakthroughs must be made. The range of technologies spans both AI and robotics research, such as design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning and planning, intelligent

robotics, sensor-fusion, and so forth. RoboCup consists of three competition tracks:

Real Robot League: Using physical robots to play soccer games.

Software Agent League: Using software or synthetic agents to play soccer games on an official soccer server over the network.

Expert Skill Competition: Competition of robots which have special skills, but are not able to play a game.

RoboCup offers a software platform that forms the basis of the software or synthetic agent league. The goal is to enable a wider range of research in synthetic (or “virtual reality”) environments, that are today proving to be critical in training, entertainment, and education[Tambe et al., 1995]. The software agent league also promotes research on network-based multi-agent interactions, computer graphics, and physically realistic animation — a set of technologies which potentially promotes advanced use of internet.

2 Technical Challenges in RoboCup

RoboCup offers significant long term challenges, which will take a few decades to meet. However, due to the clarity of the final target, several subgoals can be derived, which define mid term and short term challenges. One of the major reasons why RoboCup is attractive to so many researchers is that it requires the integration of a broad range of technologies into a team of complete agents, as opposed to a task-specific functional module. The long term research issues are too broad to compile as a list of specific items. Nevertheless, the challenges involve a broad range of technological issues ranging from the development of physical components, such as high performance batteries and motors, to highly intelligent real time perception and control software.

The mid term technical challenges, which are the target for the next 10 years, can be made more concrete, and a partial list of specific topics can be compiled. Following is a partial list of research areas involved in RoboCup, mainly targeted for the mid term time span: (1) agent architecture in general, (2) combining reactive approach and modeling/planning approach, (3) real-time recognition, planning, and reasoning, (4) reasoning and action in dynamics environment, (5) sensor fusion, (6) multi-agent systems in general, (7) behavior learning for complex tasks, (8) strategy acquisition, and (9) cognitive modeling in general.

In addition to these technologies, providing a network-based soccer server with high quality 3D graphics capabilities requires advancement of technologies for the real time animation of simulated soccer players and network-based interactive multi-user server system. These are key technologies for network-based services in the coming years.

The RoboCup Challenge shall be understood in the context of larger and longer range challenges, rather than as a one-shot challenge. Thus, we wish to provide a series of short term challenges, which naturally leads to the accomplishment of the mid term and long term challenges. RoboCup challenge is organized

into three major classes; (1) Synthetic Agent Challenge, (2) Physical Agent Challenge, and (3) Infrastructure Challenge. The RoboCup Synthetic Agent Challenge deal with technologies which can be developed using software simulator, which is described in this paper. The RoboCup Physical Agent Challenge intends to promote research using real robot, and thus requires longer-time frame for each challenge to be accomplished. Details of this challenge is described in [Asada et al., 1997], and carried out together with the RoboCup Synthetic Agent Challenge but in more moderate timeframe. The Infrastructure Challenge will be presented to facilitate research to establish infrastructure aspect of RoboCup, AI, and robotics in general. Such challenge includes education programs, common robot platforms and components standard, automatic commentary systems and intelligent studio systems for RoboCup games.

3 Overview of The RoboCup Synthetic Agent Challenge

For the RoboCup Synthetic Agent Challenge 97, we offer three specific targets, critical not only for RoboCup but also for general AI research. These challenges will specifically deal with the software agent league, rather than the real robot league.

The fundamental issue for researchers who wish to build a team for RoboCup is to design a multiagent system that behaves in real-time, performing reasonable goal-directed behaviors. Goals and situations change dynamically and in real-time. Because the state-space of the soccer game is prohibitively large for anyone to hand-code all possible situations and agent behaviors, it is essential that agents learn to play the game strategically. The research issues in this aspect of the challenge involve:

(1) machine learning in a multiagent, collaborative and adversarial environment, (2) multiagent architectures, enabling real-time multiagent planning and plan execution in service of teamwork, and (3) opponent modelling.

Therefore, we propose the following three challenges as areas of concentration for the RoboCup Synthetic Agent Challenge 97:

- Learning challenge
- Teamwork challenge
- Opponent modeling challenge

Evaluating how well competing teams meet these challenges in RoboCup is clearly difficult. If the task is to provide the fastest optimization algorithm for a certain problem, or to prove a certain theorem, the criteria are evident. However, in RoboCup, while there may be a simple test set to examine basic skills, it is not generally possible to evaluate the goodness of a team until it actually plays a game. Therefore, a standard, highly skilled team of opponents is useful to set an absolute basis for such evaluation. We hope to use hand-coded teams, possibly with highly domain-specific coordination, to provide such a team of opponents. Indeed, in a series of preliminary competitions such as

PreRoboCup-96 held at the IROS-96 conference, and several other local competitions, teams with well-designed hand-coded behaviors, but without learning and planning capabilities, have performed better than teams with learning and planning schemes. Of course, these hand-coded teams enjoyed the advantage of very low game complexities in initial stages of RoboCup — increasingly complex team behaviors, tactics and strategies will necessitate agents to face up to the challenges of learning, teamwork and opponent modeling.

Therefore, responses to this challenge will be evaluated based on (1) their performance against some standard hand-coded teams as well as other teams submitted as part of the competition; (2) behaviors where task specific constraints are imposed, such as probabilistic occurrence of unexpected events, (3) a set of task specific sequences, and (4) novelty and technical soundness of the approach.

4 The RoboCup Learning Challenge

4.1 Objectives

The objectives of the RoboCup Learning Challenge is to solicit comprehensive learning schemes applicable to the learning of multiagent systems which need to adapt to the situation, and to evaluate the merits and demerits of proposed approaches using standard tasks.

Learning is an essential aspect of intelligent systems. In the RoboCup learning challenge, the task is to create a learning and training method for a group of agents. The learning opportunities in this domain can be broken down into several types:

1. Off-line skill learning by individual agents;
2. Off-line collaborative learning by teams of agents;
3. On-line skill and collaborative learning;
4. On-line adversarial learning.

The distinction between off-line and on-line learning is particularly important in this domain since games last for only 20 minutes. Thus on-line techniques, particularly if they are to learn concepts that are specific to an individual game, must generalize very quickly. For example, if a team is to learn to alter its behavior against an individual opponent, the team had better be able to improve its performance before the game is over and a new opponent appears. Such distinctions in learning can be applied to a broad range of multi-agent systems which involve learning capabilities.

4.2 Technical Issues

Technical issues anticipated in meeting this challenge are the development of novel learning schemes which can effectively train individual agents and teams of agents in both off-line and on-line methods. One example of possible learning scheme for meeting this challenge is as follows:

Off-line skill learning by individual agents: learning to intercept the ball or learning to kick the ball with the appropriate power when passing.

Since such skills are challenging to hand-code, learning can be useful during a skill development phase. However, since the skills are invariant from game to game, there is no need to relearn them at the beginning of each new game [Stone and Veloso, 1997].

Off-line collaborative learning by teams of agents: learning to pass and receive the ball.

This type of skill is qualitatively different from the individual skills in that the behaviors of multiple agents must be coordinated. A "good" pass is only good if it is appropriate for the receivers receiving action, and vice versa. For example, if the passer passes the ball to the receiver's left, then the receiver must at the same time move to the left in order to successfully complete a pass. As above, such coordination can carry over from game to game, thus allowing off-line learning techniques to be used [Stone and Veloso, 1997].

On-line skill and collaborative learning: learning to play positions.

Although off-line learning methods can be useful in the above cases, there may also be advantages to learning incrementally as well. For example, particular aspects of an opposing teams' behavior may render a fixed passing or shooting behavior ineffective. In that case, the ability to adaptively change collaborative or individual behaviors during the course of a game, could contribute to a team's success.

At a higher level, team issues such as role (position) playing on the field might be best handled with adaptive techniques. Against one opponent it might be best to use 3 defenders and 8 forwards; whereas another opponent might warrant a different configuration of players on the field. The best teams should have the ability to change configurations in response to events that occur during the course of a game.

On-line adversarial learning: learning to react to predicted opponent actions.

If a player can identify patterns in the opponents' behaviors, it should be able to proactively counteract them. For example, if the opponent's player number 4 always passes to its teammate number 6, then player 6 should always be guarded when player 4 gets the ball.

4.3 Evaluation

For challenge responses that address the machine learning issue (particularly the on-line learning issue), evaluation should be both against the publicly available teams and against at least one previously unseen team.

First, teams will play games against other teams and publicly available teams under normal circumstances. This evaluates the team's general performance. This involves both AI-based and non-AI based teams.

Next, teams will play a set of defined benchmarks. For example, after fixing their programs, challengers must play a part of the game, starting from the defined player positions, with the movement of the opponents pre-defined, but not disclosed to the challengers. After several sequences of the game, the performance will be evaluated to see if it was able to improve with experience. The movement of the opponents are not coded using absolute coordinate positions, but as a set of algorithms which generates motion sequences. The opponent algorithms will be provided by the organizers of the challenge by withholding at least one successful team from being publicly accessible.

Other benchmarks which will clearly evaluate learning performance will be announced after discussion with challenge participants.

5 The RoboCup Teamwork Challenge

5.1 Objectives

The RoboCup Teamwork Challenge addresses issues of real-time planning, re-plannig, and execution of multi-agent teamwork in a dynamic adversarial environment. Major issues of interest in this specific challenge for the 97-99 period are architectures for real-time planning and plan execution in a team context (essential for teamwork in RoboCup). In addition, generality of the architecture for non-RoboCup applications will be an important factor.

Teamwork in complex, dynamic multi-agent domains such as Soccer mandates highly flexible coordination and communication to surmount the uncertainties, e.g., dynamic changes in team's goals, team members' unexpected inability to fulfil responsibilities, or unexpected discovery of opportunities. Unfortunately, implemented multi-agent systems often rely on preplanned, domain-specific coordination that fails to provide such flexibility. First, it is difficult to anticipate and preplan for all possible coordination failures; particularly in scaling up to complex situations. Thus, it is not robust enough for dynamic tasks, such as soccer games. Second, given domain specificity, reusability suffers. Furthermore, planning coordination on the fly is difficult, particularly, in domains with so many possible actions and such large state spaces. Indeed, typical planners need significantly longer to find even a single valid plan. The dynamics of the domain caused by the unpredictable opponent actions make the situation considerably more difficult.

A fundamental reason for these teamwork limitations is the current agent architectures. Architectures such as Soar [Newell, 1990], RAP [Firby, 1987], IRMA [Pollack, 1991], and BB1 [Hayes-Roth et al., 1995] facilitate an individual agent's flexible behaviors via mechanisms such as commitments and reactive plans. However, teamwork is more than a simple union of such flexible individual behaviors, even if coordinated. A now well-known example (originally from [Cohen and Levesque, 1991]) is ordinary traffic, which even though simultaneous and coordinated by traffic signs, is not teamwork. Indeed, theories of

teamwork point to novel mental constructs as underlying teamwork, such as team goals, team plans, mutual beliefs, and joint commitments [Grosz, 1996; Cohen and Levesque, 1991], lacking in current agent architectures. In particular, team goals, team plans or mutual beliefs are not explicitly represented; furthermore, concepts of team commitments are absent. Thus, agents cannot explicitly represent and reason about their team goals and plans; nor flexibly communicate/coordinate when unanticipated events occur. For instance, an agent cannot itself reason about its coordination responsibilities when it privately realizes that the team's current plan is unachievable — e.g., that in the best interest of the team, it should inform its teammates. Instead, agents must rely on domain-specific coordination plans that address such contingencies on a case-by-case basis.

The basic architectural issue in the teamwork challenge is then to construct architectures that can support planning of team activities, and more importantly execution of generated team plans. Such planning and plan execution may be accomplished via a two tiered architecture, but the entire system must operate in real-time. In RoboCup Soccer Server, sensing will be done in every 300 to 500 milli-seconds, and action command can be dispatched every 100 millisecond. Situation changes at milli-second order, thus planning, re-planning, and execution of plans must be done in real-time.

5.2 Technical Issues

We present a key set of issues that arise assuming our particular two tiered planning and plan-execution approach to teamwork. Of course, those who approach the problem from different perspective may have different issues, and the issues may change depending on the type of architecture employed.

The following is the envisioned teamwork challenge in this domain: (i) a team deliberatively accumulates a series of plans to apply to games with different adversarial teams; (ii) game plans are defined at an abstract level that needs to be refined for real execution; (iii) real-time execution in a team-plan execution framework/architecture that is capable of addressing key contingencies. Such an architecture also alleviates the planning concerns by providing some “common-sense” teamwork behaviors — not all of the coordination actions are required to be planned in detail as a result. The key research tasks here are:

Contingency planning for multiagent adversarial game playing: Before a game starts, one would expect the team to generate a strategic plan for the game that includes contingency plan segments that are to be recognized and eventually slightly adapted in real-time. Two main challenges can be identified in this task:

- Definition of strategic task actions with probabilistic applicability conditions and effects. Uncertainty in the action specification is directly related to the identification of possible probabilistic disruptive or favorable external events.
- Definition of objectives to achieve. In this domain, the goal of winning and scoring should be decomposed in a variety of more concrete goals that serve the ultimate final scoring goal. Examples are actions and goals to achieve specific attacking or defending positioning.

Plan decomposition and merge: A correspondence between team actions and goals and individual actions and goals must be set. The team plan decomposition may create individual goals that are not necessarily known to all the team players. Furthermore, within the contingency team plan, it is expected that there may be a variety of adversary-independent and adversary-dependent goals. The decomposition, coordination, and appropriate merge of individual plans to the service of the main team plan remain open challenging research tasks. RoboCup provides an excellent framework to study these issues.

Executing Team Plans: Team plan execution during the game is the determining factor in the performance of the team. It addresses the coordination contingencies that arise during the execution, without the need for detailed, domain-specific coordination plans. Execution also monitors the contingency conditions that are part of the global contingency team plan. Selection of the appropriate course of action is driven by the state information gathered by execution.

5.3 Evaluations

The Teamwork Challenge scenario described above has been idealized by several AI researchers, at least in the planning and multiagent communities. RoboCup, both in its simulated and real leagues, provides a synergistic framework to develop and/or test dynamic planning multiagent algorithms.

Specifically, we are planning to evaluate the architecture and teams in the following evaluation scheme:

Basic Performance: The team must be able to play reasonably well against both the best hand-coded teams, which has no planning, and against other planning-based systems. Relative performance of the team can be measured by actually playing a series of games against other unknown teams. Thus, basic performance will be measured by:

- Performance against hand-coded teams.
- Performance against other teams.

Robustness: The robustness in teamwork means that the team, as a whole, can continue to carry out the mission even if unexpected changes, such as accidental removal of the players in the team, sudden change of team composition, or changes in operation environment. For example, if one of players in the team was disabled, the team should be able to cope with such accidents, by taking over the role of disabled players, or reformulating their team strategy. Thus, this evalution represents a set of unexpected incidents during the game, such as:

- Some players will be disabled, or their capability will be significantly undermined by these accidents. Also, some disabled players may be enabled later in the game.
- Opponent switch their strategy, and the team must cope with their new strategy in real time.

- Some of opponent's players will be disabled, or their performance will be significantly undermined. These disabled players may come back to the game later.
- Teammate changes during the game.
- Weather factor changes.

The RoboCup Teamwork Challenge therefore is to define a general set of teamwork capabilities to be integrated with agent architectures to facilitate flexible, reusable teamwork. The following then establish the general evaluation criteria:

General Performance: General performance of the team, thus the underlying algorithms, can be measured by a series of games against various teams. This can be divided into two classes (1) normal competitions where no accidental factors involved, and (2) contingency evaluation where accidental factors are introduced.

Real-Time Operations: The real-time execution, monitoring, and replanning of the contingency plan is an important factor of the evaluation. For any team to be successful in the RoboCup server, it must be able to react in real time: sensory information arrives between 2 and 8 times a second and agents can act up to 10 times a second.

Generality: Reuse of architecture in other applications: Illustrate the reuse of teamwork capabilities in other applications, including applications for information integration on the internet, entertainment, training, etc.

Conformity with Learning: Finally, given the premises above and the complexity of the issues, we argue and challenge that a real-time multiagent planning system needs to have the ability to be well integrated with a learning approach, i.e., it needs to refine and dynamically adapt and refine its complete behavior (individual and team) based on its past experience.

Other issues such as reuse of teamwork architecture within the RoboCup community, and planning for team players that are not yet active in order to increase their probability of being useful in future moves, such as role playing and positioning of the team players that *do not* have the ball, will be considered, too.

6 RoboCup Opponent Modeling Challenge

Agent modeling – modeling and reasoning about other agent's goals, plans, knowledge, capabilities, or emotions — is a key issue in multi-agent interaction. The RoboCup opponent modeling challenge calls for research on modeling a team of opponents in a dynamic, multi-agent domain. The modeling issues in RoboCup can be broken down into three parts:

On-line tracking: Involves individual players' real-time, dynamic tracking of opponents' goals and intentions based on observations of actions. A player

may use such tracking to predict the opponents' play and react appropriately. Thus if a player predicts that player-5 is going to pass a ball to player-4, then it may try to cover player-4. Such on-line tracking may also be used in service of deception. The challenges here are (i) real-time tracking despite the presence of ambiguity; (ii) addressing the dynamism in the world; (iii) tracking teams rather than only individuals – this requires an understanding of concepts involved in teamwork.

On-line tracking may feed input to the on-line planner or the on-line learning algorithm.

On-line strategy recognition: "Coach" agents for teams may observe a game from the sidelines, and understand the high-level strategies employed by the opposing team. This contrasts with on-line tracking because the coach can perform a much higher-level, abstract analysis, and in the absence of real-time pressures, its analysis can be more detailed.

The coach agents may then provide input to its players to change the team formations, or play strategy.

Off-line review: "Expert" agents may observe the teams playing in an after-action review, to recognize the strengths and weaknesses of the teams, and provide an expert commentary. These experts may be trained on databases of human soccer play.

These issues pose some fundamental challenges that will significantly advance the state of the art in agent modeling. In particular, previous work has mostly focused on plan recognition in static, single-agent domains, without real-time constraints. Only recently has attention shifted to dynamic, real-time environments, and modeling of multi-agent teamwork [Tambe, 1996b].

A realistic challenge for IJCAI-99 will be to aim for on-line tracking. Optimistically, we expect some progress towards on-line strategy recognition; off-line review will likely require further research beyond IJCAI-99.

For evaluation, we propose, at least, following evaluation to be carried out to measure the progress:

Game Playing: A team of agents plays against two types of teams:

- One or two unseen RoboCup team from IJCAI-97, shielded from public view.
- The same unseen RoboCup teams from IJCAI-97 as above, but modified with some new behaviors. These teams will now deliberately try out new adventurous strategies, or new defensive strategies.

Disabled Tracking: Tracking functionality of the agents will be turned off, and compared with normal performance.

Deceptive Sequences: Fake teams will be created which generates deceptive moves. The challenger's agent must be able to recognize the opponent's deceptive moves to beat this team.

For each type of team, we will study the performance of the agent-modelers. Of particular interest is variations seen in agent-modelers behaviors given the modification in the opponents' behaviors. For each type of team, we will also

study the advise offered by the coach agent, and the reviews offered by the expert agents, and the changes in them given the changes in the opponents' behaviors.

7 Managing Challenges

In order to facilitate technical progress based on the RoboCup challenge, we offer basic resources and opportunities.

The RoboCup Challenge Committee: The RoboCup Challenge Committee will be formed to execute the challenge initiative. The committee will include members of the international executive committee for RoboCup and distinguished researchers not directly involved in RoboCup. The committee will create specific tasks and criteria for evaluation, as well as providing technical advice for the challengers.

Resources: In the RoboCup home page, basic software resources and technical information can be obtained. (<http://www.robocup.org/RoboCup>) Software includes the Soccer Server system, which is a server system for the simulation track, and various sample teams. In addition, sample test sequences will be provided. The home page also provides a set of papers and technical documents related to RoboCup.

Competitions: A series of RoboCup competitions are planned to provide opportunities to test ideas. As international events, we are planning to have RoboCup-98 in Paris (The Official Event of the World Cup), RoboCup-98 Victoria (as a part of IROS-98 conference), and RoboCup-98 Singapore (as a part of PRICAI-98 Conference). Several local competitions will be organized by local committee in each region. The final evaluation and exhibit of the results will be made at IJCAI-99.

Workshops: Workshops will be organized at major international conferences, as well as at local workshops, in order to facilitate exchange of information, to have technical discussions, and to get feedback on the status of the challengers in relation to the overall framework of the challenge.

8 Conclusion

The RoboCup Challenge-97 offers a set of three fundamental challenges, focused on learning, real-time planning, and opponent modeling. Learning and real-time planning of multi-agent systems were chosen as the first set of challenges because they are essential technical issues for RoboCup, as well as for general AI systems using a multi-agent approach. Together with the physical agent challenge, these challenges will be a basis for the RoboCup Challenge-99, and for longer research enterprises.

References

- [Asada et al., 1997] Asada, M., Kuniyoshi, M., Drogoul, A., Asama, H., Mataric, M., Duhamel, D., Stone, P., and Kitano, H., "The RoboCup Physical Agent Challenge: Phase-I," To appear in *Applied Artificial Intelligence (AAI) Journal*, 1997.
- [Cohen and Levesque, 1991] Cohen, P. R. and Levesque, H. J., "Confirmation and Joint Action", *Proceedings of International Joint Conf. on Artificial Intelligence*, 1991.
- [Firby, 1987] Firby, J., "An investigation into reactive planning in complex domains", *Proceedings of National Conf. on Artificial Intelligence*, 1987.
- [Grosz, 1996] Grosz, B., "Collaborating Systems", *AI magazine*, 17, 1996.
- [Hayes-Roth et al., 1995] Hayes-Roth, B. and Brownston, L. and Gen, R. V., "Multi-agent collaboration in directed improvisation", *Proceedings of International Conf. on Multi-Agent Systems*, 1995.
- [Jennings, 1995] Jennings, N., "Controlling cooperative problem solving in industrial multi-agent systems using joint intentions", *Artificial Intelligence*, 75, 195-240, 1995.
- [Kitano et al., 1997a] Kitano, H., Asada, M., Osawa, E., Noda, I., Kuniyoshi, Y., Matsubara, H., "RoboCup: The Robot World Cup Initiative", *Proc. of the First International Conference on Autonomous Agent (Agent-97)*, 1997.
- [Kitano et al., 1997b] Kitano, H., Asada, M., Osawa, E., Noda, I., Kuniyoshi, Y., Matsubara, H., "RoboCup: A Challenge Problem for AI", *AI Magazine*, Vol. 18, No. 1, 1997.
- [Kitano, et al., 1995] Kitano, H. and Asada, M. and Kuniyoshi, Y. and Noda, I. and Osawa, E., "RoboCup: The Robot World Cup Initiative", *IJCAI-95 Workshop on Entertainment and AI/Alife*, 1995
- [Newell, 1990] Newell, A., *Unified Theories of Cognition*, Harvard Univ. Press, Cambridge, Mass., 1990.
- [Pollack, 1991] Pollack, M., "The uses of plans", *Artificial Intelligence*, 57, 1992.
- [Stone and Veloso, 1997] Stone, P. and Veloso, M., "A layered approach to learning client behaviors in the robocup soccer server," *To appear in Applied Artificial Intelligence (AAI) Journal*, 1997.
- [Tambe, 1996a] Tambe, M., "Tracking dynamic team activity", *Proceedings of National Conf. on Artificial Intelligence*, 1996.
- [Tambe, 1996b] Tambe, M., "Teamwork in real-world, dynamic environments", *Proc. International Conf. on Multi-agent Systems*, 1996.
- [Tambe et al., 1995] Tambe, M. and Johnson, W. and Jones, R. and Koss, F. and Laird, J. and Rosenbloom, P. and Schwamb, K., "Intelligent agents for interactive simulation environments", *AI Magazine*, 16, 1995.

Playing Soccer by Modifying and Combining Primitive Reactions

Jukka Riekki & Juha Röning

Dept. of Electrical Engineering and Infotech Oulu

University of Oulu

FIN-90570 Oulu

{jpr,jjr}@ee.oulu.fi

Abstract

We describe a novel approach for basing the actions of a mobile robot on the environment. The key idea is to produce continuously primitive reactions to all the important objects in the environment. Tasks are executed by modifying and combining these reactions. We have utilized this approach in playing soccer. We describe here the control system that controlled our players in the first RoboCup competition.

1 Introduction

The subsumption control architecture created by Brooks demonstrated the importance of reactivity [Brooks, 1986; Brooks, 1990; Connell, 1990]. Quite simple robots can execute tasks in complex environments when they are able to cope with unexpected environmental events. The key to the success of these robots is direct coupling between perception and action. But these robots are capable of executing only simple tasks. The reason for this shortcoming is the lack of representations [Tsotsos, 1995].

In this paper, we will show how suitable representations foster the building of a goal-oriented and a behavior-based control system. We will describe how the representations for goals, intermediate results, and robot actions facilitate coordinating behaviors based on the given goals, satisfying several behaviors in parallel, and managing the complexity of the system.

Our control architecture is called *Samba*. Earlier versions of the architecture have been described previously [Kuniyoshi *et al.*, 1994; Riekki and Kuniyoshi, 1995; Riekki and Kuniyoshi, 1996]. The Samba architecture was inspired by the work of Brooks [1986], Chapman [1991], and Rosenblatt and Payton [1989].

In the Samba architecture, behaviors produce *action maps*. An action map specifies for each possible action how preferable the action is from the perspective of the producer of the map. The preferences are described by associating a weight to each set of action parameters. Rosenblatt has also proposed an architecture in which behaviors produce weights (votes) for different actions [Rosenblatt and Payton, 1989; Rosenblatt and Thorpe, 1995]. In his DAMN architecture, behaviors send direction and speed commands separately, whereas an action map sent by a behavior contains (direction, speed) pairs.

We propose a mode of executing tasks by modifying and combining primitive reactions. In this approach, primitive behaviors produce continuously reactions to all the important objects in the environment. These reactions are in the form of action maps. Goal-oriented behaviors execute tasks by modifying and combining these action

maps. As the maps are produced continuously based on sensor data, the robot actions are based on the current state of the environment.

This approach produces a control system that is decomposed into producers and consumers of primitive reactions. The rationale behind this approach is that everything a robot does is a reaction to objects in the environment.

In this paper, we will apply the Samba architecture to playing soccer. We will sketch a control system for playing soccer and present the preliminary results. We decided to use soccer as an application for several reasons. First, we are developing a control architecture for mobile robots that operate in a dynamic environment and cooperate with other agents (robots and humans). Soccer is an excellent testbed for such robots. Second, the Soccerserver simulator enabled us to test the architecture right away instead of building one more simulator first [Noda, 1996]. Finally, in the RoboCup tournaments we can compare our control approach with the approaches of the other research groups [Kitano *et al.*, 1995].

In the following chapter, we will describe the Samba architecture. The third chapter will itemize the control system that controlled our soccer team in the first RoboCup competition in Nagoya, August 1997. The fourth chapter will describe the experiments. In the fifth chapter, we will analyze the Samba architecture and the RoboCup competition. The last chapter will contain the conclusions.

2 Control Architecture

The *Samba* architecture contains the following types of modules: *Sensors*, *Actuators*, *Markers*, *Behaviors*, and *Arbiters*. The architecture is shown in Fig. 1. The control system is connected to the external world through Sensors and Actuators. Sensors produce data about the local environment of the robot. Actuators change the local environment as reasoned by the behaviors. Markers ground task-related data on a sensor data flow and communicate it to behaviors. Behaviors react to unexpected events and execute tasks. Arbiters select and combine commands.

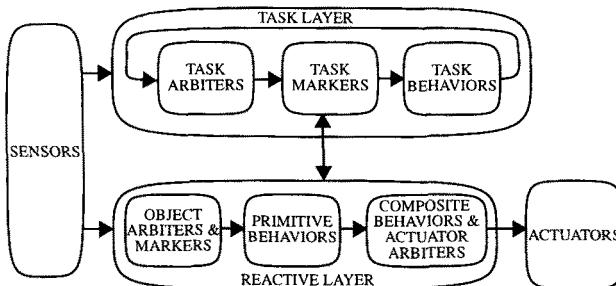


Fig. 1. Samba architecture.

The *reactive layer* produces reactions to unexpected events in the environment and the primitive actions required in task execution. The *task layer* reasons the tasks to be executed and controls the reactive layer to perform the actions which keep the task at hand to progress.

The task layer consists of *task behaviors*, *task markers*, and *task arbiters*. Task markers include task-related information about the objects manipulated or otherwise considered during task execution. They specify the goals explicitly. A task behavior executes a task by controlling a task marker. Task behaviors and task markers make up a hierarchical structure. A task marker activates behaviors at the lower layers to execute a task. Task markers at the lowest layer are connected to the reactive layer. A task arbiter receives the commands sent to a task marker. The arbiter selects the most important task from those suggested by the behaviors and sends a corresponding command to the marker.

The reactive layer consists of *object markers*, *object arbiters*, *primitive behaviors*, *composite behaviors*, and *actuator arbiters*. Object markers connect task-related objects to behaviors. They store data about the objects, update the data automatically based on sensor data, and communicate the data to the behaviors. Object markers specify the intermediate results explicitly.

Primitive and composite behaviors transform sensor and marker data into commands to actuators. Primitive behaviors produce primitive reactions to objects in the environment, such as “go to an object” and “avoid an object”. These reactions are in the form of *primitive action maps*.

Composite behaviors execute tasks by coordinating primitive behaviors. They produce *composite action maps* by modifying and combining primitive action maps based on task constraints. The constraints are specified in the task markers. For some composite behaviors the tasks are defined beforehand. These composite behaviors are reactive; together they produce the default behavior of the robot.

An *action map* specifies for each possible action how preferable the action is from the perspective of the producer of the map. The preferences are identified by associating a weight to each set of action parameters. The most common action map type is a *velocity map*, as the main task of the control system is to determine the velocity of the robot at each moment. Velocity maps are three-dimensional: an action is described in terms of a velocity vector (direction, speed), and for each action there is a weight. The weights make up a 2-dimensional surface in the polar coordinate system (direction, speed).

We have specified four different primitive action map types. Fig. 2 illustrates the different map types as one-dimensional action maps. The maps represent, in this case, turn commands.

The maps were produced in a situation in which an object was located in front of the robot. The *Goto map* contains positive weights and the *DontGoto map* negative weights for the actions driving the robot towards the object. Similarly, the *Avoid map* contains positive weights and the *DontAvoid map* negative weights for the actions avoiding the object. The Goto and Avoid maps are active; they cause actions to be performed. The DontGoto and DontAvoid maps are passive; they prohibit some actions. In the rest of this paper, we will only discuss the Goto and DontGoto map types.

The weights of the primitive action maps are calculated based on a global optimality criterion, the time to collision. A weight indicates the time it takes to reach the object if the corresponding action is performed. The global optimality criterion enables the comparison of the action maps produced by different behaviors. For example,

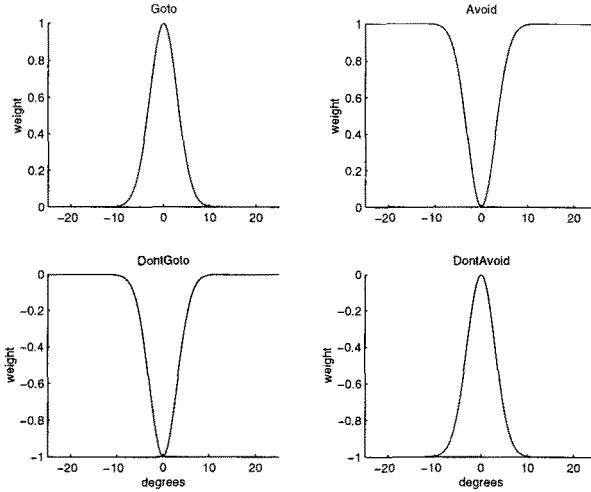


Fig. 2. Examples of different action map types.

comparing an action map for driving towards a target with a map for avoiding an obstacle can reveal that the robot would collide with the obstacle before reaching the target.

An arbiter receives a set of commands and selects one of them to be executed, or combines several of them into a single command to be executed. The combination is based on the weights included in the commands sent to the arbiter.

An actuator arbiter receives the composite action maps sent to an actuator. The arbiter combines the maps and sends the action with the highest weight in the combined map to the actuator. An object arbiter controls a set of object markers. It binds the object markers to the most important objects in the local environment.

Next, we will discuss in more detail how composite behaviors and actuator arbiters combine action maps. Action maps are combined by calculating for each action in the combined map a weight based on the weights of the maps being combined. So far, we have specified two combination methods: the weighted sum (*WS*) method and the maximum of absolute values (*MAV*) method.

In the *WS* method, the arbiter multiplies each action map by a gain and sums the resulting maps. The arbiter gains determine the mutual order of importance of the action maps. The arbiter tunes the gains dynamically based on the environment state. The weights stored in an action map determine the importance of each action from the perspective of the producer of the action map. The product of a weight and a gain determines the importance of an action in the current situation, relative to the same actions stored in the other maps being combined.

Fig. 3 illustrates the weighted sum method. The maps are one-dimensional, i.e., each action has one parameter. The parameter is, in this case, the direction of the robot relative to its current direction. The behavior producing the first Goto map suggests that the robot should turn slightly to the left, whereas the behavior producing the second Goto map suggest turning to the right. Weights of value 1.0 apply to the optimal directions (i.e., we use here a local optimality criterion). The widths of the peaks specify

the ranges of the directions that satisfy the behaviors. The arbiter has a slightly bigger gain for the first behavior (0.6 and 0.4), which means it would win the arbitration if there were not a third behavior. The behavior knows that turning slightly to the left should be avoided, because, for example, there is an obstacle. The DontGoto map produced by this behavior contains negative weights for the directions to be avoided. These negative weights cause the second behavior to win the arbitration, and the robot will thus turn to the right.

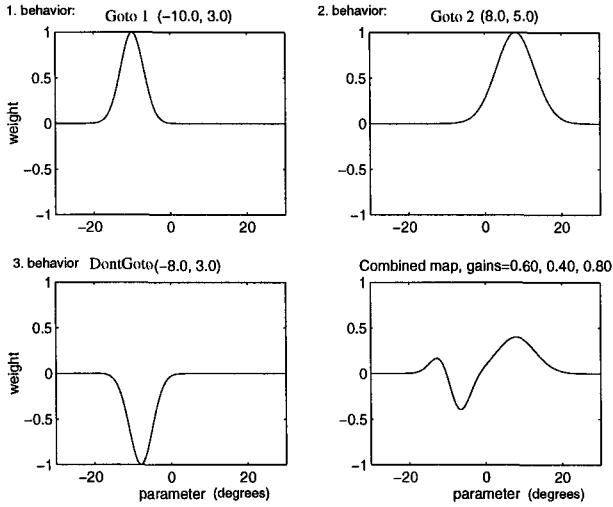


Fig. 3. An example of combining action maps using the weighted sum method.

In the *MAV* method, the arbiter calculates the weight of an action as follows: First, the arbiter computes for each map being combined the absolute value of the weight of the corresponding action. Then, the arbiter selects the weight with the maximum absolute value. Finally, the arbiter restores the sign of the selected weight, and stores the weight in the combined map.

For example, a behavior might vote for turning 10 degrees to the left by a weight of 0.6. Another behavior might vote against this command by a weight of -0.8. In this case, the arbiter stores in the combined map a weight of -0.8, as the absolute value of -0.8 is greater than the absolute value of 0.6.

The *MAV* method has the advantage that it preserves information. When the weights are calculated based on the global criterion, the weight of the combined action is the weight of the most optimal action. When time to collision is used as the optimality criterion, the weight of the action that causes a collision in the shortest time is selected.

This property is of special importance when the objects are located near each other. In a soccer game, for example, an opponent might be located behind the ball. If the actions for reaching the ball and for avoiding the opponent were to be combined using the *WS* method, the opponent would attenuate the actions for driving straight towards the ball. When the *MAV* method is used, the opponent has no effect on those actions, as the ball is closer than the opponent.

Compared with the *MAV* method, the *WS* method has the advantage that it enables a set of actions to be amplified or attenuated. For example, the actions passing the ball to team-mates in the direction of the opponents' goal can be amplified.

Composite behaviors and arbiters can use either one of the methods described above. For example, a composite behavior can first combine primitive maps using the *MAV* method and then amplify or attenuate a set of commands in the combined map using the *WS* method. In this work, we used mainly the *WS* method, because the *MAV* method was only developed recently.

3 Samba for Soccer

Fig. 4 shows a Samba control system for a soccer player. The system contains markers for all objects on the field: the opponents, the ball, the team-mates, the opponents' goal, the home team's goal, the borders, and the flags. The default location of a player is also shown by a marker. As this system plays reactive soccer, there is only one task marker, the *Specloc marker*.

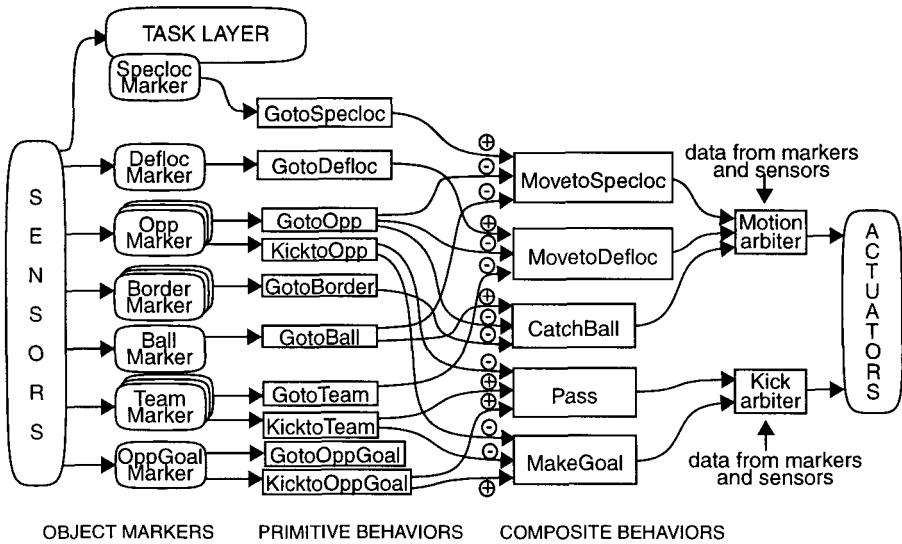


Fig. 4. A Samba control system for a soccer player. To keep the diagram clear, only part of the system is presented.

To keep the diagram clear and the discussion short, only part of the system is presented schematically in the figure and described in the text. Thus, this system should be considered rather as an example than a full description of the implemented system.

The number of team-mate (Team) and opponent (Opp) markers is limited. This is because a player does not know the locations of all the other players. Also, the behavior of a player is only affected by some of the other players. Finally, as action maps are produced for the objects identified by the markers, computation time can be shortened by constraining the number of markers.

An object arbiter selects the other players to which the team-mate and opponent markers are bound. The arbiter receives data from sensors, object markers, and task

markers. It sends commands to the object markers. The object arbiter is not shown in the figure.

Primitive behaviors produce Goto, DontGoto, Kickto and DontKickto primitive maps for the objects identified by the markers. A primitive behavior produces both Goto and DontGoto (Kickto and DontKickto) maps. The Goto and Kickto maps are marked in Fig. 4 by plus signs and the DontGoto and DontKickto maps by minus signs.

The Goto and DontGoto maps are primitive velocity maps. The Kickto and DontKickto maps are primitive *impulse maps*. They present weights for all possible different impulse vectors (direction, magnitude). The Avoid and DontAvoid maps are not utilized in this control system.

The reactive composite behaviors produce the default behavior of the player. The *MakeGoal behavior* kicks the ball towards the goal. It utilizes the KicktoOppGoal, DontKicktoOpp, and DontKicktoTeam primitive maps. The *Pass behavior* controls the player to pass the ball to a team-mate. It utilizes the KicktoTeam, KicktoOppGoal, and DontKicktoOpp primitive maps. The KicktoOppGoal map causes the team-mates in the direction of the opponents' goal to be favored.

We have not yet utilized combining composite action maps in playing soccer. The actuator arbiters select at each moment one composite action map and execute the action with the heaviest weight in this map.

The *Kick arbiter* selects the MakeGoal behavior when the player is near enough the opponents' goal and otherwise the Pass behavior. As a result, the players pass the ball to each other – avoiding the opponents – until a player near the opponents' goal receives a pass. He kicks the ball towards the goal while avoiding the team-mates and the opponents. This means that the task of scoring a goal is executed by modifying and combining the primitive reactions to team-mates, opponents, and the opponents' goal.

The Kick arbiter receives from the sensors and markers the necessary data to be able to select a composite behavior. It sends a command to the actuator only when the ball is near enough.

The *CatchBall behavior* guides the player towards the ball. It utilizes the GotoBall, DontGotoBorder, and DontGotoOpp primitive maps. The team-mates need not be considered, as this behavior is only selected when the player is nearer the ball than the team-mates.

The *MovetoDefloc behavior* guides the player towards his default location while avoiding the team-mates and opponents. It utilizes the GotoDefloc, DontGotoTeam and DontGotoOpp primitive maps.

The *GotoDefloc behavior* produces a primitive map guiding the player towards the default location indicated by the *Defloc marker*. This marker computes the default location based on the role of the player and the location of the ball. For example, the default location of an attacker moves from the half line towards the opponents' goal when the ball moves from the home team's goal towards the opponents' goal. The default locations of the players are shown in Fig. 5.

The *Motion arbiter* selects the CatchBall behavior when the player is nearer the ball than the team-mates and otherwise the MovetoDefloc behavior if the player is too far from the default location. Thus, the player is not forced to stay exactly at the default location, but can move freely inside a zone centered at the default location.

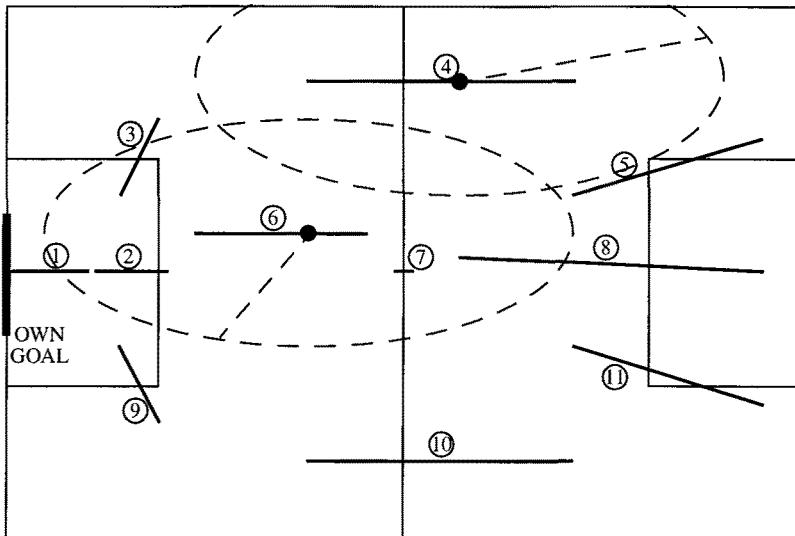


Fig. 5. Default location lines of the players. The default location of a player moves along the line based on the location of the ball. The ellipses define for players 4 and 6 the allowed operation areas when the default locations are at the locations marked by the circles.

The Motion arbiter also receives a composite map from the *MovetoSpecloc behavior*. This map is selected when the task to move to a special location is active. The MovetoSpecloc behavior produces a composite map controlling the player towards a special location. A special location is a location from which the player should make a corner kick, goal kick, side kick, or free kick. For example, when a free kick is to be executed, the special location is a location that is near the ball and at the opposite side of the ball compared to the opponents' goal.

The MovetoSpecloc behavior receives the following primitive action maps: GotoSpecloc, DontGotoBall, and DontGotoOpp. The DontGotoBall map is used, as the player might need to circumvent the ball to reach the special location. Thus, the behavior controls the player to move to the special location while avoiding the ball and the opponents. The team-mates need not be avoided, as the player nearest the ball is chosen to execute the kick.

The GotoSpecloc action map is calculated based on the data produced by the Specloc marker. This marker serves as an interface between the task and reactive layers. The Specloc marker is controlled by behaviors at the task layer. The task to move to a special location is triggered when the referee has judged for the team either a goal kick, corner kick, side kick, or free kick and the player is the one who should execute the kick.

A task marker is not necessarily connected to a primitive behavior. When all the primitive behaviors needed to execute the task indicated by the marker already exist, the marker is connected either to a composite behavior or to an actuator arbiter. If the combination of primitive behaviors is affected by the task, the marker is connected to a composite behavior. Otherwise, the marker is connected to an actuator arbiter.

4 Experiments

In this chapter, we will describe experiments conducted with the control system described in the previous chapter. For the first set of experiments, both the control system and the simulator were written using the Matlab language. For the second set of experiments, the control system was written using the C++ language.

The first set of experiments was carried out as follows: At the beginning of each experiment, the player is moving to the right. The control system and the simulator are executed in turns. First, the control system calculates an action based on the sensor data. The sensor data indicate the positions and velocities of all objects in the environment. After that, the simulator moves the player based on the action and the objects based on their predefined trajectories.

In the first set of experiments below, we demonstrate the capabilities of the control system. In these experiments, a CatchBall behavior controls the player to catch the ball while avoiding the opponents. This behavior combines the GotoBall and DontGotoOpp primitive action maps.

Fig. 6 shows the first two experiments. In the first experiment, we show that the system is able to consider environmental dynamics. The player reaches the right location at the right time.

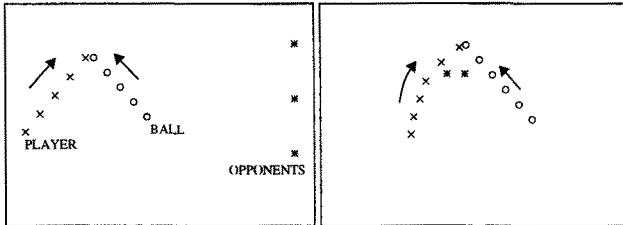


Fig. 6. The first two experiments. The trajectory of the player is shown by X's and the trajectory of the ball by O's. The opponents are marked with *'s. The arrows indicate the directions of motion. Left: environment in the first experiment. Right: environment in the second experiment.

Fig. 7 shows the action map produced by the GotoBall behavior at the first step of the first experiment. The heaviest weight is given to the action for reaching the ball at an optimal speed. The optimal speed is not the maximum speed, as the player can maintain its maximum speed only for short periods. The other actions that have heavy weights also cause the player to reach the ball.

As the opponents are not near, the effect of the DontGotoOpp action maps is insignificant. Thus, the player executes the actions with the maximum weight in the GotoBall action map. The first GotoBall action map causes the player to turn into the optimal direction. As the velocity of the ball does not change during the experiment, the player does not have to turn again. It runs forward and maintains a suitable speed to reach the ball. That is, the constant ball speed causes all the GotoBall action maps produced after the first step to have maximum weights in the direction of the motion.

In the second experiment, we show that the system is able to maintain the properties shown in the first experiment while executing several tasks in parallel. In this experiment, static opponents block the path used in the first experiment (see Fig. 6). The GotoBall action maps resemble the maps in the first experiment, but the DontGotoOpp

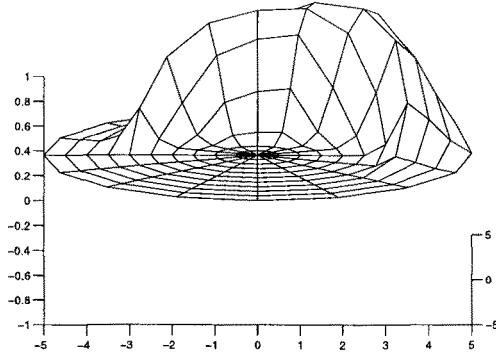


Fig. 7. The GotoBall action map at the beginning of the first experiment.

action maps have a greater effect on the player's behavior. Fig. 8 shows a DontGotoOpp action map produced for the leftmost opponent at the beginning of the experiment, when the player is moving to the right.

The CatchBall behavior combines the GotoBall and DontGotoOpp maps into a combined map (see Fig. 8). The best action is marked in the figure. The DontGotoOpp action map causes the player to select another path towards the moving ball. However, after the player has circumvented the opponents, it returns to the optimal path. Actually, the player turns continuously to the right as more optimal directions become possible.

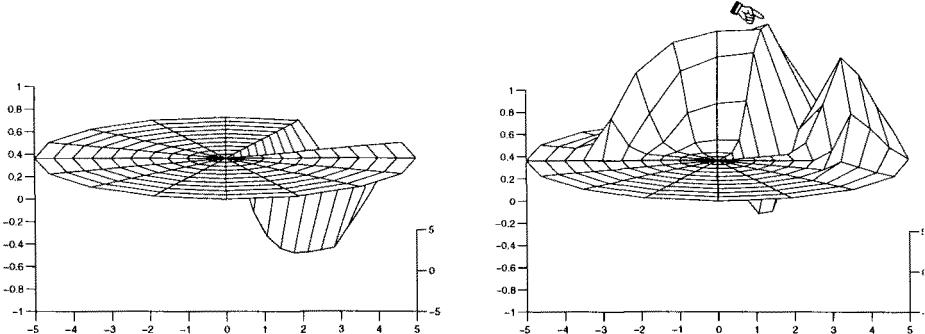


Fig. 8. Two action maps produced at the beginning of the second experiment. Left: DontGotoOpp action map for the leftmost opponent. Right: CatchBall action map.

In the third and fourth experiments, we show how the player reacts to changes in the environment. In these experiments, the ball changes abruptly its direction and speed (see Fig. 9). In the third experiment, the ball changes its direction. The player goes near the leftmost opponent, because the DontGotoOpp action map inhibits only directions causing collisions. In the fourth experiment, the ball changes its speed. The player reacts to this change by adjusting its direction.

In the second set of experiments, the control system is connected to a player in the SoccerServer's simulated world [Noda, 1996]. A player in SoccerServer can observe the locations and velocities of other objects in the field and execute dash, turn, and shout

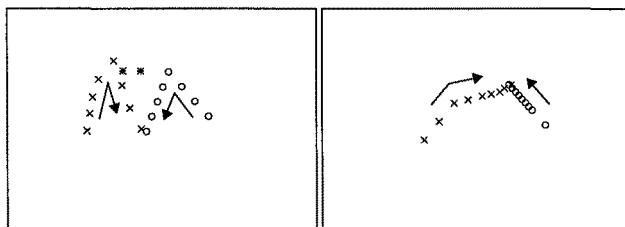


Fig. 9. The third and fourth experiments. The trajectory of the player is shown by X's and the trajectory of the ball by O's. The opponents are marked with *'s. Left: environment in the third experiment. Right: environment in the fourth experiment.

actions. The observed objects are the ball, the other players, the goals, the borders, and the flags. The players can communicate by observing each other and by shouting messages to each other. These messages are also heard by the opponents.

The experiments were carried out in the first RoboCup competition. 11 copies of the control system were connected to 11 players. Our team, CAT_Finland, played against teams implemented by other research groups. In the first round, we won two out of three matches. In the second round (elimination round) we lost the match. Among 29 teams, our final position was 9.-16.

Fig. 10 shows a situation in our first match. Our team plays from right to left. Player number 5 dribbles the ball near the opponents' goal. As the goal is blocked by opponents, he passes the ball to player number 8, who in turn passes the ball to player number 11. Next, player number 8 circumvents the opponents. Finally, player number 11 passes the ball back to player number 8 who scores a goal.

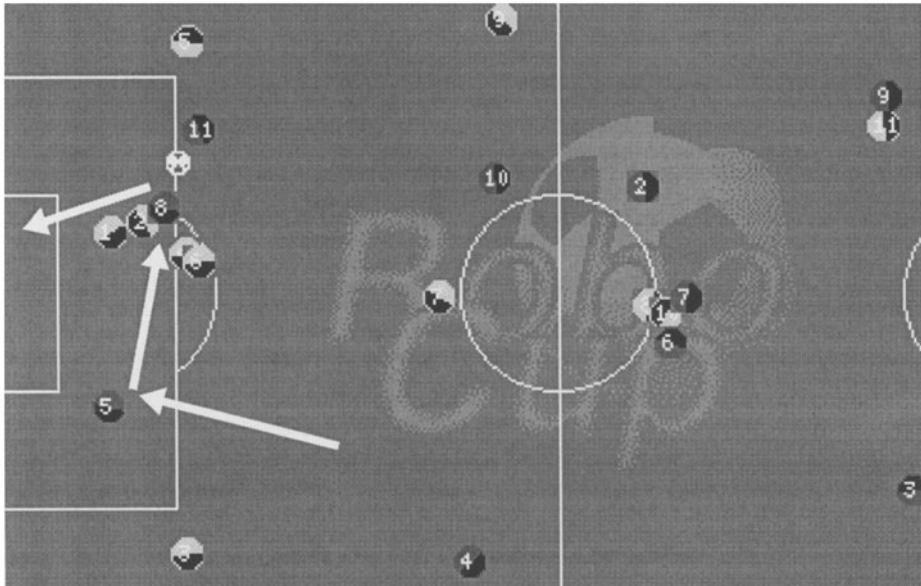


Fig. 10. Our team is attacking from right to left. The arrows show the trajectory of the ball. The players number 5, 8, and 11 attack and score a goal.

5 Discussion

The Samba architecture is a goal-oriented and behavior-based architecture. Tasks are executed by producing primitive reactions to the objects in the environment and by combining and modifying these reactions based on task constraints.

The primitive reactions are presented as action maps. As each primitive action map represents all the possible different reactions to an object, a composite behavior can produce actions that satisfy constraints on several objects in parallel. The action with the heaviest weight in the composite map is an optimal action for the composite behavior. Even though that action does not necessarily give maximum satisfaction to any one primitive behavior, it can still adequately satisfy several primitive behaviors. For example, the CatchBall behavior can select an action that does not reach the ball in the shortest possible time or avoid the opponents as much as possible but yet reaches the ball and avoids the opponents. The same holds for the arbiters: they can produce actions that satisfy several composite behaviors in parallel.

In addition to the representation of robot actions, the Samba architecture contains representations for intermediate results and goals. Object markers indicate intermediate results – the important objects in the environment. The behaviors receive from these markers the location and velocity of the objects. This reduces the amount of computation, as the behaviors do not have to compute the locations and velocities from raw sensor data. Further, the object markers serve as an interface between the producers and consumers of data. This increases the modularity of the system.

Task markers indicate the goals (that is, the objectives) explicitly. They are used both to coordinate behaviors based on given goals and to manage the complexity of the system. A task marker serves as an interface to a task-executing subsystem. All behaviors in the subsystem can be activated by activating the marker. Further, the programmer does not have to understand every single detail of the subsystem in order to use it. It is enough to know the interface.

The behavior coordination method of the DAMN architecture corresponds to one-dimensional action maps [Rosenblatt and Thorpe, 1995]. However, in the DAMN architecture behaviors send direction and speed commands separately, whereas an action map sent by a behavior contains both direction and speed commands. As a consequence, environmental dynamics are easier to take into account in the Samba architecture than in the DAMN architecture. This is because the selected direction and speed always satisfy the same behaviors.

We suggest here decomposition of the control architecture into producers and consumers of primitive reactions. A prerequisite for such decomposition is that all the actions of the robot are reactions to observable features of objects. We have found this a reasonable assumption. As the main task of the robot is to interact with the environment, it is surrounded by a rich set of observable features all the time.

We admit that some situations require goal points that are not observable. In soccer, for example, each player can have a default location specified on the basis of its role. As there are not many observable features in the field, a player cannot be instructed next to such a feature. A similar situation arises when a robot is cleaning a large hall. However, even in these situations the goal point can be associated with observable features. The goal point can be represented in a local coordinate system spanned by

observable features. For a robot cleaning a hall, the coordinate system can be spanned by landmarks on the wall. For a soccer player, the coordinate system can be spanned by lines and flags.

In this paper, we applied the Samba control architecture to playing soccer. We presented the preliminary experiments on playing soccer by modifying and combining primitive reactions.

Most of our effort was spent on developing individual behaviors and thus creating independent players with certain skills which they could modify and tune according to the prevailing situation. It was thus a pleasant surprise that the independent players did actually operate as a team, were able to dribble the ball despite the opponents, pass the ball to team-mates, and once in a while score a goal.

The pass play of our team worked well. Individual players could mostly select an optimal direction and a suitable team-mate whom to give the pass. This resulted in good and sometimes beautiful pass combinations, where the ball was moved towards the opponents' goal efficiently. Unfortunately, our players were not aggressive enough when the ball was within a scoring distance. Instead, we had a tendency to pass once more and the scoring situation was hence lost. Our pass play operated well as long as the opponent was not able to move with much higher speed than our players.

The defence play was also quite efficient. The immediate front of our goal was kept clear (most of the time), but this resulted too often in a side kick. This was one of the weaknesses of our team. Furthermore, the primitive operation to make a side kick was not good enough, causing failures and thus giving advantage to the opponent.

It was interesting to notice that a group behavior could be achieved without any explicit coordination. The interplay of Pass and MovetoDefloc behaviors caused the players to attack when the team had the ball. The Pass behavior favors the team-mates in the direction of the opponents' goal. The MovetoDefloc behavior guides the player towards the default location, which is calculated based on the location of the ball. Thus, when a player passes the ball towards the opponents' goal, all team-mates move into the same direction.

This first control system utilizing action maps proved that the calculation of action maps is computationally not so expensive as one might expect. The calculation of a primitive action map took 0.5-3 ms on a Sun Sparcstation 20.

We can conclude, based on the experiments carried out in the RoboCup competition, that the Samba control system can be utilized in executing tasks in a complex and highly dynamic environment. However, as we did not have enough time to test the program thoroughly nor to tune the numerous parameters, we cannot yet say how complex tasks the control system is able to execute. This remains to be seen in the future RoboCup competitions.

6 Conclusions

In this paper, we suggested a novel approach to executing tasks. The key idea is to produce continuously primitive reactions for all the important objects in the environment. Tasks are executed by modifying and combining these reactions. As the

reactions are produced continuously based on sensor data, the robot actions are based on the current state of the environment. We applied the architecture to playing soccer and carried out experiments in the first RoboCup competition.

Acknowledgments

The research presented in this paper was supported by the Academy of Finland, the Scandinavia-Japan Sasakawa Foundation and the Tauno Tönnig Foundation. The authors also express their gratitude to Jussi Pajala, Antti Tikkanmäki, and Petri Erälä for their hard work in making Samba tick.

References

- [Brooks, 1986] Brooks, R.A. (1986) A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14-23.
- [Brooks, 1990] Brooks, R.A. (1990) Elephants don't play chess. *Robotics and Autonomous Systems* 6(1-2):3-15.
- [Chapman, 1991] Chapman, D (1991) Vision, instruction, and action, MIT Press.
- [Connell, 1990] Connell, J.H. (1990) Minimalist mobile robotics: a colony-style architecture for an artificial creature. Academic Press, San Diego, USA, 175 p.
- [Kitano *et al.*, 1995] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. & Osawa, E. (1995) RoboCup: The Robot World Cup Initiative. In: Working Notes of IJCAI Workshop: Entertainment and AI/Alife, pages 19-24.
- [Kuniyoshi *et al.*, 1994] Kuniyoshi, Y., Riekki, J., Ishii, M., Rougeaux, S., Kita, N., Sakane, S. & Kakikura M. (1994) Vision-Based Behaviors for Multi-Robot Cooperation. In: Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems, Munchen, Germany, September 1994.
- [Noda, 1996] Noda, I. (1996) Soccer Server Manual, Rev. 1.01. Electrotechnical Laboratory, Japan. <http://www.csl.sony.co.jp/person/kitano/RoboCup/RoboCup.html>
- [Riekki and Kuniyoshi, 1995] Riekki, J. & Kuniyoshi, Y. (1995) Architecture for Vision-Based Purposive Behaviors. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Pittsburgh, USA, August 1995. pp.82-89.
- [Riekki and Kuniyoshi, 1996] Riekki, J. & Kuniyoshi, Y. (1996) Behavior Cooperation Based on Markers and Weighted Signals. In: Proceedings of the Second World Automation Congress, Montpellier, France, May 1996, in press.
- [Rosenblatt and Payton, 1989] Rosenblatt, J.K. & Payton, D.W. (1989) A fine-grained alternative to the subsumption architecture for mobile robot control. In: Proc. of the IEEE/INNS International Joint Conference on Neural Networks, Washington DC, June 1989. vol. 2 pp. 317-324.
- [Rosenblatt and Thorpe, 1995] Rosenblatt, J.K. & Thorpe, C.E. (1995) Combining goals in a behavior-based architecture. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'95), Pittsburgh, USA, August 1995, pp 136-141.
- [Tsotsos, 1995] Tsotsos, J.K. (1995) Behaviorist intelligence and the scaling problem. *Artificial Intelligence* 75:135-160.

Learning, Deciding, Predicting: The Soccer Playing Mind

Andrew Price, Andrew Jennings, John Kneen, Elizabeth Kendall

andrew.price@cse.rmit.edu.au, ajennings@rmit.edu.au,
JohnK@rmit.edu.au, Kendall@rmit.edu.au

Department of Computer Systems Engineering
Royal Melbourne Institute of Technology
Melbourne Victoria Australia.

Abstract

In this paper we present an architecture which does not rely on any fixed hierarchy of agents in an attempt to remove dependencies and predictability from our system. We also present one method of simplifying the choice between available strategies by predicting the type of chassis used by the opposition, and therefore its limitation of movement. We consider the question of when are learning methods appropriate and in what areas are other, simpler approaches more efficient. We also develop coaching as a method of implanting simple maneuvers and tactical planning into our system. We consider the problem of how to distribute intelligence within the system, and the degree of independence and autonomy accorded to each player.

1. Introduction

Considerable research has gone into various architectures, agents and systems for the control of autonomous robots.(Brooks 1986; Asada 1996). Approaches range from individual robots controlled using simple state machines through to complex multi-agent systems. Increasingly, the problem of the coordination and collaboration between multiple robots is being explored. (Michael K. Sahota 1994; Noda et. al. 1996; Sean Luke 1996; Sorin Achim 1996). One such problem is RoboCup (Hiroaki Kitano 1995), the world robotic soccer championship. Soccer is a multi-player game where players have incomplete information concerning the game state (Noda et. al. 1996). As such the approach to this problem offers significant opportunities for research into coordination, and cooperation in dynamic team based activities.

Noda (Noda et. al. 1996) has developed a simulated environment in which to explore the problem of multi-agent systems and soccer. It provides for the experimentation of cooperation protocol, distributed control and effective communication. Noda (Noda et. al. 1996) describes the important characteristics of soccer:

- Robustness is more important than elaboration. A team should take a fail-safe strategies, because play plans and strategies may fail by accidents.
- Adaptability is required for dynamic change of plans according to the operations of the opposing team.
- Team play is advantageous
- A match is uninterrupted. Each player is required to plan in real-time, because it is difficult to make a whole plan before plays.
- As precise communication by language is not expected, effectiveness must be provided by combining simple signals with the situation.

2. Physical Agents

RoboCup provides two challenges: a completely simulated soccer tournament based on the environment of (Noda et. al. 1996), and a competition for real robots as described by (International and RoboCup-97 1996). Our research efforts focus on the real robot system for the middle sized league and the architecture we present has been specifically developed for that competition. One of the major reasons for this approach is because of the flexibility that this class of robot (International and RoboCup-97 1996) allows us in the distribution of our control system. The smaller scale is more physically restricting in what may be carried on board the robot. We are using the process of developing a real soccer playing system as the basis for various coordinated robot and vision analysis research programs for use in real world industrial problems.

The architecture for our robotic soccer system differs from other approaches,(Sorin Achim 1996) in several aspects: firstly we are distributing the intelligence of each robot between on board systems and external systems. Our robots are not simply trolleys that are interfaced to an external system and act under remote control. We are incorporating several intelligence components, such as the real time reactive system, on board each robot. There are several good reasons for this approach. Including components of the intelligent control system on board increases the autonomy of the individual robots and in the case of real time reactivity, creates a tighter fusion between sensors, control and effectors.

The second significant feature of our approach, [Tomohiro Yamaguchi, 1996 #26] is that we are following the concept that some intelligent behaviors can be imparted by instruction and coaching while other intelligent behaviors must be learned. Skills of ball control and kicking do not need to be learned, they can be imparted by use of rules, algorithms and coaching. Learning in our system is aimed at the combining of simple skills to form more sophisticated behavior, and in modifying the basic skills to achieve different performance. The focus of learning in our robots is in gaining and refining skills that cannot be achieved any other way in reasonable time or must be adapted during the course of a match.

Our robots consist of an omni-directional chassis and 360 degree video camera. The chassis is capable of rapid acceleration in any direction from a standing start and has a power to weight ratio roughly twice that of a typical remote control car of similar size. The vision system can sample 360 degrees of the area surrounding the robot, and a low power real time image processing unit is mounted inside each robot. This dramatically reduces the required communication bandwidth if raw video data is not shipped off board for processing.

Our robot does not use sophisticated kicking methods since it was found by simple application of the laws of kinetic energy that the robot colliding with the ball would generate sufficient force to suitably displace it. The vision system and ability of our chassis to move in any direction allow us to reduce problems, such as dribbling the ball, to simple control algorithms as shown in Figure 1.

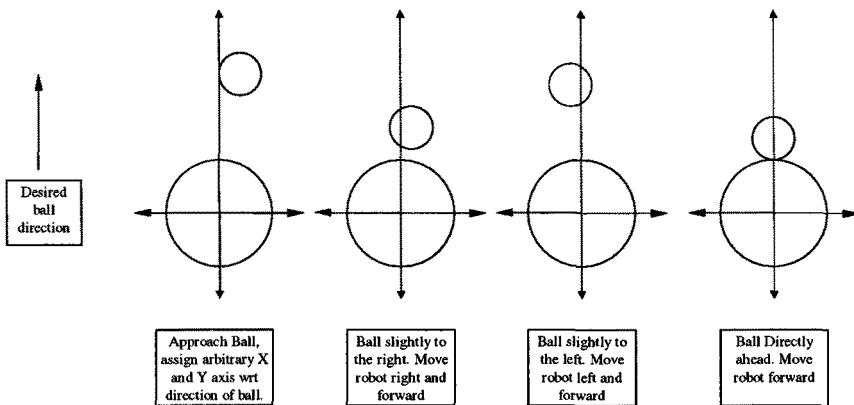


Figure 1 Simple ball dribbling algorithm

We believe that it is not necessary for our robots to learn absolutely everything, and so fundamental skills such as dribbling, kicking, shooting for goal and passing, are treated as control system problems rather than as learning problems.

The learning problem that we are focusing on is how to combine such simple tasks into complex meaningful behavior and how to adapt basic skills to exhibit more sophisticated behavior. For example if in the act of shooting for goal, the robot actually moves slightly off center of the ball, how is the trajectory of the ball affected? The actual algorithm for shooting remains the same with the addition of an offset in the position of the robot at the time of kicking. A new way of kicking does not need to be learned, simply adapt the existing one. This can be done with a simple algorithm for kicking, and by coaching the algorithm vary its behavior. In addition, we are focusing on the problem of how to select between two or more complex behaviors of relatively equal merit. This problem is different to training algorithms to be more efficient since both strategies may be entirely suitable under different circumstances. For example, certain strategies for passing and shooting may work well against one opposition, but when applied to a different opposition they may

prove to be ineffective. In order to make such distinctions we must first make predictions about the opposing team. For this reason our architecture incorporates predictive agents whose purpose is to examine aspects of the game, the opposition, and our own team and try to distil information which will help us to decide between comparable strategies.

3. Predictive Agents

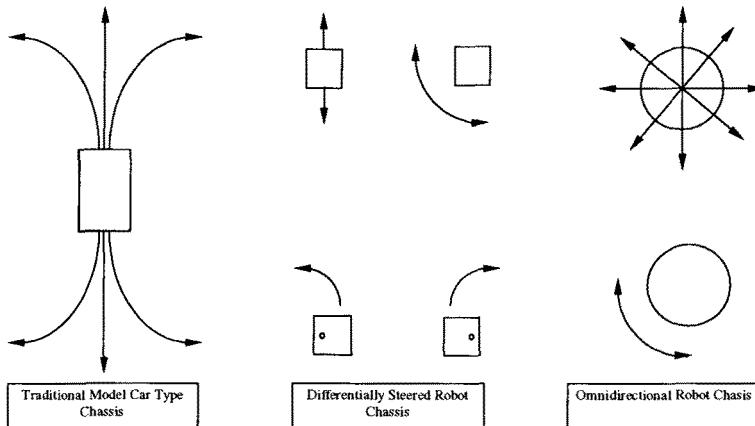


Figure 2 Relative motion of three likely chassis types

One such predictive agent determines the difference between opposition team chassis types. We consider three chassis most likely to be used in the middle sized competition; the standard ‘model car’, the differentially steered robot, and the omnidirectional robot. Basic differences between the movements of each chassis type are shown in Figure 2.

An agent which predicts the chassis type observes the motion of the robot in question and compares data received with the patterns above and infers the motions which the opposition team are capable of making. It is quite evident that a robot based on the model car chassis has only limited movement in directions parallel to its axles, but is far more likely to have significant forward and reverse speed. This kind of prediction is not uncommon in human sporting events - ‘sizing up’ your opponent is a typical part of many team games.

4. Robot Architecture Considerations

Since we are developing a control architecture for real robots we must take the physical nature of our agents into consideration. The simplest approach is to make robots which have a very basic chassis, possibly with cameras attached, and keep all of the intelligence off field, essentially replacing a human remote control with a computer remote control. This is effective provided the communication channel is of

suitable bandwidth, or the instruction set to each of the robots is limited. Very little on board processing is possible in such a system. The must defer back to the off field system for any decision making process. Despite the fact that individual robots may be equipped with different effectors, control depends heavily on an extended link between 'brain' and 'body'. For economic reasons we are unable to use individual off field systems to control each robot separately so we would be dependent on one centralized system for all of our decision making.

At the opposite extreme, all of the decision making capability could be incorporated on board each robot. Each robot would contain real time reactive control along with planning, possibly prediction and then solve the problem of communicating and coordinating with its team members. Clearly this approach is flawed if every player comes up with an idea and then has to try and convince the others to come along. Our robots would spend more time negotiating than playing soccer.

Soccer is a game for multiple players coordinating together, but this does not mean that the individual skills and personal attributes of each player are identical, nor are they completely obliterated in favour of the team mentality. It is quite possible for one player to perform individually on the field, without any assistance at all.

Our Soccer system can be viewed from two perspectives:

- It is a group of unique individuals, possibly with significantly varying abilities and skills and each with a unique perspective on the game, but each with a common purpose.
- It is a team of players coordinating and collaborating together to achieve a common goal, who must set aside individual aspirations in favour of the team mentality

By utilising on board control we are allowing each robot to express its own uniqueness, and the ability to take charge of a situation, for example, the event that the ball lands at the feet of one robot without warning. A likely move for this robot would be to attempt to trap the ball and possibly make a run for the goal, or play keepings off until the team oriented decision making process catches up with the current situation. We have therefore chosen to distribute control between onboard and off field systems with centralised coordination and collaboration. In the next section we present our architecture for achieving this distribution.

5. Architecture

Traditionally there have been two types of control architectures, horizontally partitioned and vertically partitioned (Brooks 1986) (Jennings 1994). Vertically layered behavioral architectures have the problem of how to code increasingly complicated behaviors and how to successfully order a group of behaviors one above another (Asada 1996). Traditionally, the next layer up has been a more sophisticated behavior that subsumes control when certain conditions are fulfilled.

Determining these conditions particularly for a more complex behavior is also a problem. Horizontally partitioned systems generally produce one path only from sensors to effectors. Information is received, filtered, processed and responded to in a sequential fashion. This is the typical approach to most industrial control problems where it is always desired to perform the same operation in the same way. Unlike vertically layered behavioral architectures there is little redundancy built into this architecture. Although both architectures have their place we consider several problems when implementing either approach. How can a horizontally partitioned architecture be distributed between on board and off field systems? How does such a system cope with changing its behavior to compete with the current opposition team? How does passing relate to kicking in a layered architecture? Which of these two behaviors is the higher level behavior in a Vertically layered system? What is the minimal behavior of soccer playing robots? Why is it necessary to have our robots learn everything from scratch?

It is important to note that both systems have their place, however we feel that neither are particularly suited to the problems inherent in constructing physical robots to play soccer. We present in the following section our approach to the problem and identify those problems that are still under investigation.

In our architecture we use three types of agents

1. Real-time reactive agents which take inputs, process in a timely fashion and modify outputs to effectors accordingly. Inputs can be sensors, filtered data from a camera or a request to perform from another agent.
2. Deliberative agents take the same information from inputs as reactive agents, but whose response is not immediate. Typically a deliberative agent in our system is not considering problems which must be solved and implemented prior to a change in the state of the game. Deliberative agents plan longer term strategies and implementations, often incorporating other agents into their plans, whose functions are already known.
3. Predictive agents consider inputs from all sources but rather than affect outputs and effectors directly, act as inputs to other types of agents, for example suggesting a possible place to enact a blocking strategy.

In our architecture we place no hierarchy on any of these types of agents, all are treated as being of equal significance. Nor do we seek to order agents in any particular fashion, with one subsuming control directly after another because we feel that this order is not likely to remain constant. Soccer is a dynamic game with unpredictability and incomplete information. We can estimate certain strategies that are required at a fundamental level, but if these strategies are countered by the opposition, then they will be ineffective. For example, if an opposition team worked out that "if their robot with the ball stops still, its next move is always to the right" then they could predict the best place to counter. Hence having a fixed order of subsumption would not be beneficial.

As with other architectures we allow all agents to receive and process inputs simultaneously. Not all agents will process the information at the same rate or seek to

enact their output at the same time. Similarly, if we have one agent enacting its output we may not want other agents taking control.

6. Relationships Within The Team

Figure 3 shows the relationships that we believe are important in a team of soccer playing robots. The coach provides training feedback, positive and negative rewards as well as providing new concepts for strategies. The coach can interface directly to the entire team, or to individual members of the team.

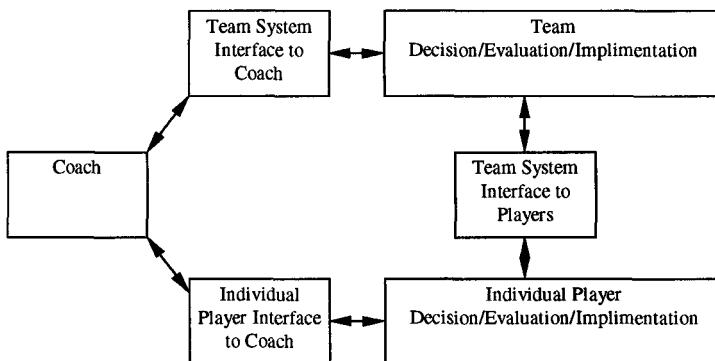


Figure 3 Relationships within our soccer system

For example, the coach may suggest a two player move in which one robot moves down the left flank with the ball and then passes it across goal to a player who has moved down the right flank who then takes a shot at goal. The mechanism for the coach to suggest this play is shown in Figure 4. ‘The Coach’ or Tactics Planning is a way of inputting approximate suggestions for what may be suitable plays in a graphical sense. All typical logic constructs may be used in developing plays. These plays can then be refined using positive and negative rewards and via corrections and adjustments from the coach to help the two robots concerned complete the play effectively.

At some stage during the game, any one of five of our robots may acquire the ball in the defensive half on the left wing. If all robots are identical in operation and effectors then our soccer system does not concern itself with which robot. The play itself becomes viable if two robots are approximately in the starting positions of this play. Significantly, the play can be *made* viable if prior to selecting the agent to implement this strategy, another agent is called which specifically sets up the conditions for this strategy.

Interfacing directly to individual members of the team is particularly important in cases where for example one player is physically different to another player and needs specific training on how to use its effectors most profitably. The coach can be

used to implant knowledge to all members or selected members of the team, and to develop each type of agent in the decision shell.

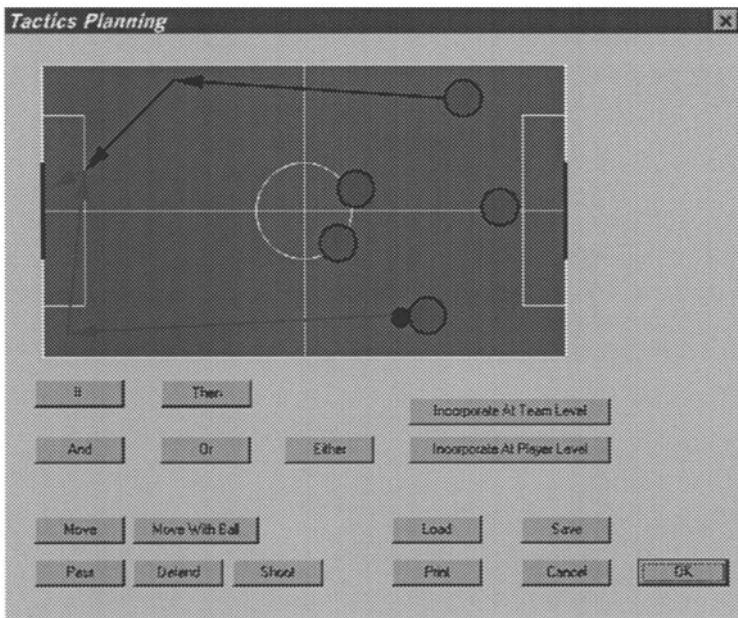


Figure 4 'The Coach'

7. Software Model

Each robot is implemented using the model shown in Figure 5. The Implementing Core handles motor control, collision sensors, vision and other I/O. Information gathered is available to all 'shells' within the model, consequently, it may be passed to other robots or off field systems via the communications layer.

All decision making agents exist within the Reacting/Planning/Anticipating (or Decision) shell. There can be any number of these agents, with varying capabilities from simple state machines through to complex fuzzy logic learning agents and genetic algorithms. Each agent within this shell will maintain a record of its past, as well as user programmed information relating to its importance and value. To take a simple case, one agent that will exist in each robot will avoid the walls. This agent will have the highest priority of any agent, and will not, as a result of its priority, be able to be overridden by another agent intent on colliding with the walls. A different agent, for example one which passes the ball, may indicate that it was successful in the past against car based chassis but not against differentially steered robots. Hence it is less likely to be considered a viable choice by the evaluation shell than one which has proven successful against differentially steered robots.

The Evaluation Shell provides the greatest number of challenges and problems, many of which have no one single solution and must be approached with care. The purpose of this shell is to examine each of the agents in the Decision Shell that have signified their readiness to produce an output and attempt to choose between them. At the same time data gathered from various anticipating agents is also considered. A number of fundamental problems exist, for example, how to correctly award merit to each agent? How to evaluate success or failure of a strategy? How to consider outside influences on the process of reward? Our research attempts to address these problems in the domain of robotic soccer.

A merit based selection system is being investigated for this shell, and we are developing a mechanism using the concept of coaching to improve the allocation of merit to each available agent. Several features of this system are probable however. Reactive agents are likely to have a higher priority than most deliberative agents due to their need to act in a timely manner. Predictive agents can influence the choice of strategy and the types of strategy which are available for selection.

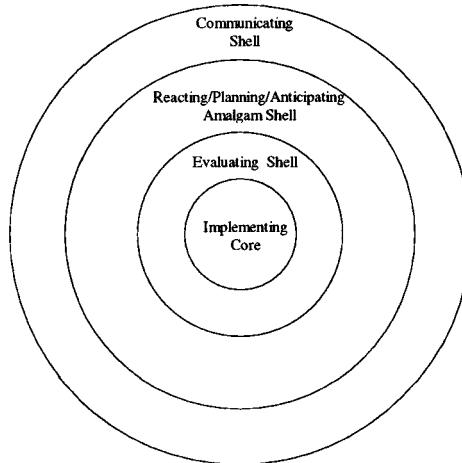


Figure 5 Control Structure for One Robot

The evaluation shell also considers the strategy currently being implemented and if it finds that the strategy is not performing as desired it can 'cancel' that strategy in favour of a more current or beneficial strategy. One example of this is if a pass is intercepted in the middle of a strategy and the ball is lost to the other side, then continuing with that strategy is pointless and should be ceased.

Control of the team can be described in the same manner as for each individual robot as shown in Figure 6, with the implementation core now consisting of five team members, all with their own unique structure as previously described. The individual shells perform exactly as before except that the agents present in the Decision Shell will be considering different problems than each individual robot. Agents at the system or team level implement strategies affecting the team rather than the individual. As before, all I/O information is available to any shell and may, as a

consequence of the communicating shell be passed out of the system, in this case, to the user interface or ‘Coach’. Similarly, information fed back to the system from individual robots is also available to the System Communicating Shell, hence we have demonstrated the presence of each of the relationships of Figure 3 within our architecture . As described for a single robot, the Evaluation Shell provides the greatest fundamental problems and we are investigating a similar merit based system implemented at the team level.

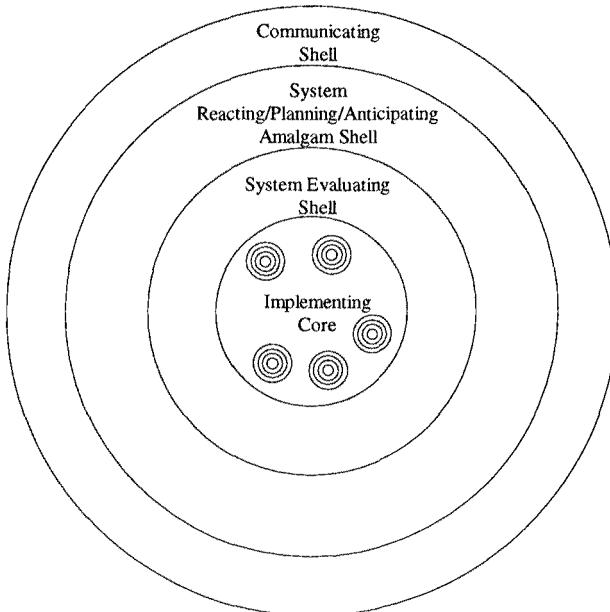


Figure 6 Control Structure for Complete System

8. Conclusion

We present an architecture which does not rely on any fixed hierarchy of agents in an attempt to remove dependencies and predictability from our system. In a competitive match, this element of variation in game plays is of great advantage. We also present one method of simplifying the choice between available strategies by predicting the type of chassis used by the opposition, and therefore its limitation of movement. Our approach to the use of learning is to only employ a learning method when all other methods are not possible. Simple passing and shooting maneuvers and a great deal of tactical planning can be implanted using our coaching approach.

We consider the problem of how to distribute intelligence within the system, and the degree of independence and autonomy accorded to each player. Although these problems are fundamental, we feel that limited solutions within the domain of soccer is a feasible goal for our research.

Bibliography

- Asada, M. (1996). "Preface to Advanced Robotics." Advanced Robotics, Vol 10. No 2.: 139-142.
- Brooks, R. A. (1986). "A Robust Layered Control System for a Mobile Robot." IEEE Journal of Robotics and Automation, RA-2, Number 1, March 1986.
- Hiroaki Kitano, M. A., Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa (1995). "Robocup: The Robot World Cup Initiative." IJCAI-95 Workshop on Entertainment and AI/Alife: 19-24.
- Noda et. al. , Itsuki Noda, Hitoshi Matsubara. (1996). "Soccer Server and Researches on Multi-Agent Systems." Proceedings of IROS-96 Workshop on RoboCup: 1-7.
- International and c. a. p. f. RoboCup-97 (1996). "RoboCup Real Robot League Regulations." Available at <http://www.robocup.org/RoboCup/rule.html>.
- Jennings, M. W. a. N. R. (1994). "Agent Theories, Architectures and Languages: A Survey." Intelligent Agents, ECAI-94 Workshop on Agent Theories, Architectures and Languages, Proceedings.
- Michael K. Sahota, A. K. M. (1994). "Can Situated Robots Play Soccer." Canadian AI-94.
- Sean Luke, L. S. (1996). "Evolving Teamwork and coordination with Genetic Programming." Genetic Programming 96, Stanford, July 1996.
- Sorin Achim, P. S., Manuela Veloso (1996). "Building a Dedicated Robotic Soccer System." Proceedings of IROS-96 Workshop on RoboCup: 41-48.

Using Decision Tree Confidence Factors for Multiagent Control

Peter Stone and Manuela Veloso

Computer Science Department

Carnegie Mellon University

Pittsburgh, PA 15213

{pstone,veloso}@cs.cmu.edu

<http://www.cs.cmu.edu/{~pstone,~mmv}>

Abstract. Although Decision Trees are widely used for classification tasks, they are typically not used for agent control. This paper presents a novel technique for agent control in a complex multiagent domain based on the confidence factors provided by the C4.5 Decision Tree algorithm. Using Robotic Soccer as an example of such a domain, this paper incorporates a previously-trained Decision Tree into a full multiagent behavior that is capable of controlling agents throughout an entire game. Along with using Decision Trees for control, this behavior also makes use of the ability to reason about action-execution time to eliminate options that would not have adequate time to be executed successfully. This multiagent behavior represents a bridge between low-level and high-level learning in the Layered Learning paradigm. The newly created behavior is tested empirically in game situations.

1 Introduction

Multiagent Systems is the subfield of AI that aims to provide both principles for construction of complex systems involving multiple agents and mechanisms for coordination of independent agents' behaviors. As of yet, there has been little work with Multiagent Systems that require real-time control in noisy, adversarial environments. Because of the inherent complexity of this type of Multiagent System, Machine Learning is an interesting and promising area to merge with Multiagent Systems. Machine learning has the potential to provide robust mechanisms that leverage upon experience to equip agents with a large spectrum of behaviors, ranging from effective individual performance in a team, to collaborative achievement of independently and jointly set high-level goals. Especially in domains that include independently designed agents with conflicting goals (*adversaries*), learning may allow agents to adapt to unforeseen behaviors on the parts of other agents.

Layered Learning is an approach to complex multiagent domains that involves incorporating low-level learned behaviors into higher-level behaviors [18]. Using simulated Robotic Soccer (see Section 2) as an example of such a domain, a Neural Network (NN) was used to learn a low-level individual behavior (ball-

interception), which was then incorporated into a basic collaborative behavior (passing). The collaborative behavior was learned via a Decision Tree (DT) [13].

This paper extends these basic learned behaviors into a full multiagent behavior that is capable of controlling agents throughout an entire game. This behavior makes control decisions based on the confidence factors associated with DT classifications—a novel approach. While operator success probabilities have previously been stored in tree structures [3], our work makes use of confidence measures originally derived from learning classification tasks. It also makes use of the ability to reason about action-execution time to eliminate options that would not have adequate time to be executed successfully. The newly created behavior is tested empirically in game situations.

The rest of the paper is organized as follows. Section 2 gives an overview of foundational work in the Robotic Soccer domain. The new behavior, along with explanations of how the DT is used for control and how the agents reason about action-execution time, is presented in Section 3. Extensive empirical results are reported in Section 4, and Section 5 concludes.

2 Foundational Work

This section presents brief overviews of Robotic Soccer research and of Layered Learning. Further details with regards to both topics can be found in [13].

2.1 Robotic Soccer

As described in [5], Robotic Soccer is an exciting AI domain for many reasons. The fast-paced nature of the domain necessitates real-time sensing coupled with quick behaving and decision making. Furthermore, the behaviors and decision-making processes can range from the most simple reactive behaviors, such as moving directly towards the ball, to arbitrarily complex reasoning procedures that take into account the actions and perceived strategies of teammates and opponents. Opportunities, and indeed demands, for innovative and novel techniques abound.

Robotic Soccer systems have been recently developed both in simulation [6, 9, 12, 14] and with real robots [1, 4, 10, 11, 15, 13]. While robotic systems are difficult, expensive, and time-consuming to use, they provide a certain degree of realism that is never possible in simulation. On the other hand, simulators allow researchers to isolate key issues, implement complex behaviors, and run many trials in a short amount of time. While much of the past research has used Machine Learning in constrained situations, nobody has yet developed a full behavior based on learning techniques that can be used successfully in a game situation.

The Soccer Server [7], which serves as the substrate system for the research reported in this paper, captures enough real-world complexities to be a very challenging domain. This simulator is realistic in many ways: (i) the players' vision is limited; (ii) the players can communicate by posting to a blackboard

that is visible (but not necessarily intelligible) to all players; (iii) each player is controlled by a separate process; (iv) each team has 11 members; (v) players have limited stamina; (vi) actuators and sensors are noisy; (vii) dynamics and kinematics are modelled; and (viii) play occurs in *real time*: the agents must react to their sensory inputs at roughly the same speed as human or robotic soccer players. The simulator, acting as a server, provides a domain and supports users who wish to build their own agents (clients).

2.2 Layered Learning

Layered Learning is a Multiagent Learning paradigm designed to allow agents to learn to work together in a real-time, noisy environment in the presence of both teammates and adversaries. Layered Learning allows for a bottom-up definition of agent capabilities at different levels in a complete multiagent domain. Machine Learning opportunities are identified when hand-coding solutions are too complex to generate. Individual and collaborative behaviors in the presence of adversaries are organized, learned, and combined in a layered fashion (see Figure 1).

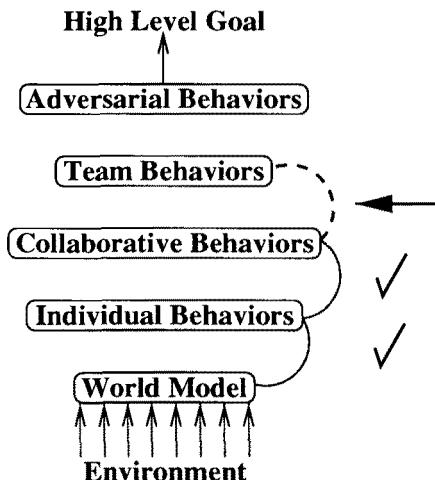


Fig. 1. An overview of the Layered Learning framework. It is designed for use in domains that are too complex to learn a mapping straight from sensors to actuators. We use a hierarchical, bottom-up approach. Two low-level behaviors have been previously learned. The work reported in this paper creates a team behavior that facilitates higher-level learning.

To date, two levels of learned behaviors have been implemented [13]. First, Soccer Server clients used a Neural Network (NN) to learn a low-level individual skill: how to intercept a moving ball. Then, using this learned skill, they learned

a higher-level, more “social,” skill: one that involves multiple players. The second skill, the ability to estimate the likelihood that a pass to a particular teammate will succeed, was learned using a Decision Tree (DT). The DT was trained using C4.5 [8] under the assumption that the player receiving the ball uses the trained NN when trying to receive the pass. This technique of incorporating one learned behavior as part of another is an important component of Layered Learning. As a further example, the output of the decision tree could be used as the input to a higher-level learning module, for instance a reinforcement learning module, to learn whether or not to pass, and to whom.

The successful combination of the learned NN and DT demonstrated the feasibility of the Layered Learning technique. However, the combined behavior was trained and tested in a limited, artificial situation which does not reflect the full range of game situations. In particular, a passer in a fixed position was trained to identify whether a particular teammate could successfully receive a pass. Both the teammate and several opponents were randomly placed within a restricted range. They then used the trained NN to try to receive the pass.

Although the trained DT was empirically successful in the limited situation, it was unclear whether it would generalize to the broader class of game situations. The work reported in this paper incorporates the same trained DT into a complete behavior using which players decide when to chase the ball, and after reaching the ball, what to do with it.

First, a player moves to the ball—using the NN—when it does not perceive any teammates who are likely to reach it more quickly. Then, using a pre-defined communication protocol, the player probes its teammates for possible pass receivers (collaborators). When a player is going to use the DT to estimate the likelihood of a pass succeeding, it alerts the teammate that the pass is coming, and the teammate, in turn, sends some data reflecting its view of the world back to the passer. The DT algorithm used is C4.5 [8], which automatically returns confidence factors along with classifications. These confidence factors are useful when incorporating the DT into a higher level behavior capable of controlling a client in game situations.

3 Using the Learned Behaviors

As described in Section 2, ML techniques have been studied in the Soccer Server in isolated situations. However, the resulting behaviors have not yet been tested in full game situations. In this paper, we examine the effectiveness in game situations of the DT learned in [13].

To our knowledge, this paper reports the first use of confidence factors from DTs for agent control. In particular, the confidence factors that are returned along with classifications can be used to differentiate precisely among several options.

3.1 Receiver Choice Functions

Recall that the DT estimates the likelihood that a pass to a specific player will succeed. Thus, for a client to use the DT in a game, several additional aspects of its behavior must be defined. First, the DT must be incorporated into a full *Receiver Choice Function* (RCF). We define the RCF to be the function that determines what the client should do *when it has possession of the ball*: when the ball is within kicking distance (2m). The input of an RCF is the client's perception of the current state of the world. This perceived state includes both the agent's latest sensory perception and remembered past positions of currently unseen objects [2]. The output of an RCF is an action from among the options *dribble*, *kick*, or *pass*, and a direction, either in terms of a player (i.e. towards teammate number 4) or in terms of a part of the field (i.e. towards the goal). Consequently, before using the DT, the RCF must choose a set of candidate receivers. Then, using the output of the DT for each of these receivers, the RCF can choose its receiver or else decide to dribble or kick the ball. Table 1 defines three RCFs, one of which uses the DT, and the others defined for the purposes of comparison.

1. Each player has a set of receivers that it considers, as indicated in Figure 2. The set of candidates is determined by the player's actual location on the field, rather than its assigned position.
2. Any potential receiver that is too close or too far away (arbitrarily chosen—but constant—bounds) is eliminated from consideration.
3. Any player that is out of position (because it was chasing the ball) is eliminated from consideration.
4. **IF** there is an opponent nearby (arbitrarily chosen—but constant—bound) **THEN** any potential receiver that cannot be passed to immediately (the passer would have to circle around the ball first) is eliminated from consideration.
5. **IF** one or more potential receivers remain **THEN** pass to the receiver as determined by the *Receiver Choice Function* (RCF):

PRW (*Prefer Right Wing*): Use a fixed ordering on the options. Players in the center prefer passing to the right wing over the left.

RAND (*Random*): Choose randomly among the options.

DT (*Decision Tree*): Pass to the receiver to which the trained decision tree (see Section 2) assigns the highest **success confidence**. If no confidence is high enough, kick or dribble as indicated below.

6. **ELSE** (*No potential receivers remain*)
 - **IF** there is an opponent nearby, **THEN** kick the ball forward;
 - **ELSE** dribble the ball forward.

Table 1. Specification of the RCFs.

As indicated in Table 1, the set of candidate receivers is determined by the players' *positions*. Each player is assigned a particular position on the field, or

an area to which it goes by default. The approximate locations of these positions are indicated by the locations of the players on the black team in Figure 2. The

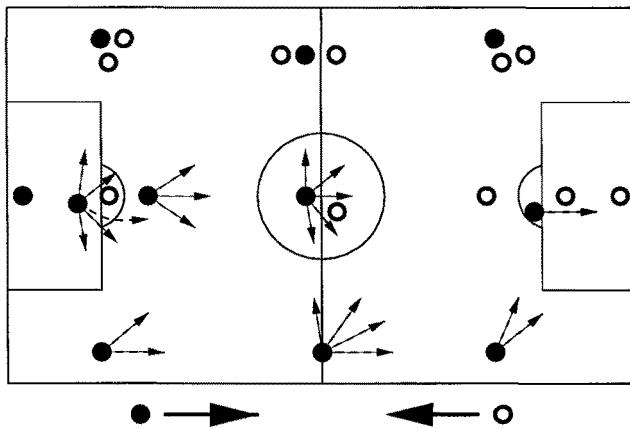


Fig. 2. Player positions used by the behaviors in this paper. The black team, moving from left to right, has a goalie, a sweeper, and one defender, midfielder, and forward on the left, center, and right of the field. The arrows emanating from the players indicate the positions to which each player considers passing when using the RCFs. The players on the left of the field (top of the diagram) consider symmetrical options to their counterparts on the right of the field. The goalie has the same options as the sweeper. The white team has the same positions as the black, except that it has no players on its left side of the field, but rather two in each position on its right.

formation used by all of the tested functions includes—from the back (left)—a goalie, a sweeper, three defenders, three midfielders, and three forwards. When a player is near its default position, it periodically announces its position to teammates; when a player leaves its position to chase the ball, it announces this fact and is no longer considered “in position” (see Table 1, Step 3). The arrows emanating from the players in Figure 2 indicate the *positions* to which each player considers passing. The clients determine which players are in which positions by listening to their teammates’ announcements.

The RCFs defined and used by this paper are laid out in Table 1. As suggested by its name, the DT—*Decision Tree*—RCF uses the DT described in Section 2 to choose from among the candidate receivers. In particular, as long as one of the receivers’ success confidences is positive, the DT RCF indicates that the passer should pass to the receiver with the highest success confidence, breaking ties randomly. If no receiver has a positive success confidence, the player with the ball should dribble or kick the ball forwards (towards the opponent goal or towards one of the forward corners). This use of the DT confidence factor is, to our knowledge, a novel approach to agent control. The RAND—*Random*—RCF is the same as the DT RCF except that it chooses randomly from among the candidate receivers.

The PRW—*Prefer Right Wing*—RCF uses a fixed ordering on the candidate receivers for each of the positions on the field. In general, defenders prefer to pass to the wings rather than forward, midfielders prefer to pass forward rather than sideways, and forwards tend to shoot. As indicated by the name, all players in the center of the field prefer passing to the right rather than passing to the left. The RCF simply returns the most preferable candidate receiver according to this fixed ordering. Again, if no receivers are eligible, the RCF returns “dribble” or “kick.” This RCF was the initial hand-coded behavior for use in games.

3.2 Reasoning about action execution time

An important thing to notice in the RCF definition (Table 1) is that the clients can reason about how long they predict they have to act. In particular, if there is an opponent nearby, there is a danger of losing the ball before being able to pass or shoot it. In this situation, it is to the passer’s advantage to get rid of the ball as quickly as possible.

This priority is manifested in the RCFs in two ways: (i) in Step 4 of Table 1, teammates to whom the client cannot pass immediately are eliminated from consideration; and (ii) in Step 6, the client kicks the ball away (or shoots) rather than dribbling. When a player is between the ball and the teammate to which it wants to pass, it must move out of the ball’s path before passing. Since this action takes time, an opponent often has the opportunity to get to the ball before it can be successfully released. Thus, in Step 4, when there is an opponent *nearby* the RCFs only consider passing to players to whom the client can pass immediately. The concept of *nearby* could be the learned class of positions from which the opponent could steal the ball. For the purposes of this paper, “within 10m” is an empirically acceptable approximation. As mentioned above, this concept is not purely reactive: the positions of opponents that are outside an agent’s field of view are remembered [2].

Similarly, the point of dribbling the ball (kicking the ball a small amount in a certain direction and staying with it) is to keep the ball for a little longer until a good pass becomes available or until the player is in a good position to shoot. However, if there is an opponent nearby, dribbling often allows the opponent time to get to the ball. In this situation, as indicated in Step 6 of Table 1, the player should kick the ball forward (or shoot) rather than dribbling.

The ability to reason about how much time is available for action is an important component of the RCFs and contributes significantly to their success in game situations (see Section 4).

3.3 Incorporating the RCF in a behavior

In Section 3.1, the method of using a DT as a part of an RCF is described in detail. However, the RCF is itself not a complete client behavior: it only applies when the ball is within kicking distance. This section situates the RCFs within a complete behavior that can then be used throughout the course of a game. The player’s first priority is always to find the ball’s location (only objects in front of

-
1. **IF** the client doesn't know where the ball is, **THEN** turn until finding it.
 2. **IF** the ball is more than 10m away **AND** a teammate is closer to the ball than the client is, **THEN**:
 - **IF** the ball is coming towards the client, **THEN** watch the ball;
 - **ELSE** Move randomly near client position.
 3. **ELSE:** (*client is the closest to the ball, or the ball is within 10m*)
 - **IF** The ball is too far away to kick (> 2m), **THEN** move to the ball, using the trained Neural Network when appropriate;
 - **ELSE** Pass, Dribble, or Kick the ball as indicated by the *Receiver Choice Function* (RCF).
-

Table 2. The complete behavior used by the clients in game situations.

the player are seen). If it doesn't know where the ball is, it turns until the ball is in view. When turning away from the ball, it remembers the ball's location for a short amount of time; however after about three seconds, if it hasn't seen the ball, it assumes that it no longer knows where the ball is [2].

Once the ball has been located, the client can execute its behavior. As described in Section 3.1, each player is assigned a particular position on the field. Unless chasing the ball, the client goes to its position, moving around randomly within a small range of the position. The player represents its position as x, y coordinates on the field.

The client chases the ball whenever it thinks that it is the closest team-member to the ball. Notice that it may not *actually* be the closest player to the ball if some of its teammates are too far away to see, and if they have not announced their positions recently. However, if a player mistakenly thinks that it is the closest player, it will get part of the way to the ball, notice that another teammate is closer, and then turn back to its position. When the ball is within a certain small range (arbitrarily 10m), the client always goes towards the ball. When the ball is moving towards the client or when a teammate has indicated an intention to pass in its direction, the client watches the ball to see if either of the two above conditions is met. A player that was chasing the ball is predisposed to continue chasing the ball. Only if it finds that it should turn back persistently for several steps does it actually turn back. As required for use of the DT, every player is equipped with the trained Neural Network (see Section 2) which can be used to help intercept the ball.

Finally, every team member uses the same RCF. Whenever the ball is within kicking distance, the client calls its RCF to decide whether to dribble, kick, or pass, and to where. The behavior incorporating the RCFs is laid out in Table 2.

4 Experiments

In this section we present the results of empirically testing how the complete behavior performs when using the different RCF options. Since the behaviors differ only in their RCFs, we refer below to, for example, “the complete behavior with the DT RCF” simply as “the DT RCF.” Also presented are empirical results verifying the advantage of reasoning about action-execution time.

In order to test the different RCFs, we created a team formation that emphasizes the advantage of passing to some teammates over others. When both teams use the standard formation (that of the black team in Figure 2), every player is covered by one opponent. However, this situation is an artificial artifact of having the same person program both teams. Ideally, the players would have the ability to move to open positions on the field. However at this point, such functionality represents future work (see Section 5). Instead, in order to reflect the fact that some players are typically more open than others, we tested the RCFs against the OPR—*Only Play Right*—formation which is illustrated by the white team in Figure 2. We also used the symmetrical OPL—*Only Play Left*—formation for testing. These behaviors are specified in Table 3.

-
- The opponent behaviors are exactly the same as the RAND behavior except that the players are assigned to different positions:
- OPR** (*Only Play Right*): As illustrated by the white team in Figure 2, two players are at each position on the right side of the field, with no players on the left side of the field.
- OPL** (*Only Play Left*): Same as above, except all the players are on the left side of the field.
-

Table 3. OPR and OPL behavior specification.

During testing, each run consists of 34 five-minute games between a pair of teams. We tabulate the cumulative score both in total goals and in games won (ties are not broken) as in Table 4. Graphs record the *difference* in cumulative goals scored (Figure 3) and games won (Figure 4) as the run progresses.

In order to test the effectiveness of the DT RCF, we compared its performance against the performance of the PRW and RAND RCFs when facing the same opponent: OPR. While the DT and RAND RCFs are symmetrical in their decision making, the PRW RCF gives preference to one side of the field and therefore has an advantage against the OPR strategy. Thus we also include the results of the PRW RCF when it faces the symmetrical opponent: OPL. From the table and the graphs, it is apparent that the DT RCF is an effective method of decision making in this domain.

In order to test the effectiveness of the reasoning about action-execution time, we compared the performance of the standard DT RCF against that of the same RCF with the assumption that there is *never* an opponent nearby:

RCF (vs. OPR)	Games (W – L)	Overall Score
DT	19 – 9	135 – 97
PRW	11 – 14	104 – 105
PRW (vs. OPL)	8 – 16	114 – 128
RAND	14 – 12	115 – 111

Table 4. Results are cumulative over 34 five-minute games: ties are not broken. Unless otherwise indicated, the opponent—whose score always appears second—uses the OPR formation.

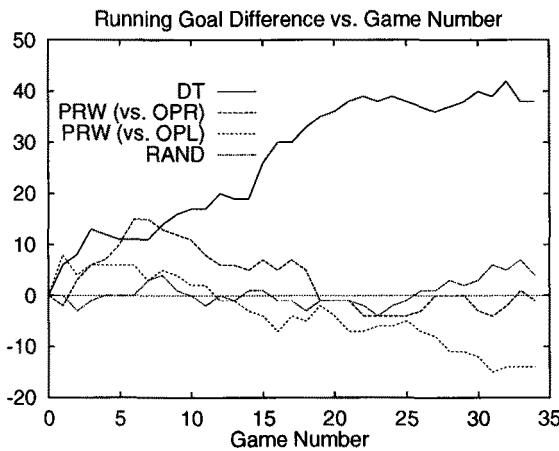


Fig. 3. The differences in cumulative goals as the runs progress.

even if there is, the RCF ignores it. This assumption affects Steps 4 and 6 of the RCF specification in Table 1. Both RCFs are played against the OPR behavior. As apparent from Table 5, the reasoning about action-execution time makes a significant difference.

RCF (vs. OPR)	Games (W – L)	Overall Score
Standard DT	19 – 9	135 – 97
No-rush DT	13 – 16	91 – 108

Table 5. No-rush DT is the same RCF as the standard DT except that there is no reasoning about action-execution time. The Standard DT RCF performs significantly better.

We expect that the DT RCF, including the reasoning about action-execution

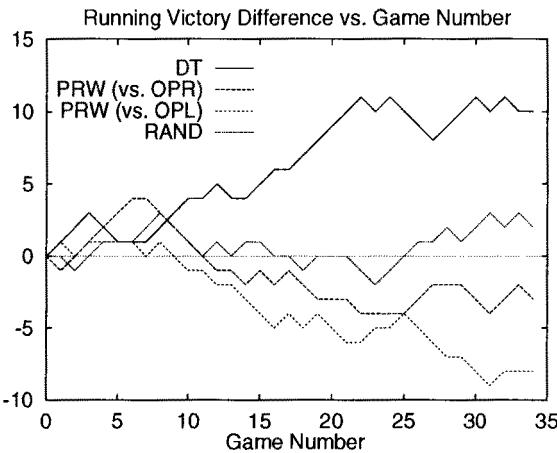


Fig. 4. The differences in cumulative games won as the runs progress.

time, will perform favorably against teams that cover our players unevenly so that the DT can find an open player to whom to pass. Indeed, it was used successfully as part of the CMUnited simulator team at RoboCup-97 [5] which was held at IJCAI-97. In a field of 29 teams, CMUnited made it to the semi-finals, before losing to the eventual champions¹.

5 Discussion and Conclusion

The experiments reported Section 4 indicate that the confidence factors provided by standard DT software can be used for effective agent control. Combined with some basic reasoning about the action-execution times of different options—necessitated by the real-time nature of this domain, the DT-based control function outperformed both random and hand-coded alternatives. Even though the DT was trained in a limited artificial situation, it was useful for agent control in a broader scenario.

Throughout this paper, the multiagent behaviors are tested against an opponent that leaves one side of the field free, while covering the other side heavily. This opponent simulates a situation in which the players without the ball make an effort to move to an open position on the field. Such collaborative reasoning has not yet been implemented in the Soccer Server. However, the fact that the DT is able to exploit open players indicates that reasoning about field positioning when a teammate has the ball would be a useful next step in the development of learned collaborative behaviors.

Along with more variable field positioning, there is still a great deal of future work to be done in this domain. First, one could build additional learned

¹ Full tournament results are available at <http://www.robocup.org/RoboCup>.

layers on top of the NN and DT layers described in Section 2. The behavior used in this paper *uses* the DT as a part of a hand-coded high-level multiagent behavior. However, several parameters are arbitrarily chosen. A behavior that *learns* how to map the classifications and confidence factors of the DT to passing/dribbling/shooting decisions may perform better. Second, on-line adversarial learning methods that can adapt to opponent behaviors during the course of a game may be more successful against a broad range of opponents than current methods.

Nevertheless, the incorporation of low-level learning modules into a full multiagent behavior that can be used in game situations is a significant advance towards intelligent multiagent behaviors in a complex real-time domain. Furthermore, the ability to reason about the amount of time available to act is essential in domains with continuously changing state. Finally, as DT confidence factors are effective tools in this domain, they are a new potentially useful tool for agent control in general. These contributions promise an exciting future for learning-based methods in real-time, adversarial, multiagent domains.

Acknowledgements

This research is sponsored in part by the Defense Advanced Research Projects Agency (DARPA), and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-95-1-0018 and in part by the Department of the Navy, Office of Naval Research under contract number N00014-95-1-0591. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the Air Force, of the Department of the Navy, Office of Naval Research or the United States Government.

References

1. Minoru Asada, Eiji Uchibe, Shioichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Coordination of multiple behaviors acquired by vision-based reinforcement learning. In *Proc. of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems 1994 (IROS '94)*, pages 917–924, 1994.
2. Mike Bowling, Peter Stone, and Manuela Veloso. Predictive memory for an inaccessible environment. In *Proceedings of the IROS-96 Workshop on RoboCup*, November 1996.
3. Andrew Garland and Richard Alterman. Multiagent learning through collective memory. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pages 33–38, Menlo Park, CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.
4. Jong-Hwan Kim, editor. *Proceedings of the Micro-Robot World Cup Soccer Tournament*, Taejon, Korea, November 1996.

5. Hiroaki Kitano, Yasuo Kuniyoshi, Itsuki Noda, Minoru Asada, Hitoshi Matsubara, and Eiichi Osawa. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.
6. Hitoshi Matsubara, Itsuki Noda, and Kazuo Hiraki. Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pages 63–67, Menlo Park, CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.
7. Itsuki Noda and Hitoshi Matsubara. Soccer server and researches on multi-agent systems. In *Proceedings of the IROS-96 Workshop on RoboCup*, November 1996.
8. J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
9. Michael K. Sahota. Dynasim user guide. <http://www.cs.ubc.ca/nest/lci/soccer>, January 1996.
10. Michael K. Sahota, Alan K. Mackworth, Rod A. Barman, and Stewart J. Kingdon. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3690–3663, 1995.
11. Randy Sargent, Bill Bailey, Carl Witty, and Anne Wright. Dynamic object capture using fast vision tracking. *AI Magazine*, 18(1):65–72, Spring 1997.
12. Peter Stone and Manuela Veloso. Beating a defender in robotic soccer: Memory-based learning of a continuous function. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 896–902, Cambridge, MA, 1996. MIT press.
13. Peter Stone and Manuela Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *To appear in Applied AI Journal*, 1998.
14. Peter Stone and Manuela Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. *in International Journal of Human-Computer Systems (IJHCS)*, 48, 1998.
15. Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim. CMUnited: A team of robotic soccer agents collaborating in an adversarial environment. In H. Kitano, editor, *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Springer Verlag, Berlin, 1998, in this volume.

A Role-Based Decision-Mechanism for Teams of Reactive and Coordinating Agents

Silvia Coradeschi and Lars Karlsson

Department of Computer and Information Science

Linköping University, Sweden

silco@ida.liu.se, larka@ida.liu.se

Abstract. In this paper we present a system for developing autonomous agents that combine reactivity to an uncertain and rapidly changing environment with commitment to prespecified tactics involving coordination and team work. The concept of roles in a team is central to the paper. Agents act and coordinate with other agents depending on their roles. The decision-mechanism of an agent is specified in terms of prioritized rules organized in a decision tree. In the decision-tree both reactivity to changes in the environment and commitment to long term courses of actions (behaviors) are present. The coordination between agents is mainly obtained through common tactics, strategies, and observations of actions of team members, rather than explicit communication. Coordinated behaviors can be specified in terms of roles and are encoded simultaneously for all participating agents.

1 Introduction

An important issue in the development of multi-agent systems is how to create a decision mechanism for an agent that combines the ability to react to a dynamically changing environment and to coordinate with other agents. A special kind of coordination is the one required in performing team work. The importance to have the ability to operate in teams has been emphasized by several authors, see for example [11] and [2]. In a team, an agent needs special capabilities to coordinate with other agents in the team. In particular an agent acts and coordinates mostly depending on his roles in the team. The concept of role is therefore crucial in the development of agents working in teams. Roles have been considered in the context of the Belief, Desire, and Intentions Multi-Agent System [6], but mainly in the perspective of team formation and changes of roles in a team, [13] and [1]. In the case of modeling team and team tactics [14] just very simple cases have been examined (two types of roles statically assigned to the agent during the team formation). Our interest is in creating a general role-based decision-mechanism for multiple automated agents that act in complex and real-time environments in which team work is required.

Our multi-agent system has initially been developed in the context of a cooperation project with Saab Military Aircraft with the aim to investigate the design and implementation of automated agents for air combat simulation [4]. In particular the aim was to develop means for specifying the behavior for automated pilots. The goal was to combine on one side reactivity to the uncertain and dynamic battlefield environment and

on the other side commitment to prespecified tactics involving coordination and team work. The behavior specification was not to be done by experts in computer science and AI, but mainly by experts in air combat and military air technology, and therefore, the system was aimed to be relatively easy to use and conceptually simple.

The system has proven to be easy to apply in other domains. We have been able to use it without any important modification in the soccer RoboCup simulated domain [3] and we intend to use some of the ideas of the system in a project for the design of an autonomous aircraft for traffic surveillance. In the following we use mainly examples from the soccer domain as most people have a better knowledge in this domain than in the air combat domain. Also in the TacAir-Soar project [12] the development of intelligent agents for the air combat domain is now coupled with the development of intelligent agents for the soccer domain. Special consideration has been given to team work, but the interest has been more centered in the tracking of teams [10] and commitment inside the team [11] than in the specification of agents through roles.

In both the soccer domain and the air combat domain, coordination between agents is essential, but it cannot be achieved by mainly communication and negotiation [8]. In fact, giving the real-time constraints in these domains, the time for communication is limited and the presence of hostile agents makes the communication risky and unreliable (jamming). Several techniques have been developed for communication-poor coordination including a physics oriented approach to coordination [9], focal points [5] and observation-based coordination [7]. In general these techniques aim at coordinating isolated agents performing rather simple tasks e.g. filling holes of various sizes in a planar surface, trying to choose the same object out of a set of objects and moving in the same location or in an opposite location with respect to another agent in a discrete time, two dimensional, simulated grid world. In our case the agents are organized in teams, have roles in the team and perform complex and in general not repetitive tasks.

What the agents do and how they coordinate with other agents is mostly determined by their roles in the team. An agent can have multiple roles, for example he can be captain of the team and midfielder. Roles can be specialized, for example a left-midfielder is a specialization of a midfielder. The decision-mechanism is specified in term of roles and the agents will execute the behaviors and actions that are proper to their roles. Specifying coordinated behavior requires to take in consideration all the roles involved in the coordination, but then for each agent the importance of the coordinated behaviors should be evaluated in the context of the roles of that agent. Therefore coordinated behavior is specified in decision-trees for multiple roles, section 6.1, and then the parts of the behavior related to a role are collected, added to the specification of the role and reconsidered in the context of the role. The coordination is then obtained with a totally distributed control.

2 Overview of the system

The basic element of the decision-mechanism in our system is the concept of *behavior module* (behavior for short). A behavior specifies what actions and subbehaviors should be executed and under what conditions to fulfill some specific aims or purposes. It can either be constantly present or it can be started under certain circumstances, go on

for a period of time and then terminate. Behaviors have different complexities; they are used for specifying coordination and the behavior proper of a role, but also for specifying simple routine as for example the positioning in the penalty area of a soccer player. Behaviors can recursively contain behaviors. The same structure is therefore repeated at several levels of the decision mechanism. A behavior is characterized by a priority, which indicates the importance of the behavior, and a decision-tree in which is coded what the behavior does. A decision-tree is a tree of hierarchically structured and prioritized decision rules. Visiting the decision-tree several actions and behaviors can be selected as candidates to be performed. The selection of those that will actually execute is done based on priority values and compatibility. Behaviors and actions are compatible if they can be executed concurrently. The priority values can change dynamically as different circumstances can alter the importance of actions and behaviors during the simulation.

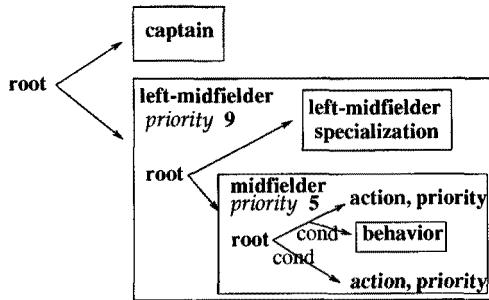


Fig. 1. Decision-tree of a left-midfielder that is also captain of the team

Each agent is equipped with a state describing the information that he uses for making decisions, and a decision-tree that specifies the general behavior of the agent (section 3). This decision-tree is the result of the composition of the behavior modules that specify the roles of the agent. The composition of behavior modules consists of creating a decision-tree with a module in each leaf and supplying a priority value for each behavior module. Conditions required to enter the behavior module can be specified, for example a corner-kick shooter behavior is entered if a corner kick is called. In Fig. 1 the decision-tree of a left-midfielder that is also captain of the team is sketched. The left-midfielder role is a specialization of the midfielder role and it is specified as a composition of a behavior module for the midfielder role and a behavior module for the left-midfielder specialization.

Coordinated behaviors are specified through decision-trees for multiple roles, section 6.1. These decision-trees have the same structure as the decision-tree of an agent, but each action and condition is associated with a specific role.

3 The decision-mechanism

A decision-tree consists of a hierarchy of decisions with one decision for each node. Two kinds of conditions can be associated with a node: conditions for entering the node and modifier conditions that change the priorities of actions and behaviors. Modifier conditions have the form (*condition* → *number*). If the condition is true, the number is added to the priority of the actions or behaviors associated with the branches. The conditions are tested in the state of the agent. The state contains the characteristics of the agent specified before the start of the simulation, the information the agent receives directly from the simulator and derived information. In order of not being purely reactive, an agent should keep track of what has happened in the past. One part of the state represents therefore the "memory" of the agent. In the memory are recorded, among other things, important past events, decisions previously taken, and messages received via communication channels. Conditions can be primitive and composite. Composite conditions are formed by applying the operators *and3*, *or3*, *not3*, *for-all* and *there-is* to other composite or primitive conditions. *And3*, *or3*, *not3* are operators in a three-valued logic: true, false and unknown. The value of a condition in fact can be unknown, for example the agent may not know his relative position with respect to the ball.

At the leaves of the decision-tree there are actions and behaviors with a basic priority value. At each step of the simulation the state is updated with information that is received from the simulator and interpreted. Then the decision-tree is visited and as a result a list of actions and a list of behaviors are created. The actions in the list are the actions that the agent will consider to perform. To this list, the actions still in progress are added. The agent then selects the actions to actually execute and then sends them to the simulator or to update the state. Some of the actions in the list are mutually exclusive as they use the same resource and in this case the agent selects those he will actually perform depending on their priority values. Some actions can be done concurrently and in this case the agent performs them in parallel. The test to check if two actions are compatible is based on the physical resources they use. Two actions are compatible if they do not use the same resources. For example the action of passing is compatible with the action of sending a message. Operators for sequential and concurrent composition of actions are also defined. If the agent decides to perform sequences of actions he is not committed to complete the sequence of actions as the sequence can be interrupted if something else becomes more important.

Behaviors are selected for execution with a mechanism similar to the one for actions and the result of the selection is recorded in the state. The behaviors continue executing until they have fulfilled their aim, they have failed or they are interrupted to give space for something more important. If a behavior is executing then the tree inside the behavior is visited and the actions are selected in the same way as we have explained earlier in this section. The actions selected are then added to the list of actions of the decision-tree that directly contains the behavior. The selection mechanism is recursively applied until the actions of the main tree are selected and executed.

Let us consider a simple example of decision-tree (Fig. 2). This is part of the tree of an attacker. One branch leads to the action of going near the opponent the attacker is marking, in case the distance to the opponent is more than 30. The other branch is entered when a midfielder has the ball, *there-is x midfielder (has-ball x)*). If the attacker

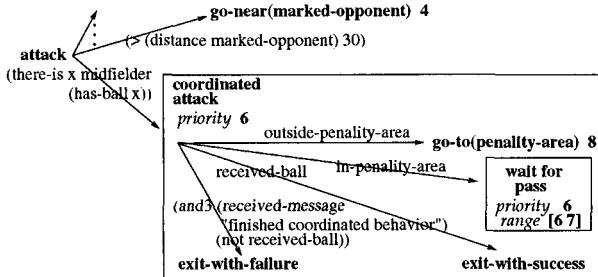


Fig. 2. Part of the decision-tree of an attacker

is outside the penalty area he goes to the penalty area, and when he is inside the penalty he waits for a pass. The decision-tree of the **wait for pass** behavior has been omitted due to lack of space. The behavior **coordinated-attack** is exited with success if the attacker receives the ball; with failure if a message from the midfielder is received that the coordinated attack is finished and the agent has not received the ball.

The conditions on the branches can refer to both information obtained from perception and information stored in the agent's internal state. An example of the latter is when a player follows an opponent due to a commitment to mark this opponent. This commitment is registered in the player's state. Making the commitment to mark a particular opponent constitute an internal action that update the state of the agent. The external actions presented in the decision tree are interpreted in terms of the low-level actions available in the simulator. For instance, **go-to(penalty-area)** makes the agent turn and move towards the penalty area. Also the information received from the simulator is interpreted. For instance, at each simulation cycle information about the ball and other players is received by the agent and using this information a list of team mates that are nearer than him to the ball is created. This permits the user to work on a level of abstraction that is more meaningful and convenient than working with the low-level data and actions of the simulator. He/she can focus more on what to do than the details of how to do it.

4 Behavior modules

Behavior modules play an important role in the system. They can be used to specify agents' roles and coordination, but also simple routines as **wait-for-pass** and in general, long term course of actions. An example of a behavior is **coordinated attack** in Fig.2. An important aspect that emerge in the use of behaviors is the need for modularity, that is the internal specification of the behavior should not be visible in the rest of the decision-tree of which it is part. In fact, roles and coordinated behaviors can be specified independently of the decision-tree of which they will be part. Coordinated behaviors can be specified for instance through decision-trees for multiple roles (see section 6.1). Also simple routines should be reusable in different decision-trees. On the other hand

the actions selected in a behavior and specially their priorities have influence in the rest of the tree and in other behaviors. We consider these issues in the subsection *Priorities in behaviors*.

Complex behaviors can contain parts that should be executed in parallel or in sequence. In analogy with sequential and concurrent actions, sequential and concurrent behaviors are defined in the last part of this section.

Priorities in behaviors There are several possibilities in deciding the priorities of the actions selected in a behavior. The simpler method is to assign to the selected actions the priority of the behavior. This gives modularity in the assignment of priorities as the priorities inside the behavior are not visible externally. The problem with this approach is that not all the actions in a behavior have the same importance and a non-essential action in one behavior can be preferred over an essential action in another behavior and make this last behavior terminate. What we would like is that the priority that the action has inside the behavior would influence which priority the action will have when competing with the actions selected in the decision-tree containing the behavior. At the same time we would like to have an external control of the priorities of the actions coming from a behavior in order to prioritize correctly one behavior over another and to have more freedom in deciding the priorities inside a behavior. We define together with the priority of the behavior also a range inside which the priority of the actions selected in the behavior can vary. Given the priority that the selected action has inside the behavior, its external priority is the priority of the behavior increased or decreased in proportion to the internal priority of the action and the range in which the priority is allowed to vary. In the example in Fig.2 the action **go-to(penalty-area)** has internally priority 8 and externally priority 7. The action **go-near(x)**, where x is the opponent that the attacker is marking, has priority 4 and the range [5 7] of the coordinated attack behavior guarantees that the actions in the coordinated attack will always have higher priority than the **go-near(x)** action. This solution in most cases prevents that actions with lower priorities interfere with higher priority behaviors. However, it still may happen that actions external to a behavior interfere with the behavior, even if the behavior has higher priority than the actions. Let us consider for example the **wait-for-pass** behavior. In this behavior the agent mainly turns in the direction of the player with the ball to spot as soon as possible if the player is passing to him. Between one turning action and another, no actions are selected in the behavior. Therefore the system can decide to perform the action **go-near(x)**. That is of course not what we want. A solution we intend to consider is to allow a behavior to reserve a resource, in this case the moving resource. The resource can be released only if it is an action that has higher priority of the behavior that claims the resource.

Interruption and termination of behaviors Behaviors can terminate, successfully or unsuccessfully, or can be interrupted. When a behavior is selected the decision-tree inside the behavior is visited and the actions selected compete with the other actions selected in the visit of the decision-tree containing the behavior. Each behavior has exiting conditions that are used to exit the behavior when it is successfully terminated and when the behavior cannot be continued and terminates with failure. In the example in

Fig.2, the behavior is successfully terminated if the attacker receives the ball. A behavior can be interrupted if an incompatible behavior with higher priority is selected or if an action selected in the behavior cannot be executed as it is incompatible with another selected action with higher priority. In this last case the behavior can be exited or not depending on whether the action is essential for the behavior. If an action essential for the behavior cannot be performed, recovery actions to be performed before exiting the behavior can be specified. The criterion used for checking the compatibility of actions, looking at the resources they use, cannot be transferred directly to the case of behaviors. The compatibility of behaviors is in fact more conceptual than due to physical limitations. For example a coordinated attack from the left is incompatible with a coordinated attack from the right, but defending a part of the field by a defender can be compatible with making the team advance passing to the midfielders. The relation of compatibility between behaviors is decided when all the behaviors are created. When a new behavior is reached during the visit of the decision-tree, it competes with the other behaviors selected during the visit and with the behaviors currently executing. The compatible behaviors with higher priority will be the ones executing in the next step of the simulation. Exiting actions can be specified for the interrupted behaviors.

Sequential and concurrent behaviors Behaviors can be sequentially and concurrently composed and these composite behaviors can be selected in a decision-tree like single behaviors. In both concurrent and sequential behaviors the interruption or the termination with failure of one or more behaviors represents a problem. In the system, if a behavior is interrupted or unsuccessfully terminated the sequence of behaviors is aborted and all the concurrent behaviors are terminated. However, there can be cases in which such a drastic solution is not necessary. To give more flexibility to the system we intend to give the possibility to specify in terms of conditions → actions what to do in case of failure or termination of behaviors.

5 Roles and construction of decision-trees for roles

Roles give an essential contribution to efficiency and effectiveness of teams. In absence of roles in a soccer team all players would tend, for example, to get the ball, leaving their penalty area unguarded, or would try to defend the penalty area without leaving some players in the central part of the field ready for a counterattack. With roles, the resources of the team are distributed depending on the need and can be utilized in an effective way. Furthermore, the coordination is facilitated by the presence of roles both with respect to specification of coordination and to recognition of coordinated tactics. In fact knowing the role of an agent gives an indication of the capabilities of the agent and of the kind of coordinated behavior he is expected to perform. For example an attacker has already a specification of tactics for scoring. Therefore, this part does not need to be considered in the specification of a coordinated attack. Also, knowing that the player that has the ball is a left-midfielder gives an indication to an attacker that wants to be coordinated with him, on what direction the attack will take. In general the roles of the agent are maintained during the entire execution, but in some special cases the roles of

the agent can also change during the execution. For example, if the team is losing, one midfielder can become an attacker.

An important aspect of roles is the inheritance relation between roles. For example, a left-midfielder is also a midfielder and a midfielder is also a player. This gives the possibility of easily specializing roles without having to redefine the parts common to several roles. Also an agent can have several roles in a team, that can be active in the same time as captain and midfielder, or that can alternate depending on the circumstances as midfielder and corner-kick shooter. There are therefore two kinds of composition of roles, one for specializing preexistent roles and one for combination of roles in the same agent. Examples of both cases can be found in Fig. 1. The problem in both cases is how to compose roles that are in general independently specified, without having to modify manually one or both the roles in order to deal with possible conflicts of actions and behaviors between roles. For example, a midfielder could try to pursue the ball in whatever position in the field the ball is, but a left-midfielder would be more inclined to pursue the ball if it is in his zone. One simple solution would be to just prioritize one role over the other, but in this case not very important actions in one role could prevent the execution of important actions in another role. The composition of roles is therefore done in term of composition of behaviors. In behaviors, as explained in section 4, a basic priority establishes the importance of the behavior, but the priority that a selected action has in this behavior, influences the priority that this action has compared with actions selected in other behaviors. With this priority assignment the actions selected in one behavior get a priority proportional to the importance that they have in the behavior. The specialization of roles is done by composition of the behavior module of the old role, and the behavior module containing the specializing part of the role. Priority and priority range of the behaviors is then decided.

The roles and the priority and priority range that the behaviors containing the roles will have, are decided when the agent is defined. The agent then executes the decision-trees created as a combination of his roles. In case an agent changes a role during the execution, he starts executing the decision-tree of the new role and stops executing the decision-tree of the previous role. As roles can have subroles in common (midfielder and attacker have for instance the subrole “player” in common), when an agent changes role, the roles’ relations are examined and just the behaviors that are in subroles not common with the new behavior are exited.

6 Coordination among agents

In both the soccer and the air combat domain team work is essential, but the possibility of communication, due to real time constraints and possibility of interception, is limited. Coordination is therefore obtained by common tactics and strategies learned and practised during training. Each agent knows the tactics of the team and his role in the tactics and tries to act according to this knowledge. The agent should initiate tactics depending on the situation and recognize the tactics that the other team members have initiated in order to do his part in it. The agent has three ways of recognizing the tactics that team mates are applying: observation of the general situation and knowledge of the tactics appropriate for the situation, observation of indicative actions of the team mates, and

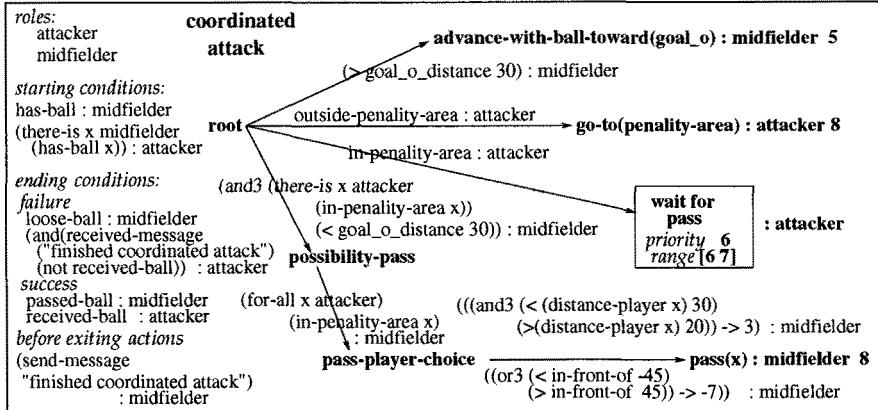


Fig. 3. Example of decision-tree for multiple agents

explicit communication. In general there can be a combination of these three aspects.

In our approach the knowledge required for applying the tactics is coded in behavior modules and the observations of the situation and of the actions of other agents influence the decisions about which tactics to adopt. Explicit communication is performed through communication actions present in the decision tree of the module. The message is stored in the state of the agent and will influence his future decisions.

6.1 Decision-tree for multiple roles

The specification of coordinated behavior is facilitated if the behavior of all the roles involved can be specified in the same structure. For this reason we introduce decision-trees in which the behaviors of a number of roles are specified. Every agent that has the appropriate role can then apply the behavior. In the decision-tree for coordinated behavior all actions and conditions are associated with roles.

Let us consider an example of a decision-tree for multiple agents. The behavior we want to specify is a simple case of coordinated attack between an attacker and a midfielder, Fig. 3. The coordinated behavior can be started if the midfielder has the ball and it is terminated when the midfielder loses or passes the ball. The midfielder advances with the ball toward the opponents' goal (**advance-with-ball-toward(goal_o)**) and when he is near the opponents' goal and an attacker is in the penalty area (**(and3 (there-is x attacker (in-penalty-area x)) (< (goal_o_distance)30))**) he considers the possibility of passing the ball to an attacker among those that are in the penalty area (**(for-all x attacker (in-penalty-area x))**). The preference to which player to pass is based on the distance of the player, and on the distance to the opponent goal (if a player is further than him from the goal the priority of passing to him is decreased by 7). The attacker, if he is outside the penalty area goes to the penalty area, and when he is inside the penalty area waits for a pass.

Performing a coordinated behavior involves a commitment towards the other agents participating in the behavior. Therefore it can be necessary to perform some actions before exiting the behavior. In our example before exiting the coordinated behavior the midfielder sends a message to advise that the coordinated attack is finished. The receiving of this message is in the ending condition of the coordinated behavior for the attacker.

The coordinated decision-trees are used in the specification of coordinated behavior, but then the parts of the trees regarding each role are transferred to the decision-trees of the role. The decision-trees of the roles are the ones that are actually executed. The branches regarding each role in a coordinated decision-tree are collected in a behavior module and this module is then added to the decision-tree of the role. Fig. 2 shows the module extracted from the coordinated attack decision-tree and how it is inserted in the decision-tree of an attacker. The figure has been explained already in section 3. We can here just point out that the starting condition of the coordinated attack is the condition in the branch for entering the behavior and the ending conditions are the condition for performing the action **exit-with-failure** and **exit-with-success**. A priority and priority range are assigned to the behavior in the role decision-tree as they depend on the importance that the behavior has in the context of the decision-tree.

7 Summary

We have described a decision mechanism for autonomous agents which attempts to provide a relatively simple way to specify coordination and team work in autonomous agents. A key concept in the paper is the concept of a role. Agents act and coordinate with other agents mostly depending on their roles. Roles can be specialized and several roles can be combined in the same agent through composition of behaviors. Behavior modules represent an important aspect of the system. They are used to specify coordination and the behavior proper of a role, simple routines and in general long term courses of action. The decision-mechanism is based on a decision-tree where actions and behaviors are assigned dynamically changing priorities and decisions are based both on sensory input and the internal state of the agent. Coordination between agents is mainly obtained through common tactics and strategies and observations of actions of team members. Explicit communication is also possible, but due to real time constraints and the possibility of interception by the opponents should be reduced to short and simple messages. Coordinated behaviors are specified in terms of roles and are encoded in decision-trees for multiple roles.

The system has been partially implemented and tested. It could unfortunately not be tested in the context of the RoboCup competition in IJCAI97. We are now working in completing the implementation and in further testing.

8 Acknowledgments

We would like to thank D. Driankov, P. Lambrix and T. Vidal for their helpful comments. Supported by the Wallenberg Foundation project “Information Technology for Autonomous Aircraft”.

References

1. S. Ch'ng and L. Padgham. Role organisation and planning team strategies. In *Proc. of the 20th Australian Computer Science Conference*. Sydney, Australia, 1997.
2. P.R. Cohen and H.J. Levesque. Teamwork. In *Nous*, number 35. 1991.
3. S. Coradeschi and L. Karlsson. A decision-mechanism for reactive and cooperating soccer-playing agent. In *Linköping Electronic Articles in Computer and Information Science*, volume 2. 1997. <http://www.ep.liu.se/ea/cis/1997/001/>. Presented at "RoboCup Workshop: Soccer as a problem for Multi-Agent Systems" in the Second International Conference on Multi-Agent Systems (ICMAS-96), Kyoto, Japan.
4. S. Coradeschi, L. Karlsson, and A. Törne. Intelligent agents for aircraft combat simulation. In *Proc. of the 6th Conf. on Computer Generated Forces and Behavioral Representation*. Orlando, FL, 1996.
5. M. Fenster, S. Kraus, and J. S. Rosenschein. Coordination without communication: Experimental validation of focal point and techniques. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*. San Francisco, CA, 1995.
6. M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proc. of IJCAI'89*. Detroit, Michigan, 1989.
7. M. J. Huber and E. H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*. San Francisco, CA, 1995.
8. J. E. Laird, R. M. Jones, and P. E. Nielsen. Coordinated behavior of computer generated forces in TacAir-Soar. In *Proc. of the 4th Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL, 1994.
9. O. Shehory and S. Kraus. Cooperative goal-satisfaction without communication in large-scale agent-systems. In *Proceeding of ECAI'96*. Budapest, Hungary, 1996.
10. M. Tambe. Tracking dynamic team activity. In *Proceedings of AAAI'96*. Portland, Oregon, 1995.
11. M. Tambe. Teamwork in real-world, dynamic environments. In *Proc. of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*. Kyoto, Japan, 1996.
12. M. Tambe, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 1995.
13. G. Tidhar, A. S. Rao, and E. A. Sonenberg. Guided team selection. In *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96)*. Kyoto, Japan, 1996.
14. G. Tidhar, M. Selvestrel, and C. Heinze. Modeling teams and team tactics in whole air mission modelling. In *Proceedings of the 8th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*. Gordon and Breach Science Publishers, Melbourne, Australia, 1995.

Using an Explicit Model of Teamwork in RoboCup

Milind Tambe, Jafar Adibi, Yaser Al-Onaizan, Ali Erdem
Gal A. Kaminka, Stacy C. Marsella, Ion Muslea, Marcello Tallis

Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90292, USA
robocup-sim@isi.edu
www.isi.edu/soar/tambe/socteam.html

Abstract. Team ISIS (**ISI Synthetic**) successfully participated in the first international RoboCup soccer tournament (RoboCup'97) held in Nagoya, Japan, in August 1997. ISIS won the third-place prize in over 30 teams that participated in the simulation league of RoboCup'97 (the most popular among the three RoboCup'97 leagues). In terms of research accomplishments, ISIS illustrated the usefulness of an explicit model of teamwork both in terms of reduced development time and improved teamwork flexibility. ISIS also took some initial steps towards learning of individual player skills. This paper discusses the design of ISIS in detail, with particular emphasis on its novel approach to teamwork.

1 Introduction

The ISIS (**ISI Synthetic**) team of synthetic soccer-players won the third-place prize in the RoboCup'97 simulation league tournament. Developed at the University of Southern California's Information Sciences Institute (ISI), ISIS was also the top US simulation team. In terms of research accomplishments, ISIS illustrated the reuse of STEAM, a general model of teamwork[16], that both reduced its development time and improved teamwork flexibility.

ISIS's development is driven by the three research challenges emphasized in the RoboCup simulation league: (i) teamwork; (ii) multi-agent learning; and (iii) agent- and team-modeling[6]. With respect to teamwork, one key novelty in ISIS is its use of STEAM, a general, explicit model of teamwork to enable teamwork among player agents. This general model is motivated by the need for flexibility in team activities, as well as reuse of teamwork capabilities across domains[14, 15, 16]. STEAM uses the formal *joint intentions* framework[1, 7] as its basic building block, but with key enhancements to reflect the constraints of real-world domains. STEAM requires that individual team members explicitly represent their team's goals, plans and mutual beliefs. It then enables team members to autonomously reason about coordination and communication in

teamwork, providing improved flexibility. Indeed, all of the current communication among ISIS agents is driven by STEAM's general purpose reasoning about teamwork. Given its domain-independence, STEAM also enables reuse across domains — here, RoboCup provided a challenging test domain, given its substantial dissimilarity from the original domain of STEAM's application (pilot teams for combat simulations for military training[16, 17]). Yet, a promising 35% of the original STEAM code was reused in RoboCup, and no new general-purpose teamwork code was required.

With respect to multi-agent learning, the second challenge in RoboCup, ISIS took some initial steps towards addressing it. Using C4.5[10], ISIS players learned off-line to choose an intelligent kicking direction, avoiding areas of concentration of opponent players. With respect to the third challenge, ISIS also performed limited agent- and team-modeling (particularly relevant to teamwork), but detailed plan-recognition of opponent-team's strategies remains an open issue for future work.

The rest of this paper is organized as follows: Section 2 describes the architecture of an individual ISIS agent. Section 3 describes the teamwork capability in ISIS. Section 4 discusses C4.5-based learning in ISIS. Section 5 then provides a summary and topics for future work. We will assume that the reader is familiar with Soccer, the RoboCup simulation league rules, as well as the RoboCup simulator[5].

2 ISIS Individual Agent Architecture

An ISIS agent is developed as a two-level architecture. The lower level, developed in C, communicates inputs received from the RoboCup simulator (after sufficient pre-processing), to the higher level. The lower level also rapidly computes some recommended directions for turning and kicking, to be sent to the higher-level. For instance, it computes three possible directions to kick the ball: (i) a group of C4.5 rules compute a direction to kick the ball towards the opponents' goal while avoiding areas of concentration of opponents; (ii) a hand-coded routine computes kicking direction to clear the ball; (iii) a second hand-coded routine computes direction to kick the ball directly into the center of the opponent's goal (without taking opponents' location into account). The lower-level also computes a direction to turn if a player is to intercept an approaching ball.

The lower level does not make any decisions with respect to its recommendations however. For example, it does not decide which one of its three suggested kicking directions should actually be used by a player-agent. Instead, all such decision-making rests with the higher level, implemented in the Soar integrated AI architecture[9, 11]. Once the Soar-based higher-level reaches a decision, it communicates with the lower-level, which then sends the relevant information to the simulator.

The Soar architecture involves dynamic execution of an operator (reactive plan) hierarchy. These operators consist of (i) precondition rules; (ii) application rules; and (iii) termination rules. Precondition rules help select operators for

execution based on the agent's current high-level goals/tasks and beliefs about its environment. Selecting high-level abstract operators for execution leads to subgoals, where new operators are selected for execution, and thus a hierarchical expansion of operators ensues. Activated operators are executed by the application rules. If the agent's current beliefs match an operator's termination rules, then the operator terminates. Agents built in other architectures such as PRS[4], BB1[3], RAP[2] for dynamic domains may be similarly characterized in this fashion.

The operator hierarchy shown in Figure 1 illustrates a portion of the operator hierarchy for ISIS player-agents in RoboCup. One key novelty in this hierarchy, to support STEAM's teamwork reasoning (discussed below), is the inclusion of *team operators* (reactive team plans). Team operators explicitly express a team's joint activities, unlike the regular "individual operators" which express an agent's own activities. In the hierarchy in Figure 1, operators shown in boxes such as **WIN-GAME** are team operators, while others are individual operators. The key here is that when executing team operators, agents bring to bear STEAM's teamwork reasoning, which facilitates their communication and coordination.

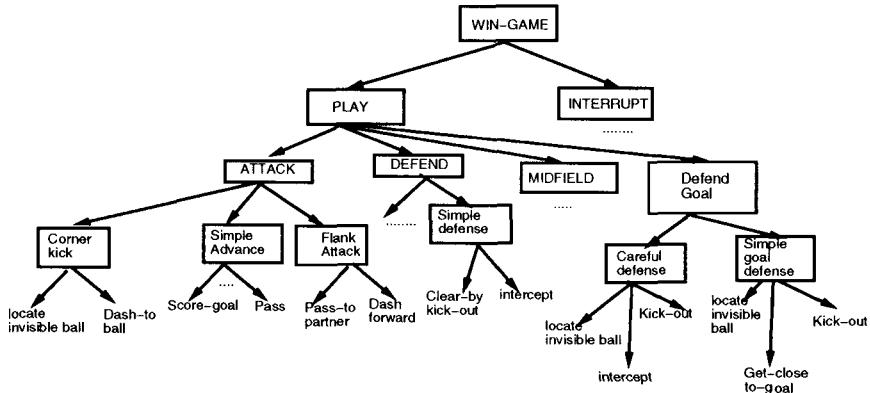


Fig. 1. A portion of the operator hierarchy for player-agents in RoboCup soccer simulation. Boxed operators are team operators, others are individual operators.

As with individual operators, team operators also consist of: (i) precondition rules; (ii) application rules; and (iii) termination rules. However, while an individual operator applies to an agent's private state (an agent's private beliefs), a team operator applies to an agent's *team state*. A team state is the agent's (abstract) model of the team's mutual beliefs about the world, e.g., the team's currently mutually believed strategy. The team state is usually initialized with information about the team, such as the team members in the team, possible subteams, available communication channels for the team, the pre-determined team leader and so forth. STEAM can also maintain subteam states for subteam participation. There is of course no shared memory, and thus each team member

maintains its own copy of the team state, and any subteam states for subteams it participates in. To preserve the consistency of a (sub)team state, one key restriction is imposed for modifications to it — only the team operators executed by that (sub)team can modify it.

In Figure 1, the highest-level operator is a team operator called WIN-GAME. When all of the player agents select WIN-GAME in their operator hierarchy, the WIN-GAME team operator is established. There are two possible operators that can be executed in service of WIN-GAME, specifically PLAY (when the ball is in play) or INTERRUPT (when the ball is not in play). When the team operator PLAY is active, one of four team operators, ATTACK, MIDFIELD, DEFEND and DEFEND-GOAL can be executed. Each of these four is executed by a different subteam. Thus, the subteam of *forwards* in ISIS, typically consisting of three players, executes the ATTACK team operator. In service of ATTACK, the subteam may execute FLANK-ATTACK or SIMPLE-ADVANCE. In service of these, one of several individual operators, such as “score-goal” may be executed. Meanwhile, a second subteam of three agents may execute the DEFEND team operator. At any one time, an ISIS player agent has only one path through this hierarchy that is active, i.e., executed by the agent.

3 Teamwork in ISIS

As mentioned earlier, teamwork in ISIS is driven by a general, explicit model of teamwork called STEAM. STEAM uses the joint intentions theory as the basic building block of teamwork, and hence this theory is briefly discussed in Section 3.1. STEAM’s communication and coordination activities, driven by this theory, are discussed in Section 3.2. STEAM was originally developed in the context of building teams of helicopter pilot-agents for real-world military simulations[16]. Originally developed within Soar, STEAM is currently encoded in the form of 283 Soar rules. RoboCup has provided a challenging domain for testing reuse of these STEAM rules, in a substantially dissimilar domain. Currently, about 35% of the rules are reused in ISIS, and this reuse may likely increase in the future.

3.1 Joint Intentions

STEAM’s general model of teamwork is based on the *joint intentions* theory[7]. A joint intention of a team Θ is based on its joint commitment, which is defined as a joint persistent goal (JPG). A JPG to achieve a team action p , denoted $JPG(\Theta, p)$ requires all teammembers to mutually believe that p is currently false and want p to be eventually true.

JPG provides a basic change in plan expressiveness, since it focuses on a team task. Furthermore, a JPG guarantees that team members cannot decommit until p is mutually known to be achieved, unachievable or irrelevant. Basically, $JPG(\Theta, p)$ requires team members to each hold p as a weak achievement goal (WAG).¹ $WAG(\mu, p, \Theta)$, where μ is a team member in Θ , requires μ to achieve

¹ WAG was originally called WG in [7], but later termed WAG in [12].

p if it is false. However, if μ privately believes that **p** is either achieved, unachievable or irrelevant, $\text{JPG}(\Theta, \mathbf{p})$ is dissolved, but μ is left with a commitment to have this belief become Θ 's mutual belief. Such a commitment helps to avoid communication failures — to establish mutual belief, an agent must typically communicate with its teammates.

Members of Θ must synchronize to establish $\text{JPG}(\Theta, \mathbf{p})$. To achieve such *team synchronization* we adapt the *request-confirm* protocol[12], described below. The key here is a persistent weak achievement goal ($\text{PWAG}(\nu_i, \mathbf{p}, \Theta)$), which commits a team member ν_i to its team task **p** prior to a JPG . μ initiates the protocol while its teammates in Θ , $\nu_1, \dots, \nu_{i-1}, \nu_{i+1}, \dots, \nu_n$, respond:

1. μ executes a **Request**(μ, Θ, \mathbf{p}), cast as an **Attempt**(μ, ϕ, ψ). That is, μ 's ultimate goal ϕ is to both achieve **p**, and have all ν_i adopt $\text{PWAG}(\nu_i, \mathbf{p}, \Theta)$. However, μ is minimally committed to ψ , i.e., just to achieve mutual belief in Θ that μ has the PWAG to achieve ϕ . With this **Request**, μ adopts the PWAG.
2. Each ν_i responds via **confirm** or **refuse**. **Confirm**, also an **Attempt**, informs others that ν_i has the PWAG to achieve **p**.
3. If $\forall i, \nu_i$ confirm, $\text{JPG}(\Theta, \mathbf{p})$ is formed. b

Besides synchronization, this protocol enforces important behavioral constraints. In step 1, the adoption of a PWAG implies that if after requesting, μ privately believes that **p** is achieved, unachievable or irrelevant, it must inform its teammates. Furthermore, if μ believes that the minimal commitment ψ is not achieved (e.g., the message did not get through) it must retransmit the message. Step 2 similarly constrains team members ν_i to inform others about **p**, and to rebroadcast. If everyone confirms, a JPG is established.

Thus, communication arises in the joint intentions theory to establish joint intentions, and to terminate them. However, communication in service of establishing and termination of each and every joint intention can be highly inefficient[16]. Hence, STEAM includes decision-theoretic communication selectivity. In particular, agents explicitly reason about the costs and benefits of communication, e.g., they avoid costly communication if there is a high likelihood that the relevant information can be obtained by other teammates via observation.

3.2 Joint Intentions in ISIS

Joint intentions are operationalized in STEAM via team operators. In particular, when all team members select a team operator such as WIN-GAME for execution (see Figure 1), they establish a joint intention. As participants in such a joint intention, STEAM enables individual team members to reason about their coordination and communication responsibilities.

Thus, based on the joint intentions theory, an individual cannot arbitrarily terminate a team operator on its own. Instead, a team operator can only be terminated if there is mutual belief that the operator is achieved, unachievable or irrelevant. Establishing such mutual belief in the termination of a team operator can lead to communication. In particular, communication on termination of team operator arises if an agent privately realizes some fact relevant to the termination

of a current team operator. Thus, if an agent's private state contains a belief that terminates a team operator (because it is achieved, unachievable or irrelevant), and such a belief is absent in its team state, then it creates a communicative goal, i.e., a communication operator. The generation of this communication operator is regulated based on its costs and benefits. When executed, this operator leads the agent to broadcast the information to the team.

Indeed, all communication in ISIS agents is currently driven by STEAM's general-purpose teamwork reasoning. A typical example of such communication is seen when three players in the "goalie" subteam execute the DEFEND-GOAL team operator. In service of DEFEND-GOAL, players in this subteam normally execute the SIMPLE-DEFENSE team operator to position themselves properly on the field and to try to be aware of the ball position. Of course, each player can only see in its limited cone of vision, and particularly while repositioning itself, can be unaware of the approaching ball. Here is where teamwork can be beneficial. In particular, if any one of these players sees the ball as being close, it declares the SIMPLE-DEFENSE team operator to be irrelevant. Its teammates now focus on defending the goal in a coordinated manner via the CAREFUL-DEFENSE team operator. Should any one player in the goalie subteam see the ball move sufficiently far away, it again alerts its team mates (that CAREFUL-DEFENSE is irrelevant). The subteam players once again execute SIMPLE-DEFENSE to attempt to position themselves close to the goal. In this way, agents attempt to coordinate their defense of the goal, while also attempting to position themselves near it.

4 Learning

Inspired by previous work on machine learning in RoboCup[13, 8], we focused on techniques to improve individual players' skills to kick, pass, or intercept the ball. Fortunately, the two layer ISIS architecture helps to simplify the problem for skill learning. In particular, the lower-level in ISIS is designed to provide several recommendations (such as several alternative kicking directions) to the higher-level, but it need not arrive at a specific decision (one specific kicking direction). Thus, an individual skill, such as a kicking direction to clear the ball, can be learned independent other possible actions. That is, the learning algorithm is not forced to simultaneously learn to select if clearing the ball is the best choice among available alternatives. Instead, that decision is left to the higher-level.

For the RoboCup'97 tournament, C4.5[10] was successfully used in ISIS to learn to select an *intelligent* kicking direction. C4.5 rules were learned off-line via a batch of training examples to select a direction to kick towards the opponent's goal while avoiding areas of concentration of opponent players. This learned kicking direction was one among three kicking directions computed in the lower-level (as discussed in Section 2). The higher-level typically selected the learned kicking direction (from the three provided to it) when players were close to the goal, and were ready to directly score a goal. While we did initial exploration

to learn the remaining two kicking directions, those results were not ready for RoboCup'97. We hope to field a team with further learned skills for RoboCup'98.

5 Summary

The overall goal in ISIS was not and has not just been one of building a team that wins the RoboCup tournaments. Rather, ISIS has taken a principled approach, guided by the research opportunities in RoboCup. Despite the significant risk in following such a principled approach, ISIS won the third place in over 30 teams that participated in the RoboCup'97 simulation tournament.

There are several key issues that remain open for future work. One key issue is improved agent- or team-modeling. One immediate application of such modeling is recognition that an individual, particularly a team member, is unable to fulfill its role in the team activity. For instance, if a forward is "covered" by the opponents, it may be unable to fulfill its role. In such cases, STEAM enables agents to reason about taking over others' roles, e.g., enabling a midfielder to take over the role of a non-performing forward. However, currently, in the absence of the required agent modeling capability, STEAM cannot engage in such reasoning. Indeed, this is partly the reason that many STEAM rules have currently not applied in RoboCup (so that STEAM reuse is limited to 35% of rules).

A second key issue arose as a lesson learned from ISIS's participation in RoboCup'97. A weakness was discovered in ISIS, that stemmed from a somewhat inappropriate interaction with the RoboCup simulator — the simulator version used in RoboCup'97 allowed agents to take up to three actions (one action per 100 ms) before sending them a sensor update (one update per 300 ms). This required that agents continually make predictions. Unfortunately, with weak predictive capabilities, ISIS agents could not always quickly locate and intercept the ball, or maintain awareness of positions of teammates and opponents. This key weakness was a factor in the single loss that ISIS suffered in the course of the RoboCup'97 tournament. However, the RoboCup simulator will evolve for RoboCup'98, towards more human-like play.

Overall, we hope to continue working on ISIS in preparation for RoboCup'98, and meet the research challenges outlined for the simulation league in teamwork, multi-agent learning and agent modeling[6]. Further information about ISIS, including the code, is available at the following web site:

www.isi.edu/soar/tambe/socteam.html.

STEAM code, with detailed documentation and traces is available at:

www.isi.edu/soar/tambe/steam/steam.html

ISIS team members can be reached at robocup-sim@isi.edu.

Acknowledgement

We thank Bill Swartout, Paul Rosenbloom and Yigal Arens of USC/ISI for their support of the RoboCup activities described in this paper. We also thank Peter Stone and Manuela Veloso for providing us player-agents of CMUnited, which provided a good opponent team to practice against in the weeks leading up to RoboCup'97.

References

1. P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 35, 1991.
2. J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1987.
3. B. Hayes-Roth, L. Brownston, and R. V. Gen. Multiagent collaboration in directed improvisation. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*, 1995.
4. F. F. Ingrand, M. P. Georgeff, , and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE EXPERT*, 7(6), 1992.
5. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*, 1995.
6. H. Kitano, M. Tambe, P. Stone, S. Coradeschi, H. Matsubara, M. Veloso, I. Noda, E. Osawa, and M. Asada. The robocup synthetic agents' challenge. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, August 1997.
7. H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press, 1990.
8. H. Matsubara, I. Noda, and K. Hiraki. Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer. In S. Sen, editor, *AAAI Spring Symposium on Adaptation, Coevolution and Learning in multi-agent systems*, March 1996.
9. A. Newell. *Unified Theories of Cognition*. Harvard Univ. Press, Cambridge, Mass., 1990.
10. J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.
11. P. S. Rosenbloom, J. E. Laird, A. Newell, , and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.
12. I. Smith and P. Cohen. Towards semantics for an agent communication language based on speech acts. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1996.
13. P. Stone and M. Veloso. Towards collaborative and adversarial learning: a case study in robotic soccer. In S. Sen, editor, *AAAI Spring Symposium on Adaptation, Coevolution and Learning in multi-agent systems*, March 1996.
14. M. Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*, December 1996.
15. M. Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, August 1997.

16. M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research (JAIR)*, 7:83–124, 1997.
17. M. Tambe, W. L. Johnson, R. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), Spring 1995.

Decision Making by the Characteristics and the Interaction in Multi-agent Robotics Soccer

Akihiro MATSUMOTO, Hiromasa NAGAI

Dept. of Mechanical Engineering, Toyo University

2100 Kujirai, Kawagoe, Saitama 350, JAPAN

E-mail: {akihiro, nagai}@robot.eng.toyo.ac.jp

Abstract. This paper deals with a decentralized decision-making method on multi-agent robotic system. Soccer game is adopted as an example for its dynamic characteristics. The soccer simulator has been designed and implemented which enables us to test various approaches to the multi-agent studies. The modeling of capabilities of a single robot and team strategy is explained at first. Each robotic agent has different characteristics, and the decision making by each robotic agents are done by considering its characteristics and interaction between robotic agents. Several simulation experiments with different combinations of the components and formations have been tested. The result of the simulation experiments was that similar combination of the components resulted in better performance, and the formation where robotic agents are located in the center axis between the goals showed better defense behaviors.

1 Introduction

The authors have been engaged in the research on the coordination and the cooperation of the group of autonomous robots by the decentralized approach. Although decentralized system is expected to be much more applicable in the future manufacturing system or other multi-agent system, centralized approach is still considered to be effective in current situation. One of the reasons for this is that there are little good methods for evaluating decentralized system. Another reason is that decentralized controlling system and decentralized decision-making system are often confused.

In such an decentralized robot system, robots should be autonomous enough to dynamically organize robot groups depending on the situation without any supervisor in the system. The related research are [1], [2], [3], [4], for example, where communication network is assumed and indispensable (*e.g.*, [5]).

Then the authors have concentrated in the game of soccer (association football) where each player behaves by its own decision for the victory of the team, and each player with different capabilities selects different strategy in real-time for offense and defense, depending on the situation. From this viewpoint, soccer game is an interesting example of the decentralized decision making system. For this target, our research group is constructing plural mobile robots with several sensors and wireless communication interface, and on the other hand,

implementing soccer simulator on the workstation [8], [9]. Currently the implementation of this system is done apart from *RoboCup* (e.g., [6]). Our research is independently done from this project, but the research domain is very similar.

In this paper, the strategy of selecting cooperative behavior in offense team and the assessment method of evaluating such behavior is described. This approach is the first step for evaluating decentralized decision-making system.

2 Modeling of the Characteristics of Robot Players

The modeling of the behavior of a single robot and that of plural robot is explained in [8]. For the ease of understanding, let us explain them here again.

2.1 Assumptions

Following assumptions have been made to implement robotics soccer on the workstation.

- The members of the teams are fixed until either team gets a goal.
- The shape of the robot is modeled by a circle with the radius 20 cm, and kinematically modeled as a differential-drive mobile robot.
- The size of the field is 12 meters by 9 meters, which is 1/10 of the maximum size of the real field, and there are invisible wall on the goal lines and touch lines, *i.e.* the ball do not go outside of the field).
- Each robot has a sensor, actuators, communication interface, and a ball handling mechanism as well as its brain.
- Robots can distinguish the ball and other robots by its own sensors. The range of the sensing area of the robot is expressed by a half-circle.
- If the ball is within a certain distance from one robot player, it is considered that the ball is owned by the robot. The ball is grabbed by opponent player if the ball is beyond that distance during dribble.
- The capabilities of robots in the same team are known to the teammates beforehand. The capabilities of robots in the opponent team are unknown.
- The position of robot itself is measured by itself by dead-reckoning, and does not consider positional errors.
- The positions of the teammates are obtained by communication interface through the data base manager. The positions of the opponents are measured by sensors.

2.2 Characteristics and the Behavior Rules of a Single Robot

Several types of characteristics are defined in this research. Although there are much more types of characteristics in real soccer, three types are assumed as shown in Table 1 in order to minimize complexity and the computing cost. This idea also accepts some multi-agent research that assumes homogeneous agents in the decentralized/distributed system. In table 1, type 1 moves faster than

others and kicks strongly, but the sensing distance is shorter than others. On the other hand, type 3 moves slower and kicks less strongly, but the sensing distance is longer. Type 2 is the middle of type 1 and type 3. In this paper, type 1 is called Forward-type (F-type), type 2 is Mid-fielder-type (M-type), and type 3 is Defender-type (D-type).

Table 1. The capabilities of robots

max. value	type 1	type 2	type 3
velocity [m/s]	0.5	0.4	0.3
acceleration [m/s^2]	0.1	0.2	0.3
omega [deg/s]	80.0	90.0	100.0
sensing distance [m]	1.5	2.0	2.5
kick velocity [m/s]	5.0	4.0	3.0

Each robot player has an instinct to search the ball or opponent players until it is found within its sensing area, and also to avoid obstacles. In order to search the ball or other opponent players, the robot player wanders for a certain direction which is generated by a random number. If it finds the ball within its sensing area, it tries to dribble to the opponent goal until it can shoot. Of course, this is not always possible; in such situation, it selects to continue dribble or to pass to others by the rules defined as follows: Three behaviors, (1) dribble, (2) shoot, (3) pass, are defined in this research. Two behaviors (dribble and shoot) are considered to be the independent plays, whereas pass is considered to be the cooperative play that requires other robot players. The algorithm of pass is explained later.

3 Interaction Between Agents

3.1 Necessity of Selecting Behaviors

In multi-agent robotics, the cooperation algorithm including conflict resolution of decentralized decision-making is important. In robotics soccer, cooperative plays are, for example,

- (offense) pass to the players who is suitable for shooting,
- (offense) support motion to receive the pass,
- (defense) obstruction of shooting by positioning along the line between the opponent and defense goal,
- (defense) obstruction of passing by positioning along the line between the opponent and its teammate.

The sender of pass is always the owner of the ball, but the receiver is not fixed during the game. This relationship is dynamically configured in soccer

games; although a robot who do not own the ball moves to receive the pass, that behavior could be in vain. In this sense, the cooperators do not always receive an expected result; the conflict of decisions are always resolved by the owner of the ball. In this report, pass is made only between two robots for the moment, but for the future, a kind of combination play where more than three robots are involved should be modeled.

3.2 Behavior Rules for Robot Group

Plural robots get together and organize one team of robotics soccer. The algorithm of team play is also embedded in each robot player as modeled previously. Pass is considered to be the cooperative behavior. Pass is sent based on the relative position of the robot players in the field and the capability of the robot players. The receiver of the ball is dynamically selected depending on the situation. The decision is done in the following sequences:

[offense - the robot player who owns the ball]

1. First it decides to continue dribble or to pass by comparing the capability factor of itself and the opponent who is within its sensing area.
2. If the opponent is less stronger, then continue dribble. If otherwise, it takes following procedure.
 - (a) try to pass to the robot player who located nearer to the goal of the opponent
 - (b) if the distances are the same, try to pass to the robot player who is better at shooting (i.e. F-type)

[offense - the robot players who do not have the ball]

It simply moves to the certain distance from the goal line of the opponent, and then tries to receive the pass.

[defense]

It simply moves to the certain distance from the goal line of its goal, and then searches the opponent players to interfere. The relationship of these behaviors are illustrated in Figure 1.

3.3 Decision Algorithm to Pass or Dribble

The algorithms of selecting the independent behaviors and cooperative behaviors as described here. These formulations are based on the evaluation which is based on the calculations of the distances to other robots in same team and the distances to the goal, and the comparison of the capabilities with other robots. There are two steps for this decision; one is for selecting dribble or pass, and the other is for selecting to which robot the ball should be passed.

The decision algorithm of selecting dribble or pass is based on the following idea. As mentioned above, if the opponent appears within the sensing area of the robot who owns the ball, the robot compares the capability of itself and that of the opponent. Since the capability of the opponent cannot be known as mentioned in the previous chapter, the average value of all types of robots are

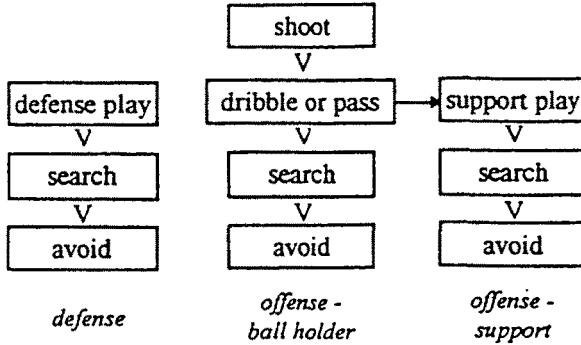


Fig. 1. The relationship of behaviors of plural robots

substituted by the capability of the opponent (This must be refined for future such that some learning algorithm should be included). If the capability of the robot who owns the ball is bigger than that of the average, it continues to dribble. It is expressed by an index *my_spec* as follows:

$$my_spec = \alpha \frac{vel_{my_max}}{vel_{ave}} \cdot \beta \frac{acc_{my_max}}{acc_{ave}} \cdot \gamma \frac{\omega_{my_max}}{\omega_{ave}} \cdot \delta \frac{kick_{my_max}}{kick_{ave}} \quad (1)$$

where *vel_{my_max}* is the maximum velocity of the robot who owns the ball, *vel_{ave}* is the average velocity of the robots, *ω_{my_max}* is a maximum rotation velocity of the robot who owns the ball, *ω_{ave}* is the average rotation velocity of the robots, *acc_{my_max}* is the maximum acceleration of the robot who owns the ball, *acc_{ave}* is the average acceleration of the robots, *kick_{my_max}* is the maximum kicking capability of the robot who owns the ball, *kick_{ave}* is the average kicking capability of the robots, $\alpha, \beta, \gamma, \delta$ are weight coefficients (currently set to 2.0, 1.0, 1.0, 1.0, respectively, by heuristics).

Next, potential function [7] has been introduced. By using the distance *r_{min}* between the nearest opponent and the robot who owns the ball is calculated. The potential field *U_{opp}* is calculated by using the weight coefficient *k_{opp}* such as

$$U_{opp} = k_{opp} \frac{1}{r_{min}^2} \quad (2)$$

The potential field *U_{opp}* is compared to the threat or pressure from the opponents in the real soccer. The capability factor *my_spec* is converted to potential function form *U_{self}* by using weight coefficient *k_{self}*, such as

$$U_{self} = k_{self} \frac{my_spec}{r_{min}^2} \quad (3)$$

Then the index *E_{pd}* for selecting pass or dribble is formulated as

$$E_{pd} = \sum U_{opp} - U_{self} \quad (4)$$

The robot continues to dribble if $E_{pd} \leq 0.0$, to pass otherwise. This means that the robot passes the ball to one of the teammate if the opponent seems to be stronger than itself.

3.4 Decision Algorithm to Whom the Pass Should Be Sent

Having decided to pass to teammate, the owner of the ball must decide to whom it should pass the ball. First, it calculates the capability index my_spec_i for every teammate i by

$$my_spec_i = \alpha_i \frac{vel_i}{vel_{ave}} \cdot \beta_i \frac{acc_i}{acc_{ave}} \cdot \gamma_i \frac{\omega_i}{\omega_{ave}} \cdot \delta_i \frac{sensor_i}{sensor_{ave}} \cdot \epsilon_i \frac{kick_i}{kick_{ave}} \quad (5)$$

in a similar manner with the previous formulation, where my_spec_i is capability factor of the teammate i , vel_i is the maximum velocity of the teammate i , ω_i is the maximum rotation vector of the teammate i , acc_i is the maximum acceleration of the teammate i , $kick_i$ is the maximum kicking capability of the teammate i , $sensor_i$ is the maximum sensing distance of the teammate i , α_i , β_i , γ_i , δ_i , ϵ_i are weight coefficients for the teammate i (currently set to 4.0, 1.0, 1.0, 1.0, 1.0, respectively for all i , by heuristics).

Next, it calculates the distances with all teammates, and evaluates the maximum distance d_{max} of them.

$$d_{max} = \max_i(|\mathbf{x}_i - \mathbf{x}|) \quad (6)$$

Also, it calculates to the distances between all teammates and the offense goal, and evaluates the maximum distance g_{max} of them.

$$g_{max} = \max_i(|\mathbf{x}_i - \mathbf{x}_g|) \quad (7)$$

Then, the index P_i is calculated for each teammate i such as

$$P_i = (my_spec_i)^{-1} + \zeta_i \frac{|\mathbf{x}_i - \mathbf{x}|}{d_{max}} + \eta_i \frac{|\mathbf{x}_i - \mathbf{x}_g|}{g_{max}} + \theta_i \quad (8)$$

where \mathbf{x}_i is the position vector of the teammate i , \mathbf{x} is the position vector of the robot who owns the ball, \mathbf{x}_g is the position vector of the goal of the opponent, ζ_i and η_i are weight coefficients for the teammate i , θ_i is a value which represent the (equal to 1 if the opponent is on the direction to the teammate i and equal to 0 for otherwise).

Having calculated P_i for all teammates, the owner of the ball sends pass to the teammate i who has the smallest index value. The meanings of Equation (8) is as follows: the owner of the ball tries to find a teammate who is nearest to the offense goal, but if the distances are equal, it tries to send the pass to forward player who is much suitable for shooting.

The other behavior such as defense or support plays are defined similarly, and partly described in [8].

4 Simulation Experiments

4.1 Strategy Evaluation by Behavior Criteria

It is also important for the team and the manager of the team to evaluate the behaviors of all players. In other words, the team strategy must make the best use of the capabilities of all teammates. In this sense, certain assessment points are given to each behavior that represents the contribution of the player to the team. These points are defined as shown in Table 2. This method of assessment means that each player should play its assigned role in the team. The role could be assigned dynamically during the game or through some learning algorithms for future. This assessment is done in every timing of the selection of behaviors. The values of dribble in Table 2 is much smaller than others, because dribbling continues for a certain time whereas passing and shooting are done in an instant. Please note that these values are independently designed from the victory or defeat of the game.

Table 2. Assessment points for each behavior

behavior	forward (points)	mid-fielder (points)	defender (points)
dribble	3	2	1
pass	10	20	30
shoot	30	20	10

4.2 Original Soccer Simulator

Given the behavior rules for each robot and that for the team, two kinds of simulation experiments are done. The original simulator has been implemented in C language on SparcStation5, which is different from Soccer Server by Dr. Noda for *RoboCup-97* [10]. The number of robot players are limited to 5 (not 11) due to the insufficient computing power. Currently the simulator is being rewritten so as to fit Soccer Server for *RoboCup-97*.

The first experiment is to investigate the effect of the formation of robot players with the same characteristics, and the second experiment is to investigate the effect of the characteristics of the robot players. In order to fix the kick-off team, only one player in team B is initially located very near to the center circle so that the ball is found in its sensing area. An example of initial positions of robot players is shown in Figure 2. The game continues until either team gets goal, and the total assessment points are compared as well as the victory of the game. Note that the results are always the same if the same random seeds are given. In other words, the results depend on the random seeds which is used for deciding the wandering direction to search for the ball or other players.

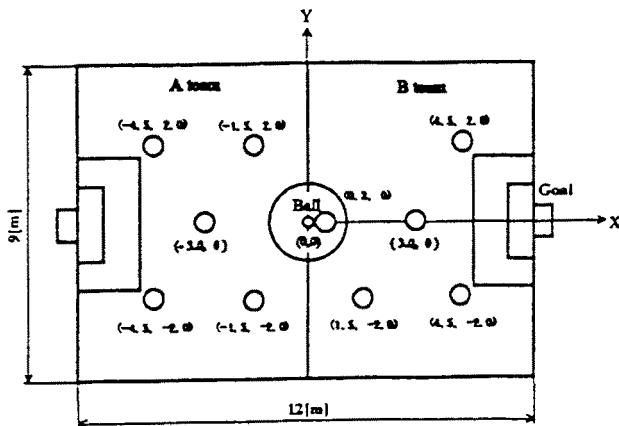


Fig. 2. An example of initial positions

4.3 Experiment 1: the effect of formation

This experiment is to investigate the effect of the formation of robot players who have the same characteristics. Provided that five robot players in one team, four kinds of formations have been used in this experiment as shown in Figure 3. The characteristics of the components in the teams are fixed into one. In order to observe the cooperative plays (passes), the robot players are set to D-type who tend to send pass rather than to dribble. The results of this experiment are shown in Table 3.

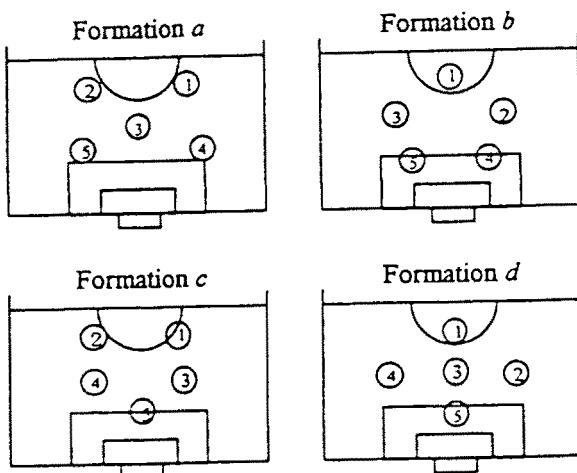


Fig. 3. The formation of robot players

Table 3. The result of experiment 1

	team A		team B		(kick off)	time	winner
	Formation	total points	Formation	total points		(sec)	
(1)	b	107	a	54		703	B
(2)	c	91	a	19		504	A
(3)	d	77	a	91		358	A
(4)	a	207	b	57		1238	A
(5)	c	130	b	176		1401	A
(6)	d	69	b	16		264	A
(7)	a	128	c	52		821	A
(8)	b	151	c	20		631	A
(9)	d	85	c	36		547	A
(10)	a	334	d	316		2453	B
(11)	b	35	d	130		557	B
(12)	c	19	d	83		324	A

4.4 Experiment 2: The effect of the combination of characteristics of the robot players

Assuming one fixed formation (*formation a*, for example), several combinations of the types of characteristics are tested. The combination is shown in Figure 4. *Pattern FMD* is the reference combination which is composed of two F-types, one M-type, and two D-types. *Pattern FFF* is composed of F-types only, *pattern MMM* of M-types only, and *pattern DDD* of D-types only.

These four types of teams had some games, and the results are shown in Table 4.

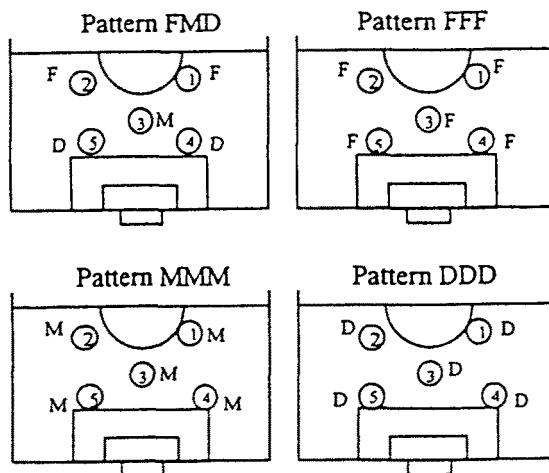
**Fig. 4.** The patterns of the combination of different characters

Table 4. The result of experiment 2

	team A		team B (kick off)		time (sec)	winner
	pattern	total points	pattern	total points		
(1)	FMD	0	FMD	90	111	B
(2)	FFF	105	FMD	27	155	A
(3)	MMM	0	FMD	90	111	B
(4)	DDD	128	FMD	30	383	A
(5)	FMD	0	FFF	90	111	B
(6)	FMD	134	MMM	10	203	A
(7)	FMD	12	DDD	89	503	B

5 Discussions

5.1 Discussions on experiment 1

As expected, D-type robot players tend to send passes rather than to dribble. If all robot players were F-type, the holder of the ball would rush to the goal to the opponent and the ball would be grabbed by opponents. Generally, there is a tendency that the kick-off team loses (9 times out of 12). The reason is that the owner of the ball is the nearest to the opponent goal among the teammates so that it could not send pass and the ball was grabbed by an opponent. This is due to the combination plays; the algorithm of support plays must be refined.

In the case of (2), the team A which employs *formation c* surrounded the opponent who dribbles the ball, and grabbed the ball. On the contrary, *formation c* lost the game with kick-off (cases (7) - (9)). This means that the *formation c* is suitable for defense. *Formation d* took the best score both in kick-off side (two wins and one loss) and non-kick-off side (three wins). This is because center line is kept by three robot players in *formation d* and this is robust enough for defense. At the present condition, this formation is the best.

In cases (10) - (12), all robot players in team B obtained some assessment points. On the contrary, in cases (3) and (6), only partial robot players got points. This status comes from the relative positions of robot players, and sometimes two robot players send passes meaninglessly. On this point, the assessment point must be revised.

5.2 Discussions on experiment 2

Generally, the situation is very similar to the experiment 1. In cases (1) - (3), (5), (6), the elapsed time is very short, because the owner of the ball dribbled and shot just after the kick-off, or the opponent grabbed the ball just after the kick-off and then dribbled and shot. In the employed formation (*formation a*),

there is only one robot player in the center, which is not robust enough for the defense. This means that the defensive positions (position 4 and 5) should be much closer with each other in order to cover the center with their sensing areas.

From the results in Table 4, *pattern FFF* and *pattern DDD* took the good results, while *pattern FMD* did not show any interesting features. The authors have expected that the different combination of characteristics should be stronger than others, but the results are different. Since the number of robot players are not enough, the effect if the interaction between players are small, and effect of the characteristics of each player have resulted directly. Moreover, if much more interactions (*e.g.* support plays, combination plays are defined, the result would be different.

In order to investigate the effect of the characteristics of the robot players, much more experiments should be done including other formations.

5.3 General Discussions

For the two experiments, the initial positions seem to be influential to the results. The relationships between relative positions and the sensing distance of robots is also important.

All players should keep their position considering their own sensing distance. The assessment points shown in Table 2 must be expanded to evaluate cooperative behaviors such as support plays for the passes, combination plays, etc. The success or the failure of the passes must also be introduced. As mentioned before, meaningless continuation of passes should not be evaluated.

Also, the calculations for deciding pass or dribble are too much complicated and need computing power. This part needs to be refined.

What would be the optimization function, the elapsed time, total assessment points, or the victory of the game? In many simulation experiments, the authors felt the strong correlation between total assessment points and the victory of the game. But there are exceptions in experiment 1. In order to discuss the team strategy, different formations could be taken depending on the tide of a game, for example. Or, different combinations of robot players may produce other strategy. In any case, the algorithms of the support plays and defense plays must be refined. This paper does not intentionally use any leaning methods nor game theory until now. Since some weight parameters are used in the modeling as explained in chapter 3, and since assessment points could be feedbacked to robot players, such approach can be introduced without any major changes of the system, and the authors feel that it is really necessary.

6 Conclusion

The platform of robotics soccer for multi-agent studies have been designed and implemented. Assessment points for each behavior are defined, and the total points during the game are compared. By using this simulator, several formations of the team and the several combinations of the characteristics of the robot

players have been tested. Through the observation of the robotics soccer game, the simulator gave us interesting results as well as many important hints on multi-agent research. For the moment, the number of robot players are limited to five due to the lack of computing power. If enough computing power is provided, the number of the robot players in a team should be expanded to eleven.

References

1. Ueyama,T., Fukuda,T., Arai,F.: "Approach for Self-organization – Behavior, Communication, and Organization for Cellular Robotic System –", *Proc. International Symposium on Distributed Autonomous Robotic Systems*, pp.77-84, 1992.
2. Asama,H., Ozaki,K., Ishida,Y., Yokota,K., Matsumoto,A., Kaetsu,H., Endo,I.: "Collaborative Team Organization Using Communication in a Decentralized Robotic System", *Proc. IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, pp.816-823, 1994.
3. Ozaki,K., Asama,H., Ishida,Y., Yokota,K., Matsumoto,A., Kaetsu,H., Endo,I.: "Negotiation Method for Collaborating Team Organization among Multiple Robots", *Distributed Autonomous Robotic Systems (Asama et al. Eds.)*, Springer-Verlag, Tokyo, pp.199-210, 1994.
4. Ichikawa,S., Hara,F.: "An Experimental Realization of Cooperative Behavior of Multi-Robot System", *Distributed Autonomous Robotic Systems (Asama et al. Eds.)*, Springer-Verlag Tokyo, pp.224-234, 1994.
5. Matsumoto,A., Asama,H., Ishida,Y., Ozaki,K., Endo,I.: "Communication in the Autonomous and Decentralized Robot System ACTRESS", *Proc. IEEE Workshop on Intelligent Robots and Systems*, pp.835-840, 1990.
6. Kitano,H., Asada,M., Kuniyoshi,Y., Noda,I., Osawa,E.: "Robocup: The robot World Cup Initiative", *Working Notes of IJCAI Workshop: Entertainment and AI/Alife*, pp.19-24, 1995.
7. Khatib,O.: "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots", *The International Journal of Robotics Research*, Vol.5, No.1, 1986.
8. Matsumoto,A., Nagai,H.: "The modeling of the Team Strategy of Robotic Soccer as a Multi-agent Robot System", in *Distributed Autonomous Robotic System 2 (Asama et al. Eds.)*, Springer-Verlag, Tokyo, pp. 115-126, 1996.
9. Matsumoto,A., Nagai,H.: "An approach to the Evaluation of Decentralized Decision Making in Multi-agent Robotic Soccer", *Proc. of the 3rd France-Japan Congress on Mechatronics*, pp. 411-415, 1996.
10. Noda,I.: "Soccer Server v.1.5", <http://ci.etl.go.jp/~noda/soccer/server.html>, 1997.

Real-Time Vision Processing for a Soccer Playing Mobile Robot

Gordon Cheng and Alexander Zelinsky

Department of Systems Engineering
Research School of Information Sciences and Engineering
The Australian National University
Canberra, ACT 0200, Australia
<http://wwwsyseng.anu.edu.au/rsl/>

Abstract. This paper describes vision-based behaviours for an Autonomous Mobile Robot. These behaviours form a set of primitives that are used in the development of basic soccer skills. These skills will eventually be utilised in an effort to tackle the challenge that has been put forward by the “The Robot World Cup” initiative. The focus of the discussion will be on the vision processing associated with these behaviours. Experimental results and analysis of the visual processing techniques are also presented.

1 Introduction

In this paper we will discuss the basic soccer skills we have developed for a mobile robot. A self-contained autonomous mobile robot in a physical game of soccer provides an excellent challenge for robotics research. Under the initiative of the “The Robot World Cup” (RoboCup) [Kitano et al., 1997] the challenge was to construct robots to play a game of soccer. The system is our preparation for the up and coming RoboCup events (e.g. RoboCup’98). Our current focus is on the primary sensing capabilities of the robot with emphasis on high and robust performances for the overall system. We believe providing a robot with specialised skills must come first, and must not be ignored before moving onto the design of higher level skills. We have commenced work on the construction of visual based behaviours for this challenge. We have committed to the development of localised vision sensing for our robot.

In order for a robot to play soccer it must be able to react to its changing external environment quickly. It must also be able to navigate freely on the playing field while avoiding any obstacles in its way. A Behaviour-based [Brooks, 1986] approach for the construction of our system have been chosen. Our system are built from individual competence modules; which involve a strong coupling of the robot’s sensors to its actuators. The end-effect of the collective interaction of the individual modules with the external environment produces a competent and coherent control system for the entire robot.

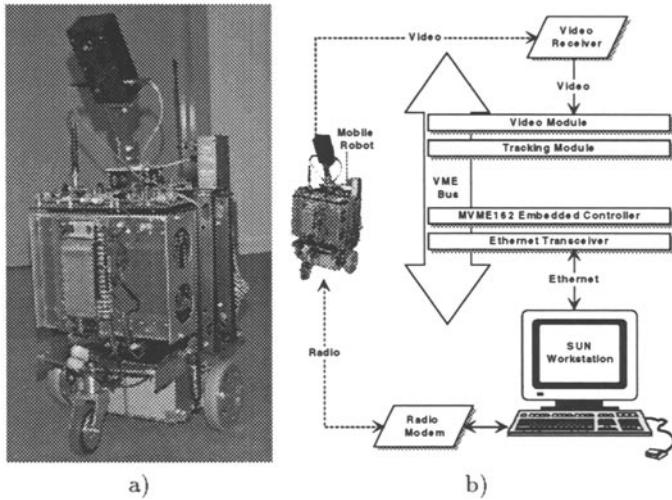


Fig. 1. System Configuration: a) Yamabico Mobile Robot b) System Overview

2 Our System

The configuration of our experimental system is shown in Figure 1 consists of three components, an Off-board vision processing unit, a Radio communication unit, and a Mobile Robot. The vision processor is a Fujitsu MEP vision system. It comes with two processing modules, a video module and a tracking module. The system is designed to support five tracking module simultaneously. These modules are connected via a VME-bus backplane to a Motorola MVME-162 Embedded Controller running the VxWorks operating system. The VxWorks operating system handles all program executions. A tracking module can track up to 100 templates in each video frame in a 30 Hz video stream. Currently we are using one tracking module. The video module provides two selectable input channels and one output channel. All signals are NTSC video format. A NTSC video receiver is connected to one of the input channels on the video module. This receiver is used to accept video signals from the mobile robot. The communications system is a SUN-workstation (running Solaris OS) with a radio modem attached. The communication system manages all network traffic between the vision system and our mobile robot(s). For example, once a image frame is processed, a command from the vision system is send back to the robot, guiding it on its way.

The Yamabico mobile robot shown in Figure 1a [Yuta et al., 1991]. It has a multi-processor based architecture that houses a number of processor modules. All of these modules communicate through the Yamabico-bus, using a Dual-Port-Memory mechanism. The robot has a MC68000-CPU master module, running the Morsa OS [Yuta et al., 1991]. A T-805 locomotion module, that provides all of the motor feedback and control of the robot [Iida and Yuta, 1991]. An ultrasonic module is also provided on the robot, it is not used in our experiments.

In addition, a radio modem, a small size CCD camera and a video transmitter has been included in our system. The modem is used to communicate with the communication system. The video transmitter and camera provide video input to the vision system.

2.1 Information flow

The basic information flow of the system is the robot transmits video signals taken by the on-board CCD camera. A vision system connected to a video receiver receives the signals for processing and then sends the results to a SUN workstation. The workstation transmits the results back to the robot via the radio modem. An overview of this organisation is shown in Figure 1b.

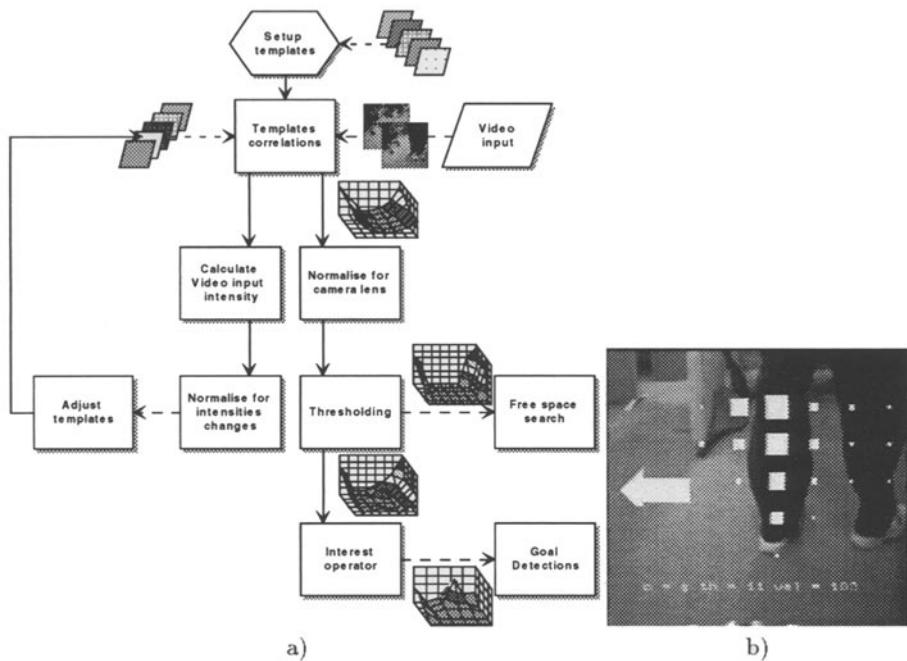


Fig. 2. Vision Processing: a) Processing flow, b) Template Matching

3 Visual Processing

One of the great challenges in robotics is to provide robots with appropriate sensing capabilities, that can supply sufficient information to allow the accomplishment of a task within a reasonable response time. Many sensors such as sonar or

infra-red can only provide low-level range information, and cannot provide any kind of goal information. We believe vision can accommodate this deficiency and provide a robot with the wealth of information its needs to accomplish tasks more effectively. By using efficient techniques, real-time performance can be achieved. The vision processing methods we are using are based on cross-correlation template matching. Template matching is a simple and effective method for image processing that allows us to achieve real-time performance. The free space detection process can be summarised as follows. Each image is segmented into a grid of 8 columns and 7 rows, producing 56 individual cells, each cell is 64x64 pixels in size. Each of these cells in the live video is correlated against a stored image of the floor taken at the beginning of each experiment. Due to the lens distortion on the CCD camera that we are using, normalisation is performed on the correlation values. After normalising we perform adaptive thresholding of the correlation values, the result of this process determines the amount of free space available to the robot. The goal-detection process utilises the values produced by the normalisation stage of the free-space detection process. An interest operator is applied to the normalised values, this operator highlights any features for the goal seeking behaviour to focus its attention on that could be interesting. To verify that the correct goal has been found a simple reassurance scheme is used. Each of these operations is explained later in the paper. The flow of processing for lighting adaptation is shown in Figure 2a.

$$D = \sum_x^m \sum_y^n |g(x - m, y - n) - f(x, y)| \quad (1)$$

3.1 Template Matching

For each template match, a correlation value is produced ranging from 0 to 65536. This value determines how well matching occurred (the lower the value the better the match). Figure 2b shows the correlation values using white squares, the better the template match the smaller the squares. These correlation values are calculated by using Equation (1). The basic assumption of this technique is that the floor the robot is to travel is of constant texture. As with template matching a set of stored templates are needed. To reduced storage space, one grid cell of the floor is stored for each row of templates to be matched. This cell is taken from the center of the image of a cleared floor. The correlation between the center and the outer edge of the image can vary due to the distortion of the lens. The normalisation stage of the visual processing combats this problem. Figure 3a shows a clear floor image taken from the CCD camera mounted on top of the robot. Figure 3b shows a plot of correlation values of this image. Figure 3c shows a plot after normalisation.

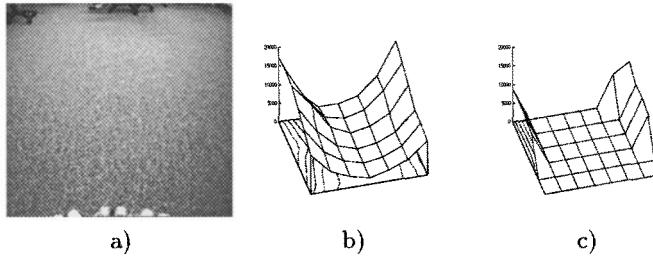


Fig. 3. a) Floor, b) Correlation plot of a floor, c) Normalised correlated values

$$E(x, y) = \begin{bmatrix} e_{(0,0)} & e_{(0,1)} & \dots & e_{(0,n-1)} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ e_{(m-1,0)} & \dots & \dots & e_{(m-1,n-1)} \end{bmatrix} \quad (2)$$

3.2 Normalisation

Empirically it was determined that a polynomial relationship for the correlated values existed, due to the distortion of the camera lens. In Figure 3a, we can see the confidences of each template match plotted by the correlated values for each location of the image. The lower the value, the higher the confidences. To overcome this problem, a polynomial curve was fitted to the plot. For implementation purposes the discrete form shown in Equation (2) was used to perform the normalisation. At each location of the template match, an associated error value is computed.

3.3 Calculating Free-space

Free-space is determine using a threshold value, this is calculated by using the new correlated values produced from the normalisation stage. The threshold value is determined using the variances between all the normalised correlated values, refer to Equation (3). This threshold is then applied to the normalised correlated values, the results are shown in Figure 3b.

$$\sigma = \frac{\sum_n f(x, y) - \bar{f}}{n - 1} \quad (3)$$

3.4 Determining velocity

The velocity of the robot is calculated using the amount of free space that is available. The size of free space is determined by the Free-Space behaviour, and is calculated from the ratio of the maximum horizontal and vertical free space available in its view. The calculation is shown in Equations (4) and (5).

$$top_{speed} = max_{speed} \times \frac{y}{max_y} \quad (4)$$

$$velocity = top_{speed} - | \frac{top_{speed} \times x}{max_x} | \quad (5)$$

3.5 Interest-Operator

The Goal detection scheme in the original system searches the complete images for a given goal, it was noticeably inefficient [Cheng and Zelinsky, 1996]. An interest operator was introduced to increase the efficiency of the goal seeking behaviour by reducing the search space. This interest-operator can also be used as a feature detector that doesn't use an explicit model, and leads to landmark acquisitions. The goal detection processing is as follows. An image from the video camera is shown in Figure 4a. This image is then processed through the correlation step described previously. This produces the correlation values shown in Figure 4b. Normalisation is then performed on the values, as shown in Figure 4c. The final step involves applying the Interest-operator, its effect is shown in Figure 4d. The Interest-operator produces a single peak that easily identifies a feature of interest in the robot's view, using Equation (6). This calculation has the effect of producing a second derivative of the correlated values.

$$R(t) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (6)$$

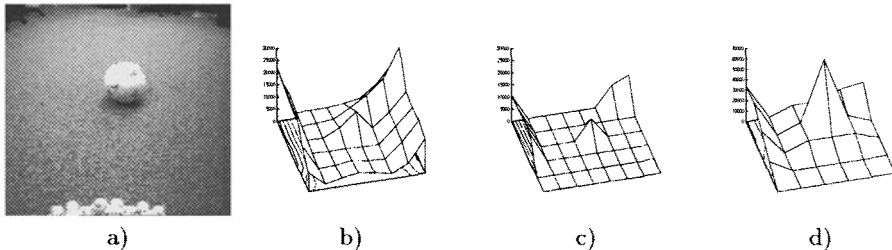


Fig. 4. Interest Operator: a) Soccer ball, b) Correlated Ball, c) Normalised Ball, d) Segmented Ball

3.6 Screen to floor transformation

After determining where in the video image we would like to move toward, we will need to transpose this location in the image to real world coordinates. Due to the fix configuration of the current camera system, minimum visual coverage

is provide to the robot, as shown in Figure 5a. By exploiting this constraint we have derived a simple pixel to floor transformation.

The coordinate transformation is calculated by measuring four distances of the floor, as illustrated in Figure 5b and Figure 5c. The first distance is the height of the camera to the floor, $height_y$. The second distance is the nearest view point of the camera, $blind_y$. The third distance is the furthest visible point to the top view of the camera, $blind_y$ plus $length_y$. The fourth distance is the furthest point to the left/right of the camera view, $length_x$. With these measurements we can calculate the three angles required, using Equations (7), (8) and (9). From the angles we can determine the angular ratio in both the x and the y directions. Figure 5b shows the side view of the robot. Figure 5c shows the top view of the robot.

$$\alpha = \tan^{-1} \left(\frac{height_y}{blind_y} \right) \quad (7)$$

$$\theta = \tan^{-1} \left(\frac{height_y}{blind_y + length_y} \right) \quad (8)$$

$$\beta = \tan^{-1} \left(\frac{blind_y + length_y}{length_x} \right) \quad (9)$$

The following equation calculates the transformation of the y^{th} pixel to the vertical distance on the floor.

$$y = \frac{height_y}{\tan \left(\theta + \frac{pixel_y(\alpha - \theta)}{screen_y} \right)} + \frac{robot_y}{2} + blind_y \quad (10)$$

where $pixel_y$ is the y^{th} pixel of the screen. $robot_y$ is the length of the robot in the y direction.

The following equation calculates the transformation of the x^{th} pixel to the horizontal distance on the floor.

$$x = \tan \left(\frac{\beta (1 - 2pixel_x)}{screen_x} \right) \times y \quad (11)$$

where $pixel_x$ the x^{th} pixel of the screen.

From the Equations (10) and (11), we can calculate the floor space (x, y, θ) for the robot to travel towards from the corresponding point in the image space.

3.7 Lighting Adaptation

A scheme for combating the problem of subtle lighting changes have been incorporated into our system. Others researchers have also experienced problems using vision sensors, Lorigo [Lorigo, 1996] Horswill [Horswill, 1994]. Problems such as the shadow of an obstacle being mis-interpreted as being part of the obstacle. In our implementation, we use the correlation processor to determine the

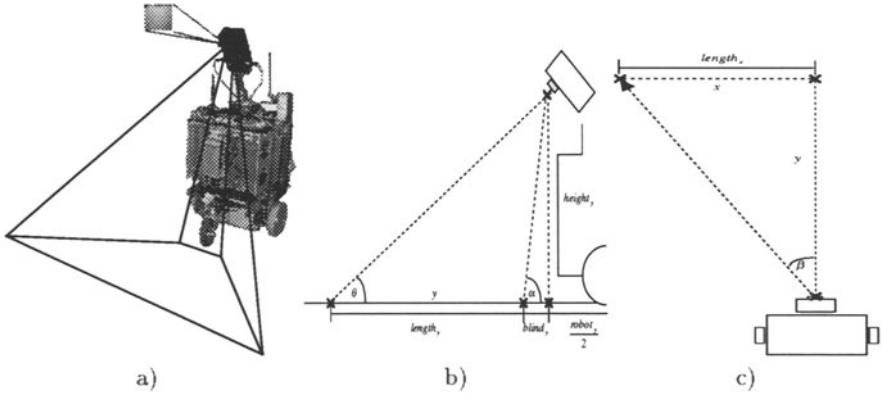


Fig. 5. Camera configuration: a) Coverage, b) Side view, c) Top view

average intensity of an given region in the video steam. This allows the system to dynamically adapt to various lighting condition in real-time.

$$\bar{I}_{now} = \frac{\sum_x^m \sum_y^n |f(x, y) - Z_t(x, y)|}{m \times n} \quad (12)$$

where $f(x, y)$ is the greyvalue of the pixel
in the last frame

$Z_t(x, y)$ is a zero template

$$\bar{I}_{new} = \begin{cases} |\bar{I}_{old} - \bar{I}_{now}| \leq Sensitivity \bar{I}_{old} \\ |\bar{I}_{old} - \bar{I}_{now}| > Sensitivity \bar{I}_{now} \end{cases} \quad (13)$$

$$\forall(m, n) : g(m, n) = g(x, y) + (\bar{I}_{old} - \bar{I}_{new}) \quad (14)$$

The adaptation process are as follow:

- calculate current image intensity, using Equation (12);
- determine the differences between the old intensity of the initial template in comparison to the new intensity of the current image frame;
- check if these differences are within a sensitivity range, using Equation (13);
- finally adjust each pixel of the template according to this differences, using Equation (14).

Figure 6 shows our system adapting to subtle light changes in the environment. The size of the black squares indicate the level of correlation, after adaptation the correlation improves markedly.

3.8 Recognition/Reassurance

The final stage of the goal detection process, reassures the goal detector that it is looking at the visual cue that it is seeking. Prior to the execution of an experiment a set of templates of the goal (e.g. Ball) are stored. All templates are recorded from various positions from the goal.

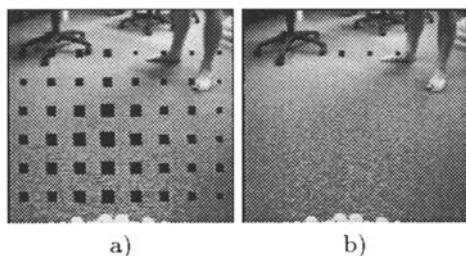


Fig. 6. Adapting to lighting changes: a) before, b) after

3.9 Noise filtering

Filtering is also performed to overcome the problems caused by noisy transmission of the video signals to the vision system. The filter also performs sanity checking on all the commands that are sent to the robot, this checking is done by a voting scheme. This ensures the robot does not get into states such as start-stop motion between consecutive video frames.

3.10 Speed of processing

We deliberately slowed down the processing of video frames, this allowed us to analyse the performance of our overall system. It serves as an indication of the minimum amount processing that is required without a substantial loss in performance and functionality. In normal operation the system runs at 30 Hz (i.e. 30 video frames a second), at 2 Hz the system's performance begins to degrades to the point that it becomes unreliable.

4 Visual Behaviours

We have based our system on three basic visual behaviours, Collision Avoidance, Obstacle Avoidance, and Goal Seeking. The focus of the discussion will be on the vision processing associated with these behaviours. The modular computational structure of a Collision Avoidance behaviour can be simplified into Detection and Avoidance. The detection stage involves determining the availability of free-space in front of the robot for unrestrained motion. If insufficient free space is available the robot suppresses its other behaviours and activates the collision avoidance scheme (e.g. looking for a ball). A situation in which this behaviour can be activated is when the robot is not able to move away from an obstacle, such as a dynamically moving obstacle (such as other robots). Therefore this behaviour acts as a safeguard for the obstacle avoidance behaviour. The Obstacle Avoidance behaviour works by searching for free space within the robot's view of its environment. The robot's basic strategy is to move to where there is free space. The visual processing of the free space search for both of these behaviours will be discussed later. One of the problems with other vision-based avoidance

methods (such as optical flow Kröse *et al.* [Kröse et al., 1997]) is their inability to integrate with goal based navigation and the inability to distinguish between a goal and obstacles. The Goal Seeking behaviour exploits the free space searching ability of the vision system. This behaviour allows the robot to focus its attention on searching for a given goal. Once a goal is detected, the robot will perform the associated action for that goal, (e.g. Soccer ball servoing).

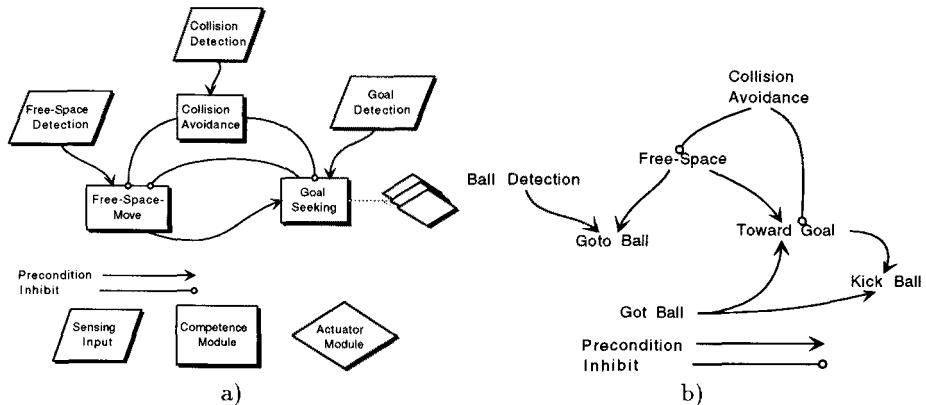


Fig. 7. System Architecture: a) General Architecture, b) Simplified network of soccer behaviours

4.1 System Architecture

The general system control architecture is illustrated in Figure 7. *Detection* modules are depicted as diagonal boxes, actual *behaviour* modules by rectangular boxes and the diamonds are an indication of the *associated actions*. The arrows indicate *preconditions* to a behaviour, and lines with the small circle indicate *inhibition*. Figure 7 shows that the Collision Avoidance has the highest priority over all the other behaviours. The Free-Space-Move needs to be active when the Goal Seeking behaviour is activated. By adapting the goal seeking behaviour we can perform the appropriate actions.

4.2 Soccer skills

By exploiting the opportunistic property of the Goal Seeking behaviour we have developed basic soccer playing skills for our robot. A simplified network of behaviours is shown in Figure 7b. In this version of the network of behaviours has multiple instances of the Goal Seeking behaviours. i.e. Kick Ball, Toward Goal and Goto Ball exist.

Got Ball Got Ball is determined by visually detecting if a ball is directly in front of the robot.

Goto Ball The Goto Ball behaviour is activated by Goal Detection, once a ‘Ball’ is detected the robot causing the robot to go towards the ball (visually servoing), as shown Figure 8a-b. The robot avoids obstacles on its way using Free-Space-Move. The motion of the robot is safeguarded by the Collision Avoidance behaviour. This behaviour is an instance of the Goal Seeking behaviour driven by a visual cue, i.e. “The Ball”.

Toward Goal This behaviour is triggered by a precondition from Got Ball, and its subsequent motion is determined by the Free-Space-Move behaviour together with its goal to move toward the *Soccer Goal*.

Dribbling The effects of dribbling was achieved through the combination of Got Ball detection, and Free-Space-Move.

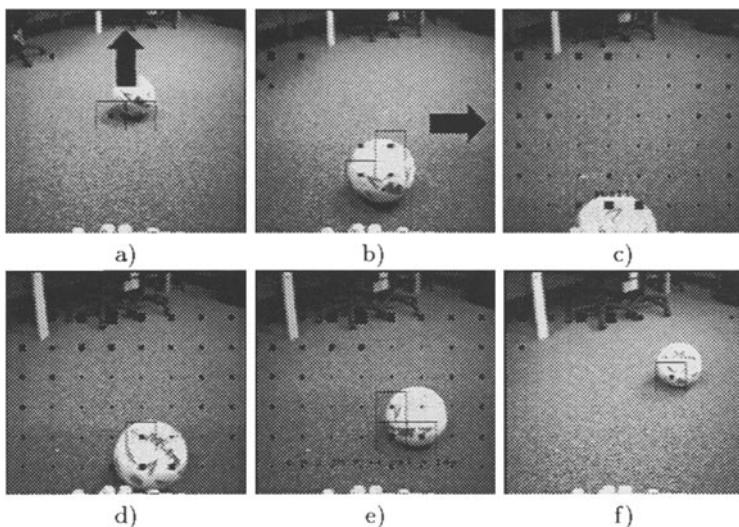


Fig. 8. Kicking

Kick Ball This behaviour simply waits until the *Soccer Goal* are sufficiently close enough, then it activates a kick action. That is done by stopping then accelerating rapidly, producing a kick like action. Figure 8 shows visual servoing and then the kicking action by our robot.

5 Conclusion and Further Work

The system we have presented here demonstrated a set of basic soccer playing behaviours for a mobile robot. The was achieved through the use of local vision.

All of the behaviours exhibit real-time performance that are robust and reliable in both static and dynamic environments. In normal operation the current system can perform its task at a robot velocity of up to 0.5 m/sec. Our aim is to move the robot at 1 m/sec.

Image correlation is an efficient method for achieving real-time vision processing. Although our system performs visual processing using special hardware, we believed that the vision processing we have described is well within the reach of current commercial processors. This has inspired us to begin the construction of on-board vision processing modules for our future works.

Presently our work has been primary focused on the development of robust vision based soccer skills for a mobile robot. The system in its current states does not conform with the rules that has been established in the present RoboCup. Further work will be gear toward a colour based image correlation system. This will allow our system to conform with the RoboCup specifications. Colour template matching will also enhance the reliability of our system.

Acknowledgements

This research is funded by a grant from the Australian Research Council and supported by Wind River Systems supplier of the VxWorks® operating system.

References

- [Brooks, 1986] Brooks, R. (1986). A Layered Intelligent Control System for a Mobile Robot. In *IEEE Trans. on Robotics and Automation, RA-2*.
- [Cheng and Zelinsky, 1996] Cheng, G. and Zelinsky, A. (1996). Real-Time Visual Behaviours for Navigating a Mobile Robot. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 973–980. IROS'96.
- [Horswill, 1994] Horswill, I. (1994). Visual Collision Avoidance by Segmentation. In *Intelligent Robots and Systems*, pages 902–909. IROS'94.
- [Iida and Yuta, 1991] Iida, S. and Yuta, S. (1991). Vehicle Command System and Trajectory Control for Autonomous Mobile Robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, Osaka, Japan. IROS'91.
- [Kitano et al., 1997] Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. (1997). RoboCup: The Robot World Cup Initiative. In *Proceedings of the first International Conference on Autonomous Agents*, pages 340–347, Marina del Rey, CA.
- [Kröse et al., 1997] Kröse, B., Dev, A., Benavent, X., and Groen, F. (1997). Vehicle navigation on optic flow. In *Proceedings of 1997 Real World Computing Symposium*, pages 89–95.
- [Lorigo, 1996] Lorigo, L. M. (1996). Visually-guided obstacle avoidance in unstructured environments. Master's thesis, Massachusetts Institute of Technology.
- [Yuta et al., 1991] Yuta, S., Suzuki, S., and Iida, S. (1991). Implementation of a Small Size Experimental Self-Contained Autonomous Robot. In *Proceedings of the 2nd Int. Symposium on Experimental Robotics*, pages 902–909, Toulouse, France.

A Method Applied for Soccer's Behaviors Using Proper Feedback and Feedforward Control

Hironori Mizuno^{1*} and Masakatsu Kurogi¹ and Yukihide Kawamoto¹ and
Yoichi Muraoka¹

School of Science and Engineering, Waseda University, 3-4-1 Okubo Shinjuku Tokyo
169, JAPAN

Abstract. As a practical way of achieving "Athletic Intelligence(AI)", we have tried to build a robot capable of playing soccer. In this paper, we made a mobile robot that can shoot and dribble a ball. In order to shoot the rolling ball, the robot must predict the future position of the ball so that it can move ahead of the ball. That is, the robot should be controlled by feedforward control rather than feedback control because feedback control does not allow enough time to catch the ball. Therefore, we think that it is important that the robot has internal model with which it can predict the target position. As long as the ball is within the field of view, the robot is under feedforward control. At the final stage of shooting, control is switched to feedback to minimum errors. When dribbling the ball through the flags, the robot must move without touching the flags and also keep the ball in front under feedback control. Since the robot has an internal model, it should follow the target using feedback control. We have checked that proper use of both control improves the two athletic movements so the robot must have an internal model that can predict the future state of the target object.

1 Introduction

As a practical way of achieving "Athletic Intelligence(AI)", we have tried to build a robot capable of playing soccer. In this paper, we made a mobile robot, called "Shoobot", that can shoot and dribble a ball. Its skills are measured in terms of the quality of specific tasks.

In this paper, we will discuss the two control problems at the level of the computational model. We especially focus on the issue of which of feedback or feedforward control should be applied to a certain given situation. We define feedforward control as which makes Shoobot move on a trajectory to the position of the target precticed by the internal model.

In the shooting problem, there are three steps.

1. The ball is far from Shoobot and thus the distance cannot be measured accurately enough. Since Shoobot cannot predict the trajectory of the ball, it has no choice but to follow the ball; this is FB control.
2. When the ball is within the scope of the visual system of Shoobot, the distance

* <http://www.info.waseda.ac.jp/muraoka/members/hiro/>

can be obtained accurately enough to allow the robot to predict the future position of the ball. Since the trajectory is obtainable, Shoobot can set one contact point and move ahead of the ball; this is FF control.

3. When the ball is near, Shoobot can be shot at the final stage. Shoobot adjusts its course so that it can position the ball just in front of the shooting device. The second step is crucial for to enable shooting behavior. Without it, Shoobot and the ball will arrive at the contact point at the same time, thus making it difficult to shoot the ball precisely.

In the dribbling problem, Shoobot must satisfy the following two different constraints. 1) Shoobot must keep the ball in front of its body. The local constraint can be satisfied through FB control. 2) Shoobot must move through the flags and finally reach the goal. This global constraint can be satisfied through FF control

It is vital for a soccer-playing robot to have an internal model which enables feedforward control.

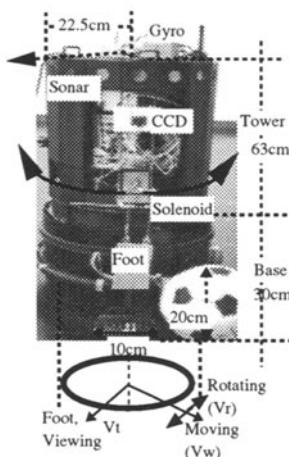


Fig. 1. "Shoobot"

2 Feedback and Feedforward

FB is a method of reducing the error less from the target in the system. FB control is kept under the constraint of the current state and thus does not predict the future state. On the other hand, we define FF control as that which makes Shoobot move on the trajectory to the target position predicted by the internal model.

Feedforward Control (FF) and Feedback Control (FB)

	Prediction	Following target	Controlling Time	Extra Time
FF	Yes	No	fast	Yes
FB	No	Yes	slow	No

Although, unlike FB control, FF control does not correct error, the predicted state may be updated by new information immediately after it becomes available. If FF control is performed well, Shoobot's motion is stable. As a result, Shoobot can have extra time to prepare for three next action.

2.1 Mechanism of shooting

In order to shoot a rolling ball, FF control is required so that Shoobot can move ahead of the ball, because FB control does not allow enough time to catch the ball.

Shoobot requires a model that can predict the position of the rolling ball in near future. If Shoobot can always predict the contact point, it can go there smoothly without stopping. That is, the internal model guides Shoobot to where it should go and when it should stop. In this case, Shoobot has the merit that it can move quickly and describe its own trajectory without suffering from the effects of the external state on the way.

Using a CCD camera attached on the body as shown figure 1, we have determined that how each pixel on the captured 2D image corresponds to a position in real space. However, not all pixels correspond to a specific point because the lens is bent at its circumference and a pixel corresponding to a far point becomes relatively longer. (the camera is set to look down), and because the extent of the distance is limited according to the camera's position on the body and its direction. Our single-camera (1.57[rad] wide, with 1/2 CCD, attached at a height of 0.7m) on the tower can distinguish an object at the maximum distance of only about 2[m]. If we can use two cameras, this limit of the distance can be made longer.

If the ball is beyond the limit of the distance, Shoobot cannot move via FF control because the future position of the target cannot be predicted. We will explain FF and FB control of Shoobot according to the distance.

1. The case that Shoobot cannot determine the distance because the ball is too far (beyond 2[m]): Shoobot moves via FB control by maintaining constant visible angle to the ball. In this case, Shoobot does not use distance information.
2. The case that Shoobot can determine the distance (within 2[m]): Shoobot predicts the future position from past positional data and moves quickly using FF control.
3. The case that the ball is near to Shoobot: After robot arrives at the predicted position, it waits for the rolling ball. In order to shoot the ball with one shot, Shoobot adjusts its position to the minimum distance.

In the second process, Shoobot calculates the speed vector of the ball from past positions and predicts the next position. We assumed that in the internal model, the ball rolls at a constant speed along straight line. If Shoobot's view is separate from its body, that is, captured from a global position [IRO96] such as a camera on the ceiling, Shoobot can use the position information in the entire space. With a global camera, Shoobot does not require separate processes such

as those described above. This means that Shoobot can always predict the target position. In either case, robot must finally catch the ball using FB control.

When robot moves only via FB control, gain in FB control must be adjusted. If this gain is larger, robot can follow the movement of the rolling ball. In reverse, if the gain is small, robot cannot catch up with the rolling ball.

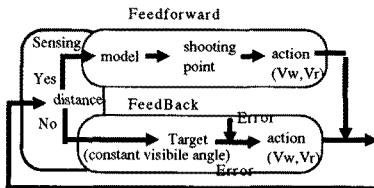


Fig. 2. Flow chart for shooting

2.2 Mechanism of dribbling

We will explain here how the robot dribbles the ball through the flags. In shooting behavior, the most important thing is whether robot can predict the position of the rolling ball and move ahead to the predicted point. In this case, it is not necessary to consider for Shoobot's path.

On the other hand, in dribbling behavior, it is necessary to consider the path, because robot must always keep the ball in front of its body. That is, dribbling behavior must simultaneously satisfy two constraints. One is that robot moves with the ball in front of the body. The other is that robot carries the ball to a target such as a goal.

1. Dynamic change of moving target

Robot can have only one target to move because robot's path cannot be separated. In the task in which robot must dribble the ball through the flags without touching them. In this case, the next target is decided by the position of the next flag. If next target fixes at one point, Shoobot must change its wheel angle more frequently as Shoobot comes near it, which modification leads to unrequired motion.

2. Dynamic control of the wheel speed to keep the ball in front

Shoobot must move within practical constraints such as the robot's shape and the ball size. Since both Shoobot and the ball is round, the contact between them may be at only one point. Therefore, Shoobot can push the ball without dragging. If the ball has little friction with the floor, it causes the ball to roll quickly and move around the body unexpectedly. Shoobot must always modify the wheels' direction keeping the ball on the line to the target, which was decided in the first process.

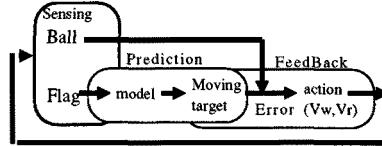


Fig. 3. Flow chart for dribbling

3 Shooting motion

We explain here how to implement the shooting motion on Nomad200.

3.1 Shooting the rolling ball

There is a vital limit in the time to reach the predicted point. In order to shoot the ball, Shoobot should judge whether it can reach the point or not within this time. Forestalling time also depends on the current conditions of the set of the wheel angle and speed. Check whether Shoobot starts moving. When the ball is

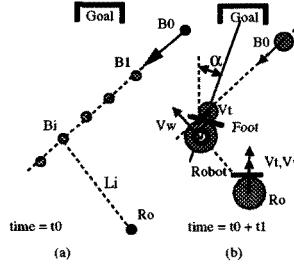


Fig. 4. How to predict the position of the rolling ball

observed moving from B_0 to B_1 , as shown in figure 4, predicted point (B_i) is i times the ball vector in the rolling direction. Each distance (L_i) from the present position (R_o) is calculated.

$$\mathbf{B}_i = \text{Forestall}(Internal)(i) + \mathbf{B}_0 \quad (i = 1, \dots, \text{number})$$

For each point, Shoobot compares the time required to move (RT_i) to the rolling time (BT_i). Since Nomad200 has nonholonomic structure as stated in Chapter 3, moving time is calculated as the total time of rotation and moving.

First, the wheels rotate at the maximum speed until the wheel angle becomes angle indicated in the target direction. After that, Shoobot moves at the maximum speed to the goal. If this time is shorter than that required to reach the rolling ball, Shoobot can start to move to the target point. In the reverse case, Shoobot cannot reach the target point before the ball arrives there.

$$\begin{aligned}
RT_i &= \frac{L_i}{Max_{RobotSpeed}} + \frac{Rot_i}{Max_{RotSpeed}} \\
BT_i &= (Interval)(i) \\
RT_i > BT_i &: Startmoving \\
RT_i \leq BT_i &: Notmoving
\end{aligned}$$

3.2 Moving algorithm

Moving time RT_i is defined as the sum of each time at the maximum speed. Since we define rotation time and moving time independently, this sum represents a maximum time as the upper limit. Since Nomad200 can perform rotation and movement of the wheel simultaneously, there is one merit that the total time may be shorter. However, it is difficult to reduce the wheel errors when adjusting both parameters simultaneously. Therefore we have tuned each parameter to attain a balance between the rotation and the moving.

For wheel movement and rotation, 0.83 and 0.36 second respectively, are needed to achieve top speed (Vt: 0.6[m/s], Vr: 1.31[rad/s]). Moving time at the beginning is greater than the rotational one. It is desirable not to alter the moving speed frequently in order for Shoobot to reach the goal faster. We adopted the moving strategy below. If the angle to the next target is larger than a certain value, rotation of the moving direction should proceed rather before forward motion. After the moving direction is set, Shoobot begins to move at the maximum speed. The moving speed is decreased as Shoobot approaches the target. As a result, we found that Shoobot arrives at the goal with an error of less than 10 cm regardless of the wheel angle at the starting point. If the differential angle is less than 0.44 rad, Shoobot winds its way through the global path.

We found that Shoobot can move stable by means of these 3 process: moving to the predicted position of the ball, waiting for the ball using FF control and finally shooting it using FB control.

3.3 Shooting a stationary ball

Shoobot requires an internal model to use FF control. We explain the task to make Shoobot move around the ball. This movement is performed so that wheel direction finally corresponds to the direction of the ball and the goal. Shoobot moves around the ball smoothly by changing the wheel direction, and can finally reach the point behind the ball.

As the internal model [ALIFE96], Shoobot uses the potential field model with repulsive and attractive forces. As explained in 2.1, as long as Shoobot can determine the distance to the ball, Shoobot can move around the ball via FF control.

Imagine a relative coordinate in which the ball is at the origin and the goal is on the x-axis. To generate the path described above, a force should be applied

to Shoobot in such a way that it is pushed toward the -y direction if positioned in Quadrants I and IV and straight to the target if positioned near the ball. Therefore, it is desirable for its path to be heart-shaped when viewed behind the goal.

To produce an path of this shape, we have designed a dynamics model, in which an attractive force and a repulsive force is virtually set on the coordinate defined by the goal and the ball. Figure 5 shows that this model assumes three fixed points: P1 on the positive x-axis which has a repulsive force, P2 on the negative x-axis which has an attractive force, and O on the origin which has a repulsive force less than P2. Combination of these forces generates the moving path of Shoobot. This explains the 3 fixed forces produced at each point (P1,O,P2). (1) The constant attractive force +a direction. (2) The repulsive force at the origin which is not in inverse proportion to the distance. (3) The repulsive force in the -b direction which is in inverse relation with the square of the distance. A point far from the origin is not influenced by force (2) or (3) and

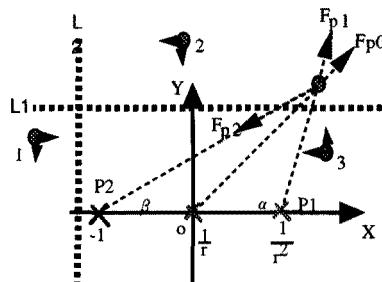


Fig. 5. A field model of repulsive and attractive force

is pulled to point P2 by only force (1). When Shoobot approaches sufficient close to the ball, forces from the origin and P1 begin to strongly push Shoobot away and it is eventually drawn to point P2. The origin represents the position of the ball. The repulsion prevents unwanted collision. In this dynamic model, the functions for the calculation are shown below. F_{p1} , F_o , F_{p2} represent an action force from each point P1, O and P2 respectively. R_{p1} , R_o and R_{p2} represent the distances of P1, O, and P2 respectively to Shoobot. F_x and F_y represent the x- and y-coordinates of the force, respectively.

$$F_x = F_{p1} + \frac{F_o(R_x - a)}{R_o} + \frac{F_{p2}(R_x - b)}{R_{p2}},$$

$$F_y = + \frac{F_o R_y}{R_o} + \frac{F_{p2} R_y}{R_{p2}}$$

$$F_{p1} = c \frac{1}{R_{p1}^2},$$

$$F_o = \frac{1}{R_o},$$

$$F_{p2} = -1$$

After dt time units, Shoobot moves to the new position (R_x, R_y) calculated from vector (F_x, F_y).

$$\frac{dR_x}{dt} = F_x,$$

$$\frac{dR_y}{dt} = F_y$$

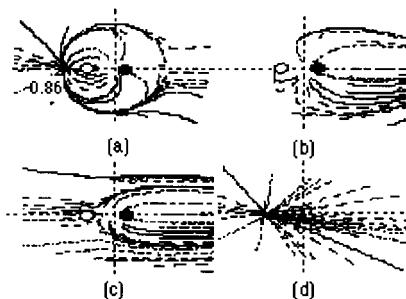


Fig. 6. Comparison of result with 3 different parameters

The dynamics of the path is manipulated via 3 parameters. Black circle marked at P_1 and white circle marked at P_2 , the value (+a) to the length of P_1 , the value (+b) to the length of P_2 , the value (+c) represents weight as shown in Figure 5

Figure 6 represents 4 distinctive features of the results of computer simulation. For each case, the parameters are (a) $a=0.5$, $b=0.1$ and $c=0.05$, (b) $a=0.5$, $b=0.1$ and $c=1.0$, (c) $a=0.5$, $b=0.1$ and $c=3.0$, (d) $a=10.0$, $b=1.0$ and $c=0.2$, and illustrate the domain that is limited from -2 to +2. As the value +c increased (a-b-c), the stream of the lines tend to be pulled in the positive x direction. This trend is dependent on the lengths of P_1 and P_2 and we did not choose the path behind the ball in the case of straight-line motion (d). The shooting algorithm described below uses the same parameters as in case (a). According to the parameters in case (a), Shoobot is pulled down at the point that is 0.86 [m] in the x negative axes.

4 Dribbling the ball

Nomad200[Nomad200] can wind its way using 2DOF (moving forward or backward and turning left and right). The body is divided into the two parts shown

in figure 1. The tower rotates around independently of the basement in which three wheels are equipped. Nomad200's movement is limited in an action under the nonholonomic structure: it can go forward in only one direction at one time.

While Shoobot is adjusting the wheel angle, the ball is also rolling. It is desirable for appropriate Shoobot movement to shift the target in the internal model to reduce errors. This is achieved through two interactive controls related to the influence of errors of the ball on change of the global target and the effects of the changing target on motion to keep the ball in front.

Shoobot has two standard coordinates on the body. One is virtual in the camera's frame, and the other, called the body coordinate, is such that at the initial time, the wheels' direction is defined as X-axis and the other axes are defined by the left-handed rule. The target can be represented directly as a position on the body coordinate.

The center of gravity of the ball seen in the 2D image corresponds to the position in the 3D body coordinate though this position includes some error. As a result, Shoobot can predict the moving point on the 3D body coordinate on which Shoobot, the ball and the flags exist.

The camera is set to look down and is parallel to the body coordinate so that Shoobot can easily determine the position from visible coordinates. The field of view extends 0.5[m] forward and 0.3[m] to the right and left sides. Unless Shoobot can adequately keep the ball in front, Shoobot will lose sight of the ball (0.22[m] in diameter).

4.1 Dribbling the ball on a line

In this section, we describe how Shoobot can actually dribble the ball on a line. Since Shoobot is bilaterally symmetric, we explain its behavior in the case of rolling, using only one side. We describe how Shoobot moves around the ball on the left side. We have selected the entire path for the case that Shoobot reaches the goal with the minimum distance and time.

In dribbling, we have classified the relationship between the path, Shoobot and the ball into two categories. The 1st category corresponds to the situation where the ball is within the safety zone, and the 2nd category outside the safety zone. Furthermore, the 2nd category is divided into 3 patterns.

We define the safety zone in which Shoobot can push the ball using feedback control. This zone is bounded by two lines drawn away from at a fixed width from the moving direction.

If the ball is in the safety zone, Shoobot has only to push the ball using feedback control. It controls and rotates wheels slightly in order to position the ball on a center line in sight. If the ball is outside the zone, Shoobot can move around the ball with three patterns of movement.

A constant angle from the moving direction is fixed for each side of. The line forming this angle is represented by the dot line in figure 7(a-2). We define the counterclockwise rotation as the positive angle. The following 3 patterns of movement shown in Figure 8, are adopted to bring the ball back within the safety zone.

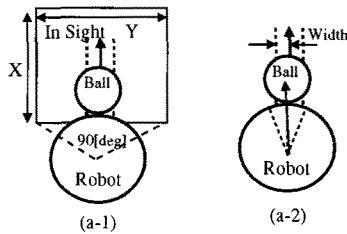


Fig. 7. Controlling the rolling ball

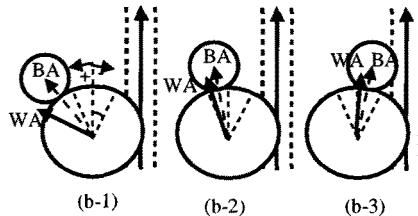


Fig. 8. Three patterns of movement

1. The position of the ball is beyond the positive fixed angle. Fig.8(b-1)
If Shoobot pushes the ball forward, it will immediately lose the ball on the left side. If the speed is slow, wheel angle is dynamically calculated to be larger than the ball angle at which the ball is lost. If Shoobot does not slow down, the rotational speed should be proportionally accelerated. Since the wheels does not rotate at the top speed (0.79 deg/s), it is necessary to reduce the speed.
2. The position of the ball is within the positive fixed angle. Fig.8(b-2)
When Shoobot pushes the ball forward, it will gradually lose the ball to left side. In this case, the ball must be shifted towards the right. Wheel angle should be set slightly larger than the ball angle.
3. The position of the ball is within the negative fixed angle. Fig.8.(b-3)
In this case, the ball may gradually roll to the right when feedback control is used. Unless it does not push the ball enough, wheel angle may be set a slightly smaller than the ball angle. With the repetition of these processes, Shoobot gradually nears the global path without confusing the direction of the rolling ball.

4.2 Dribbling the ball through the flags

We explain the task of dribbling the ball through flags and finally shooting the ball into the goal. Shoobot always has only a target on the way because Shoobot can chose the adequate path under multiple targets. It is important not to touch the flags, the next target (T_n) chosen by Shoobot is a point on the line drawn

from the present Shoobot's position to a circle around the next flag. This line to the target ($R_n T_n$) is defined as the ideal path along which keep the ball. Shoobot sets the new target (T_{n+1}, T_{n+2}) using all sensing data and its present position. This corresponds to FF control.

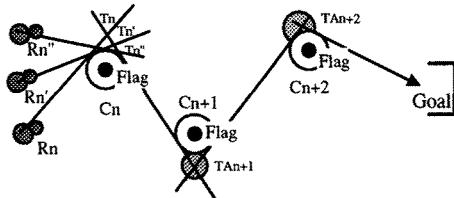


Fig. 9. Plan for dribbling

Merely by pushing the ball, it may deviate from the correct direction. Furthermore, Shoobot must return to a new path using FB control. Where the fixed target is on the path, Shoobot can dribble the ball at the speed of 0.62m per sec. After passing through a flag, Shoobot must decide a new target around the next flag.

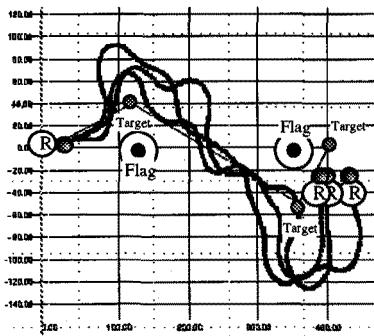


Fig. 10. Paths after dribbling the ball

5 Discussion and Summary

We found that feedforward control is required to determine the position of the space from visible information and to operate the distance and position using an internal model. If no appropriate internal model exists, Shoobot should be controlled to follow the present target by feedback control wherein Shoobot moves while maintaining the ball at a constant angle of view.

An internal model of the environment (used to determine the speed vector of the ball) enables Shoobot to predict the position of contact and to move there before the ball arrives. We have checked that athletic movement is improved through proper use of FF and FB control.

Bizzi's experimental result [Bizzi84] showed that a monkey's arm moves as if there is an original path to the target in his brain. Kawato suggested that humans control their arms efficiently using FF control with a highly sophisticated internal model [Kawato96]. These results indicate that creatures with brain may make use of internal representation, though the representation may be different from the form of the coordinates we used in our research. [Arbib89]

There is a criticism that the use of representation leads to symbolizing (see [Brooks95]). We support the notion that robots should include internal representation as indicated by the present experiments. With an internal model, the robot's movement becomes more stable and the target can be reached in a shorter time. Furthermore, Shoobot itself can also modify its path by transferring the force position represented in the internal model such as potential field in section 3.3.

We conclude that the use of an internal model enables Shoobot to set its target and to use feedforward control. As a result, the robot can move more smoothly.

References

- [Brooks95] R. A. Brooks : Intelligence without Reason. *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents*, L. Steels, R. Brooks(Ed), 25–81
- [ALIFE96] Mizuno H., M. Kurogi, Y. Kuroda, Y. Muraoka : An Architecture for Soccer Shooting Robot. ALIFE V, PP-12, May (1996)
- [IROS96] Mizuno H., M. Kurogi, Y. Muraoka : Building "Shoobot" Capable of Dribbling and Shooting. 96IROS, WS4, Pre-RoboCup, (1996)
- [Arbib89] Arbib, M. A. : *The Metaphorical Brain 2: Neural Networks and Beyond*. Wiley-Interscience, (1989)
- [Bizzi84] Bizzi, E., Accornero, N., Chapple, W., Hogan, N.: Posture Control and Trajectory Formation during Arm Movement. *The Journal of Neuroscience*, 4, 2738-2744 (1984)
- [Kawato96] Kawato, M. : *Bi-Directional Theory Approach to Consciousness. Cognition. Computation and Consciousness*. Oxford University Press, Oxford (1996)
<http://www.erato.atr.co.jp/>
- [Nomad200] Nomadic Technologies, Nomad200 User's Manual

A Legged Robot for RoboCup Based on "OPENR"

Masahiro Fujita¹, Hiroaki Kitano² and Koji Kageyama¹

¹ D21 Laboratory, Sony Corporation,
6-7-35, Kitashinagawa,
Shinagawa-ku, Tokyo, 141 JAPAN

² Sony Computer Science Laboratory Inc.
Takanawa Muse Building, 3-14-13 Higashi-gotanda,
Shinagawa-ku, Tokyo, 141 JAPAN

Abstract. We propose a physical robot platform for RoboCup based on **OPENR**, which is proposed as a standard architecture for **Robot Entertainment**, a new entertainment field devoted to autonomous robots. The platform will be provided as a complete autonomous agent so that researchers are able to start working within their research fields. It is possible to rewrite all of, or parts of, the software of the RoboCup player. The configuration of the platform is quadruped type with a color camera, a stereo microphone, a loudspeaker, and a tilt sensor. In order to reduce the computational load, we propose that the walls of the field and the player itself are painted in different colors so that it is easy to identify the position of essential objects.

1 Introduction

In this paper, we propose a physical robot platform for RoboCup. RoboCup[3] was proposed as a new standard problem for A.I. and robotics research. RoboCup aims at providing a standard task for research on fast-moving multiple robots, which collaborate to solve dynamic problems. Although building a real robot for RoboCup is an important aspect of robotics, it is a money and time-consuming task for academic researchers. Thus, researchers are eager for a physical robot platform on which it is easy to build their academic research.

Previously, we proposed **OPENR**³ as a standard architecture and interfaces for **Robot Entertainment**, which is a new entertainment field using autonomous robots[2]. OPENR aims at defining a standard whereby various physical and software components which meet the OPENR standard can be assembled with no further modifications, so that various robots can be created easily.

One of the purposes of proposing OPENR is that we wish to further promote research in intelligent robotics by providing off-the-shelf components and basic robot systems. These robots should be highly reliable and flexible, so that researchers can concentrate on the aspect of intelligence, rather than spending a substantial proportion of their time on hardware troubleshootings.

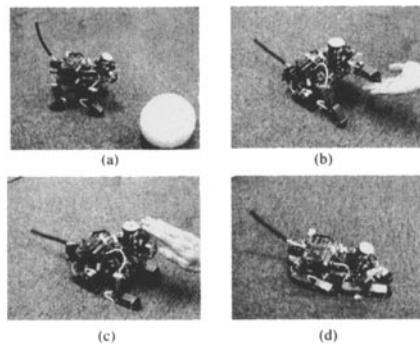


Fig. 1. MUTANT: Fully autonomous pet-type robot

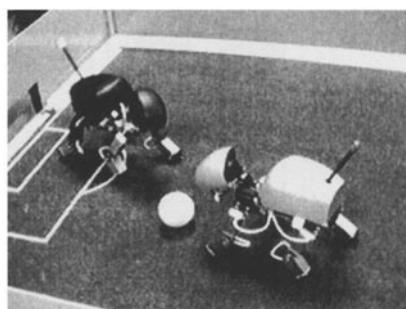


Fig. 2. Remote-operated soccer game robot

For the feasibility study of OPENR, we developed an autonomous quadruped pet-type robot named MUTANT[2] (Fig.1). This robot did not implement all ideas of OPENR, but it allowed us to test some of the software components and the agent architecture proposed in OPENR. By adding an extension module, we also implemented and tested a remote-operated robot system for soccer game (Fig.2). Using these robots, software and application feasibility studies were carried out.

As the next step, we have been developing new dedicated LSIs for the next generation prototype robot which meets the OPENR standard completely. We propose a platform for a RoboCup player based on this new prototype robot. In addition, we also propose a game environment such as a field and a ball, and some algorithms suitable for this platform.

³ **OPENR** is a trade mark of Sony Corporation.

2 OPENR

2.1 Requirements

In order for OPENR to be accepted as an open standard for entertainment robot, there are several requirements which OPENR should satisfy. We argue that four types of scalabilities are essential for such a standard. These are (1) size scalability, (2) category scalability, (3) time scalability, and (4) user expertise scalability.

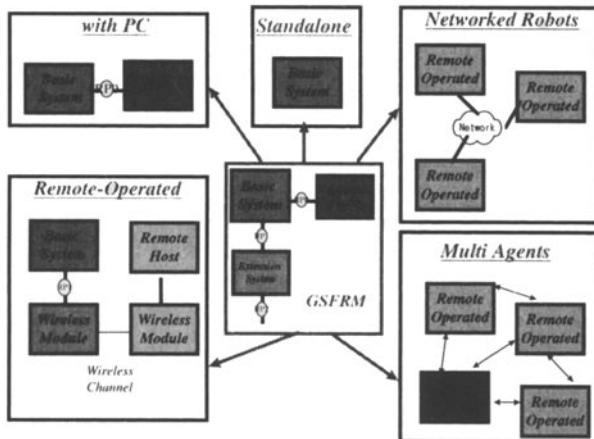


Fig. 3. Generic System Functional Reference Model and Examples of Derived System Architectures

Size Scalability (Extensibility): OPENR must be extensible for various system configuration. For example, in a minimum configuration, a robot may be composed of only few components, and perform a set of behaviors as a complete agent. This robot can be scaled up by adding additional physical components. It should also be possible to scale up such a system by having such robots as sub-systems of large robot systems.

Category Scalability (Flexibility): Category scalability ensures that various kinds of robots can be designed based on the OPENR standard. For example, two very different styles of robots, such as a wheel-based robot and a quadruped-legged robot, should be able to be described by the OPENR standard. These robots may have various sensors, such as cameras, infra-red sensors, and touch sensors and motor controllers.

Time Scalability (Upgradability): OPENR must be able to evolve together with the progress of hardware and software technologies. Thus, it must maintain a modular organization so that each component can be replaced with up-to-date modules.

User Expertise Scalability (Friendly Development Environment): OPENR must provide a development environments, both for professional developers and for end-users who do not have technical knowledge. End users may develop or compose their own programs using the development environment. Thus, it is scalable in terms of the level of expertise that designers of the robot have.

2.2 OPENR Strategy

Our strategy to meet these requirements consists of the following:

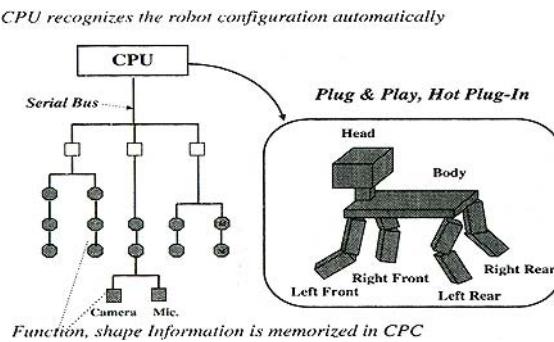


Fig. 4. Configurable Physical Component

Generic Reference Model: To meet the requirements of Extensibility and Friendly Development Environment, we define a generic system functional reference model (GSFRM) composed of Basic System, Extension System and Development System. By defining GSFRM, we are able to construct various kinds of robot systems with extensibility and development environments, as shown in Fig.3.

Configurable Physical Component: To meet the requirements of Flexibility and Extensibility, we devise a new idea of Configurable Physical Component (CPC). The physical connection between the robot components is done by a serial bus. In addition every CPC has non-volatile memory with (1) functional properties, such as an actuator and a two dimensional image sensor, and (2) physical properties, which help solve the dynamics of the robot consisting of these CPCs.

Object-Oriented Programming: To meet the requirements of Up-gradability and Flexibility, we employ an object-oriented OS, **Aperios** [6], which supports the Object-Oriented Programming (OOP) paradigm from the system

level with several types of message passing among objects. In addition, Aperios is capable of customizing APIs by system designers.

Layering: To meet the requirements of Up-gradability and Friendly Development Environment, we utilize the layering technique which is often used for multi-vendor open architecture. OPENR divides each functional element into three layers, Hardware Abstraction Layer (HAL), System Service Layer (SSL), and Application Layer (APL), as shown in Fig.5

In order to achieve the software component concept, an Agent Architecture for entertainment application was studied. The details is described in [2].

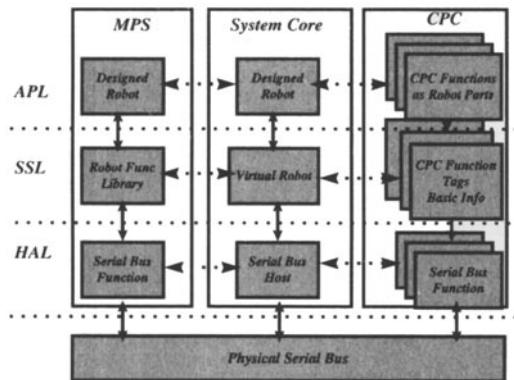


Fig. 5. Layering for Basic System

3 Proposed RoboCup System

3.1 Basic Behavior of RoboCup Player

Although RoboCup includes multi-agent co-operation and team play problems, at the primary stage, let us consider an individual player's behavior. The typical action sequence in this situation is (1) to find the ball, (2) to get close to the ball, (3) to kick or to dribble the ball toward the opponent goal, (4) to shoot to get a goal.

Thus, in this simple case the RoboCup player must somehow handle the position information of the following objects:

1. the ball,
2. the opponent goal,
3. the player itself.

3.2 Proposed RoboCup System

In general, it is difficult for a standalone robot system to perform these tasks in real time in a real world environment because of its limited computational power. The remote operated robot system depicted in Fig.3 can solve the computational power problem; however, in general, much computational power is necessary for image processing tasks. This implies that it is necessary for each robot to be equipped with a video transmitter. This is sometimes difficult for regulation reason.

Another solution to this problem is to set up a camera overlooking the entire field, and to distribute the image signal to all host computers[1]. In this RoboCup architecture, each robot can use the huge computer power of the host computer, or a special engine such as image processing hardware. However, the image information taken by the overlooking camera is not the image taken from each robot's viewpoint.

We consider that technologies to process the image from each robot viewpoint without any global information will become very important in Robot Entertainment in future. Therefore, we decide to build RoboCup System with standaolone robots under local communication constraint.

To reduce the computational load for the standalone robot, we propose to utilize color processing to identify the objects listed above. In other words, these object must be painted with different colors with which RoboCup player can easily identify the objects. Fig.6 and Fig.7 show examples of the painting of a field and a player.

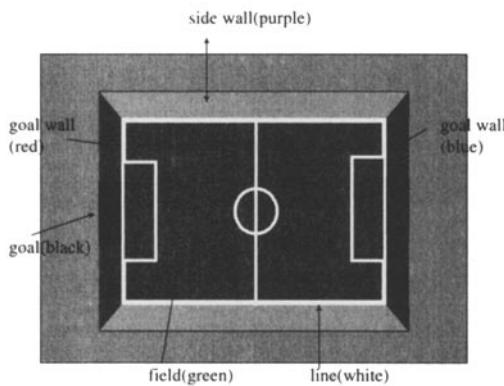


Fig. 6. Color Painted Field

We still need much computational power for the proposed color-painted RoboCup System to process the color image sequences and other tasks. There-

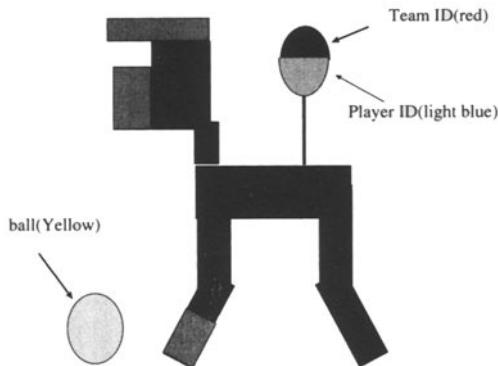


Fig. 7. Color Painted Player and Ball

fore, we developed a dedicated image processing engine to provide the required computational power. The details are described below.

It should be noted here that this field does not satisfy the RoboCup regulation defined by [4]. Furthermore, the size of the proposed robot described below does not satisfy the regulation, either.

4 Physical Robot Platform

4.1 Mechanical Configuration and Sensors

Fig.8 shows a mechanical configuration of the proposed platform. The features are as follows:

Robot Category: A quadruped legged walking robot.

Size and Weight: The size of the robot is about $110mm$ (length) $\times 90mm$ (width) $\times 110mm$ (height). The weight of the entire robot is about $1.3kg$.

Degree of Freedom (DoF): Three DoF for each leg and three DoF for neck and head.

Actuators: Each joint has a DC Geared Servo Motor which is composed of a DC coreless motor, gears, a motor driver, a potentiometer, and a feedback controller. The torque of the geared motor is $4.2kgcm$ for the leg joints and $1.2kgcm$ for the neck-head joints. Each leg is designed to meet OPENR Configurable Physical Component (CPC) standard.

Image Sensor: A micro color CCD camera consisting of a lens module, a driver IC, a timing generator, A/D converter, and micro-controller is mounted on the head. The output signal format is 4:2:2 digital YUV. The image signal is downsampled into 360×240 pixel resolution. The control signal is available through OPENR serial bus.

Microphone and Loudspeaker: A stereo microphone is mounted on the head, and the stereo signal is converted into the stereo digital 16bit with 32kHz sampling frequency. A monaural loudspeaker is mounted on the head, and 16bit DA converter through a speaker amplifier drives the loudspeaker. The audio module is connected to the OPENR serial bus.

Posture Sensor: Three acceleration sensors are mounted in the body to estimate the gravitation direction. These signals are AD-converted into 10 bit data with about 120Hz sampling frequency. This sensor module is connected to the OPENR serial bus.

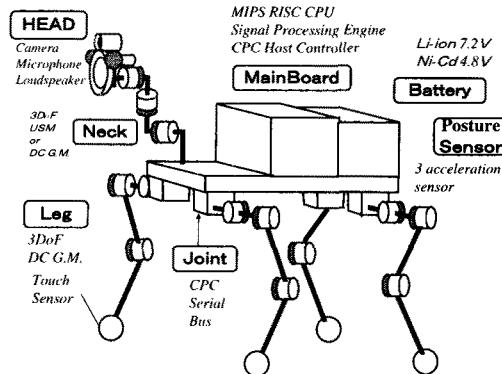


Fig. 8. Mechanical Configuration and Sensors

4.2 Hardware Architecture

The electronic hardware architecture is shown in Fig.9. The features are listed as follows:

CPU: A 64bit MIPS RISC CPU with 100MHz pipeline clock. The performance of the CPU is 133 MIPS.

Memory: 8MBytes SDRAM with 66MHz clock and 1MBytes Flash Memory. As Media for Program Storage in OPENR, we employ PCMCIA memory card.

ASIC-1:

Peripherals: Peripherals such as SDRAM controller, timer, and DMAC are integrated in a dedicated LSI for this platform. We pay attention to the bandwidth of the system bus because huge amounts of data, including the image signal is read from and written to the SDRAM. We design the DMAC in order for the CPU to be free from just copying the data.

Signal Processing Engines: This LSI includes a three layer multi-resolution filter bank for the CCD camera image, a color detection engine with eight color properties in YUV coordinate, an inner product engine for fast image processing, and a 16bit integer DSP with 50MHz for audio signal processing.

CPC Host Controller: We also include a host controller for OPENR CPC serial bus. All sensors are physically connected to this serial bus. (The image signal from the CCD camera is an exception.)

ASIC-2: Each component of the above CPC includes another dedicated LSI, the OPENR serial bus functional device. This LSI includes a single channel feedback controller for the DC servo motor, an interface for 16bit AD-DA converter, and four channel 10bit AD converter for general sensors. Each LSI has an EPROM for CPC information storage. This LSI works as a three port HUB for branching the serial signal stream.

Battery: A 7.2V Li-ion battery is utilized for the electrical circuits. Almost all electrical devices are operated with 3.3V, which is supplied via a DC-DC converter. A 4.8V NiCa battery is utilized for motor power. Since the current required for the motors is higher than the current limitation of the Li-ion battery, we separated the power supply into two batteries. Using these batteries the physical robot works for about 30 minutes, which is an acceptable length for the RoboCup game.

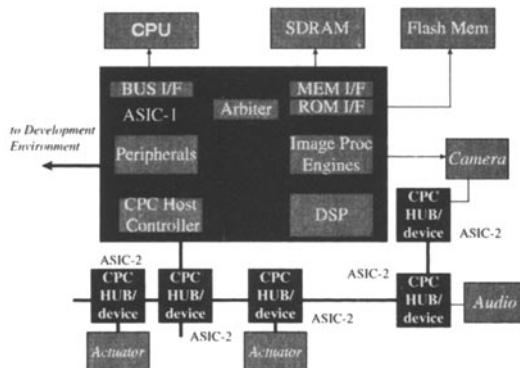


Fig. 9. Electronic Block Diagram

4.3 Software Architecture

We employ an object-oriented OS, Aperios[6], for the following reasons:

Object-Oriented Programming: Aperios supports the OOP paradigm from the system level. For communications among objects, several types of message passings such as asynchronous, synchronous, and periodic message passings.

Dynamic Loading: Originally Aperios supports an object migration protocol to accommodate heterogeneity. It is possible for the OPENR to use this migration mechanism in order to download application objects from the Development Environment to the robot Basic System. The downloading can be executed when the main application program is running.

In addition, we build some objects, such as the Virtual Robot and the Designed Robot defined in OPENR (Fig.10). The Virtual Robot supports APIs for communication with each CPC device, and the Designed Robot supports APIs for communication with a part of the robot with semantics such as left-front leg and head.

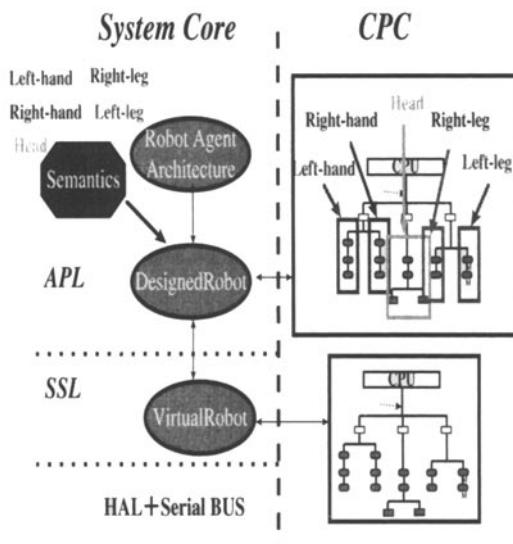


Fig. 10. Virtual Robot and Designed Robot

5 Algorithm Examples

In this section, we describe some examples of algorithms for (1) ball finding and kicking and (2) player positioning. These two algorithms are just examples, and

it depends on the strategy whether the above information is explicitly calculated or not. For example, kicking the ball may be considered as the hand-eye coordination problem[5]. Most of the algorithms have the explicit representation of the object position, however, if the sensor data has sufficient information to estimate the ball position, we can solve this problem without explicit representation of the ball position.

5.1 Ball Finding and Kicking

Since we define the colors of object exclusively, yellow color is assigned only for the ball (a yellow tennis ball is assumed to be used). In this case, to find the ball we just need to find a yellow object. To keep the ball in the center of the image, the robot head direction is feedback controlled and the walking direction is also controlled using the horizontal head direction.

Both the horizontal and vertical neck angle can be utilized to estimate the distance between the ball and the robot body (Fig.11). This information is also used as a queue to execute a kick motion.

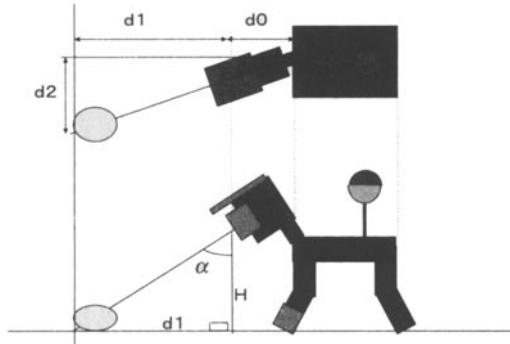


Fig. 11. Ball Positioning

5.2 Player Positioning

Here, the position means the two dimensional absolute position in the field, and we assume that the robot knows the absolute size of the field. It may not be necessary to measure the absolute position, however, it is easy to modify the algorithm into the relative position with relative information, namely, the proportion ratio of the width, length and height of the field, the wall, or the robot itself.

Assume that the robot knows H , the height of its camera view point from the ground. Then, it is almost the same problem as the estimation the distance between the ball and the robot.

1. To find the opponent goal side with the goal color,
2. to measure the neck's vertical angle along the edge of the downside of the wall,
3. to find the point that gives the minimum angle value x ,
4. $H \tan^{-1}(x)$ is the distance from the wall.

Repeating the same estimation of the distance to the side wall, the robot can get the two-dimensional position in the field.

There are many algorithms to estimate the robot position in the field, and it depends on the strategy of the game whether the accuracy of this information is important or not.

6 Conclusion

We proposed and developed a physical robot platform for RoboCup based on OPENR. The proposed RoboCup System, which is composed of the color painted field, the ball, and the robot players, does not satisfy the regulation defined by RoboCup Committee[4]. However, by assuming the color painted world, we can use the computational power for other tasks such as corporation and team play strategy.

In this paper we didn't describe the communication among robot players. We think that it is possible for this robot platform to communicate each other using the loudspeaker and the stereo mic. At least, signals from referee, such as a whistle, can be utilized as queues for the start of the game, and the end of the game, and so on.

In this paper we didn't describe the software development environment. This issue is very important for Robot Entertainment System, and as of the time of writing this paper, it is still under development.

OPENR will be able to supply more useful mechanical and electrical hardware and software components so that many researchers can share and test their ideas, not only for RoboCup but also for Robot Entertainment and for an Intelligent Robot Society.

7 Acknowledgement

The implementations of the proposed RoboCup System and robot platform are being performed by the members of Group 1, D21 laboratory. The MCM camera is developed by Sony Semiconductor Company, and Aperios is developed by Architecture Laboratory, Sony Corporation.

Authors would like to thank to Dr. Tani at Sony Computer Science Laboratory for his advice to the hand-eye coordination problem.

References

1. Achim, S. and Stone, P. and Veloso, M.: Building a Dedicated Robotic Soccer System. Proceedings of IROS-96 Workshop on RoboCup. (1996) 41–48
2. Fujita, M. and Kageyama, K.: An Open Architecture for Robot Entertainment. Proceeding of the 1st International Conference on Autonomous Agents. (1997) 435–440
3. Kitano, H. : RoboCup: The Robot World Cup Initiative. Proceedings of International Joint Conference on Artificial Intelligence: Workshop on Entertainment and AI/ALife. (1995)
4. Kitano, H. and Asada, M. and Kuniyoshi, Y. and Noda, I. and Osawa, E.: RoboCup: The Robot World Cup Initiative. Proceedings of the First International Conference on Autonomous Agents. (1997) 340–347
5. Smagt, P. van der and Groen, F.C.A. and Kroese, B.J.A.: A monocular robot arm can be neurally positioned. Proceedings of the 1995 International Conference of Intelligent Autonomous Systems. (1995) 123–130
6. Yokote, Y.: The Apertos Reflective Operating System: The Concept and Its Implementation. Proceeding of the 1992 International Conference of Object-Oriented Programming, System, Languages, and Applications. (1992)

JavaSoccer

Tucker Balch

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-208 USA

Abstract. Hardware-only development of complex robot behavior is often slow because robot experiments are time-consuming and robotic hardware is subject to failure. The problem is exacerbated in multi-robot tasks like robot soccer where successful experiments require several robots to operate correctly simultaneously. Robotics research proceeds more quickly through a cycle of experimentation in simulation and implementation on hardware than through hardware-only implementation. To facilitate research in the RoboCup soccer domain we introduce JavaSoccer, a portable system supporting development of multi-robot control systems in simulation and, potentially, on soccer robots. In simulation, JavaSoccer emulates the dynamics and dimensions of a regulation RoboCup small size robot league game.

1 Introduction

Soccer is an increasingly active domain for mobile multiagent robotics research. The soccer task is simple and familiar to most people, yet provides opportunities for diversity and cooperation in the individual team members [5, 3, 2]. No matter the domain, multi-robot team design is challenging because success depends on coordination between complex machines. Since robot hardware is subject to frequent failure and experimentation is time consuming, hardware-only robot development is difficult.

To address these issues many researchers turn to a cycle of simulation and robot experiments to develop their systems. Behaviors are prototyped in simulation, then implemented on robots. Even though further refinement of the control system is often required for use on robots, simulation can rapidly accelerate control system development. To facilitate experimentation in RoboCup soccer, we introduce JavaSoccer, a dynamic simulation of RoboCup small size league play.

JavaSoccer is part of the JavaBots software distribution available from the Mobile Robot Laboratory at Georgia Tech. JavaBots is a new system for developing and executing control systems on mobile robots and in simulation. Individual and multi-robot simulations are supported, including multiple robot types simultaneously. The JavaBot system's modular and object-oriented design enables researchers to easily integrate sensors, machine learning and hardware with robot architectures. Additionally, since JavaBots is coded in Java, it is easily ported to many operating environments. To date it has run under Windows 95, Linux, SunOS, Solaris and Irix without modification.

The JavaBot system is being utilized in several ongoing research efforts at Georgia Tech, including RoboCup robot soccer, foraging robot teams and a “robotic pet” development effort. At present, control systems developed in JavaBots can run in simulation and on Nomad 150 robots. Hardware support will soon be provided for ISR Pebbles as well. In the future we hope to expand this capability to small soccer robots, so that behaviors tested in the JavaSoccer simulation will run directly on hardware.

Another important soccer simulator, the SoccerServer developed by Noda is used in simulation competition at RoboCup events [6]. While JavaSoccer is focused on simulation and control of existing, physical robots, SoccerServer is targeted at simulation of more futuristic, human-like players. The SoccerServer is coded in C++ and runs in most Unix/X environments. JavaSoccer also runs in Unix/X environments, but it benefits from the high degree of portability afforded by Java, allowing it to run in Windows 95 and NT machines as well.

The remainder of this chapter will describe JavaSoccer in detail, including the simulation kernel, JavaSoccer rules, robot capabilities and directions for future work.

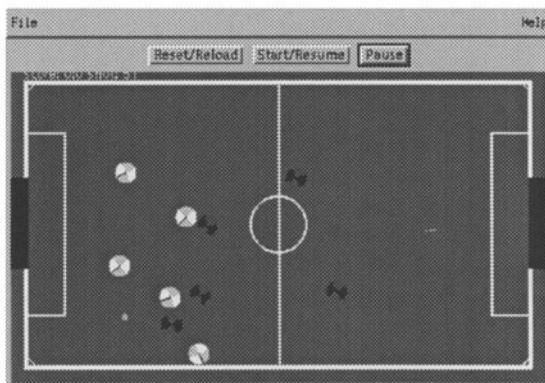


Fig. 1. A JavaSoccer game in progress.

2 The Simulation System

JavaSoccer runs in the JavaBotSim simulator distributed with JavaBots. At runtime, the simulator reads in a description file that describes which objects will be present in the simulation and where they should appear in the environment. Obstacles, small soccer robots, Nomad 150 robots, golf balls and “squiggle” balls have all been implemented to date. A description file appropriate to soccer simulation is included with JavaBots in the JavaSoccer directory (`robocup.dsc`). Simple modifications to the description file enable users to change the robot type and control systems used as well as adjust the color scheme of the robots.

Each object in the simulation includes two important components: a drawing method and a dynamics simulation method. The simulation kernel, or core, runs in a loop, calling the two methods for all the objects in turn at each step. The kernel also keeps track of time, and passes elapsed time to the dynamics methods so they know how far to move their associated simulated objects. The simulation can run in real-time or faster or slower than real-time according the `time` parameter set in the description file.

In addition to the drawing and dynamics methods, each robot has an associated control system object that considers sensor inputs and makes actuator outputs for the robot. A soccer robot control system is coded in Java by extending the `ControlSystemSS` class (`SS` stands for Small Soccer robot). Several example control systems are provided with the software. The intent is for these examples to be modified by researchers for the purpose of testing their own soccer strategies.

Since the API between the control system and a simulated robot is the same as for actual hardware, the same control algorithm can be tested in simulation and in real robot runs. The capability for running soccer robot control systems on small robot hardware is not available yet, but users can develop and run systems for Nomad 150 robots.

3 RoboCup Rules in JavaSoccer

Rules for JavaSoccer play are based on RoboCup's regulations for the small size robot league [4]. Some differences are necessary in JavaSoccer due to the simulation environment. Any differences are noted below:

Field dimensions The field is 152.5cm by 274cm. Each goal is 50cm wide. The defense zone, centered on the goal, is 100cm wide by 22.5cm deep.

Defense zone A defense zone surrounds the goal on each side. It is 22.5cm from the goal line and 100cm wide. Only one defense robot can enter this area. Note: this is not enforced by JavaSoccer, but it will be eventually.

Coordinate system The center of the field is $(0, 0)$ with $+x$ to the right (east) and $+y$ up (north). The team that starts on the left side is called the "west" team and the team on the right is called the "east" team. At the beginning of play, the ball is placed at $(0, 0)$.

Robots Robots are circular and 12cm in diameter. In RoboCup rules 15cm is the maximum size. The simulated robots can move at 0.3 meters/sec and turn at 360 degrees/sec.

Team A team consists of no more than 5 robots.

Ball The simulated ball represents an orange golf ball. It is 40mm in diameter and it can move at a maximum of 0.5 meters/sec when kicked. It decelerates at 0.1 meters/sec/sec. Ball collisions with walls and robots are perfectly elastic.

Field Coloring Coloring of the field doesn't really matter in the simulation, but the image drawn on the screen matches the RoboCup regulations.

Robot marking Robots are marked with a two-color checkerboard pattern. In RoboCup rules they are marked with two colored ping-pong balls.

Length of the game At present, once the game starts, it runs forever. This will be changed to conform with the 10 minute halves of RoboCup.

Wall A wall is placed around all the field, except the goals. Ball collisions with walls are perfectly elastic. Robots may slide along walls if they aren't headed directly into them.

Communication There is presently no facility for robot to robot communication, but this may be added if researchers are interested in the capability.

Goal keepers At present, the goal keeper has no means for grasping the ball. This feature may be added later.

Kick-off/Restart/Stop Kick-off, restart and stop of the game are managed automatically. The west team gets to kick-off first, with subsequent kick-offs made by the scored-against team.

Robot positions at kick-off The robots are automatically positioned as shown in Figure 2 at kick-off. The numbers next to the players indicate values returned by the `getPlayerNumber()` method. Specific positions are not specified in the RoboCup rules, except that only the kick-off team can be within 15cm of the ball.

Fouls/Charging Currently no fouls are enforced by JavaSoccer. There is however a "shot clock." At the beginning of a point, the clock is set to count down from 60 seconds. If no score occurs in that time, the ball is returned to the center of the field. The ball is also returned to the center if it gets stuck for a period of time. Those types of fouls that can be evaluated objectively by the running program will eventually be added. RoboCup rules include several fouls and charging violations.

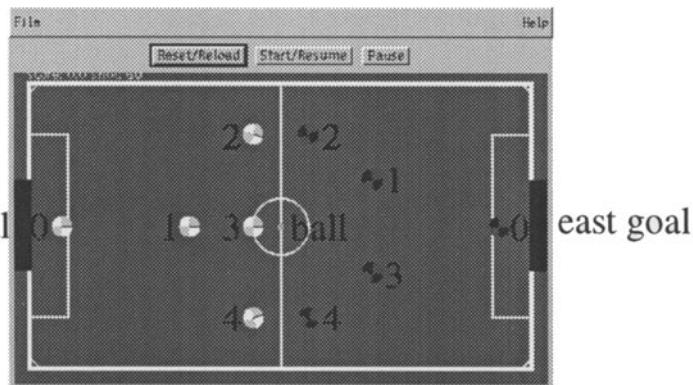


Fig. 2. Starting positions for a JavaSoccer kickoff. The west team is kicking off.

4 Capabilities of a JavaSoccer Robot

The interface between the robot's control system or "brain," and its sensors and actuators is provided by an API defined in the `SocSmall` Java class. The most complete description of the interface is in the javadoc-generated documentation for `SocSmall`, and `SocSmall`'s parent class, `Simple`, but a synopsis is provided here:

— Sensors

- Detect whether the robot is in a position to kick the ball.
- Get a vector pointing to the ball.
- Get a vector pointing to the team's goal.
- Get a vector pointing to the opponents' goal.
- Get an array of vectors pointing to all the other players.
- Get an array of vectors pointing just to the robot's teammates.
- Get an array of vectors pointing just to the robot's opponents.
- Get the player's number on the team (0 - 4), 0 is the goalie.
- Get the player's position on the field.
- Get the player's heading.
- Get the time in milliseconds since the game started.

— Actuators

- Kick the ball at 0.5 meters/sec.
- Push the ball by driving over it.
- Set desired heading. The robot can turn at 360 degrees/sec.
- Set desired speed, up to 0.3 meters/sec.

Each of these capabilities is implemented as a method, or function call, in the `SocSmall` class.

5 Developing New Team Behaviors

New team behaviors are implemented by extending the `ControlSystemSS` class. The easiest way for a developer to accomplish this is by revising of one of the existing teams provided with JavaSoccer in the `JavaSoccer/teams` directory. Some of the example teams were developed using Clay, a behavior-based configuration system [1]. While developers may find Clay helpful in designing their strategies, it is not required.

Two methods must be implemented to complete a control system: `Configure()` and `TakeStep()`. `Configure` is called once at set-up time to provide for initialization and configuration of the control system. `TakeStep()` is called repeatedly by the simulation kernel to allow sensor polling and action selection. Sensors are read by calls to `abstract_robot.get*` methods, actuator commands are sent by calls to `abstract_robot.set*` methods.

Sensor example:

```
abstract_robot.getBall(curr_time)
```

reads the ball sensor and returns a 2-dimensional vector towards it. The `curr_time` parameter is a timestamp that helps prevent redundant sensor accesses (this is more important for use on real robots than in simulation). All other sensor methods are similar in form.

Actuator example:

```
abstract_robot.setSteerHeading(curr_time,0)
```

In this call, the second argument is the direction for the robot to head, in radians. So the robot will steer in the east, or $+x$, direction. Readers are invited to look at the example teams for more detail on sensor and actuator methods.

6 Simulating New Hardware

One of the advantages of the JavaBot system is the ease with which new objects and robot types may be added to the simulation system. It is a fairly simple matter to revise an existing class to emulate new robot hardware. For instance, to create a new, larger simulated robot with the same capabilities as a `SocSmall`, one need only extend the `SocSmall` class and revise the `RADIUS` variable. Other more complex changes are still straightforward, the developer need only ensure that all the methods defined by the `SimulatedObject` interface are implemented.

7 Installing and Running

The entire JavaBots distribution, including JavaSoccer is available as a “zip” file on the world-wide-web. Installation entails downloading the distribution and un-zipping it. Instructions are available at

<http://www.cc.gatech.edu/~tucker/JavaBots>

8 Future Directions

Our goal in building JavaBots and JavaSoccer is to accelerate the development of behaviors for physically-embodied robots by supporting identical control system code in simulation and on robot hardware. At present, for small soccer robots, the control systems are only supported in simulation. An important next step will be to implement the API for small mobile robot hardware. This has already been demonstrated in JavaBots for Nomad 150 robots.

Other potential improvements include: closer conformance to RoboCup regulations, simulation of several robot types actually used in RoboCup competition, and new simulation features such as robot position traces and player number display.

9 Acknowledgments

The author is indebted to Ron Arkin, director of the Mobile Robot Laboratory for access to computer and robot hardware used in the development of JavaBots. Itsuki Noda's SoccerServer provided an excellent example soccer simulation system [6]. I also wish to thank Peter Stone, Manuela Veloso, Hiroaki Kitano and Maria Hybinette for their advice in this endeavor.

References

1. T. Balch. Clay: Integrating motor schemas and reinforcement learning. College of Computing Technical Report GIT-CC-97-11, Georgia Institute of Technology, Atlanta, Georgia, March 1997.
2. T. Balch. Social entropy: a new metric for learning multi-robot teams. In *Proc. 10th International FLAIRS Conference (FLAIRS-97)*, May 1997.
3. Tucker Balch. Learning roles: Behavioral diversity in robot teams. In *AAAI-97 Workshop on Multiagent Learning*, Palo Alto, CA, 1997. AAAI.
4. RoboCup Committee. Robocup real robot league regulations. <http://www.csl.sony.co.jp/person/kitano/RoboCup/rule.html>, 1997.
5. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proc. Autonomous Agents 97*. ACM, 1997. Marina Del Rey, California.
6. Itsuki Noda. Soccer server: a simulator for robocup. In *JSAI AI-Symposium 95: Special Session on RoboCup*, 1995.

RoboCup-3D: The Construction of Intelligent Navigation System

Atsushi Shinjoh^{1,2}

kaminari@iamas.ac.jp

¹ International Academy of Media Arts and Sciences,
3-95, Ryoke-cho, Ogaki-city, Gifu 503, Japan.

² Softopia Japan, 4-1-7, Kagano, Ogaki-city, Gifu 503, Japan.

Abstract. RoboCup-3D is the attempt to build a virtual three-dimensional viewer for the RoboCup software league. **Intelligent Navigation System(INS)** is a basic concept of this RoboCup-3D, and it aims at creating a system which assists and navigates human cognition and activity in virtual three-dimensional space. Based on INS, we built **SPACE**, which is the first three-dimensional viewer for the RoboCup software league.

In this paper, we are proposing the basic concept of this research and will show the possibility and the aims of future research of INS based on the analysis of **SPACE**.

1 Introduction

Recently, research and development about the virtual three-dimensional world is proceeding on the computer because of the improvement of the computer's ability. The progress of the hardware has made the real-time rendering process of three-dimensional objects on a personal computer. In addition to these factors, programming languages which are suitable for building three-dimensional worlds such as Virtual Reality Modeling Language(VRML)[11], Java3D[10] and OpenGL[8] are being made. Based on these trends, some projects on the practical use of virtual three-dimensional space are being done such as the cyber mall³[11], multi user communication systems[9], and cyber showroom⁴. In spite of these activities, most of these projects have not succeeded today. Because of these present conditions, there are many people who doubt the possibility of virtual three-dimensional space[2].

There are two problems in these projects. One is the problem of the lack of reality. A virtual space which has realty and rich content does not exist except in entertainment games and movies. Compared with the above projects, these contents are not an imitation of the real world, but the original of each virtual world. Furthermore, to create this content needs a lot of time, money and the excellent skills for creating virtual three-dimensional objects today. Part of this

³ The cyber mall is the 3D shopping mall on the net.

⁴ <http://vrml.sgi.com/>

problem will never solved at all, because the construction depends on human's skill and talent, and doesn't depend on its effort. However, the rest part of this problem will be solved by the development of hardware and software which will assist in the construction of three-dimensional objects. The other is the problem of human's cognition in the virtual three-dimensional world. In the real world, human-beings usually can afford information from the objects based on its experience.[3] Human-beings also will be able to use its body(such as hands, legs, eyes, ears, and other parts) and its several senses for its perceptions. Based on these perceptions, human-beings form its experiences. In the virtual three-dimensional world, humans can only use their hands for operating the keyboard and some pointing devices, and cannot use its senses. Furthermore, the virtual three-dimensional world only exists in two-dimensional displays, and human-beings never use its experiences. For these reasons, most of the virtual three-dimensional worlds are not suitable for human's cognitions. Some entertainment games have coped with this problem by creating and using original devices. Some research in the study fields of virtual reality have also tried to cope with this problem in the same ways. However, cognition of the virtual three-dimensional world is still difficult, because these approaches only deal with the method of navigation, and do not deal with the whole navigation system. Because the computer display separates the virtual three-dimensional world and the real world, human-beings never use the real world's cognition. This means that the method which assists human cognition is needed for coping with this problem.

For solving and coping with this problem, we propose the basic concept called **Intelligent Navigation System(INS)** which is the basic concept and aims at creating a system which assists and navigates human cognition and activity in the virtual three-dimensional space. For developing the research of **INS**, we will describe the research of RoboCup-3D, which aims at making a three-dimensional viewer for the RoboCup software league, because **INS** is the main concept of this research.

2 Intelligent Navigation System

Intelligent Navigation System(INS) is the basic concept of RoboCup-3D which includes some research areas such as artificial intelligence, artificial life, and virtual reality. This kind of research such as adaptive autonomous agents has been done[1][7]. However, because this research was separated from the real world's objects, there are no objects which we could compare with this kind of agent. Because we developed this research based on the research of RoboCup-3D, **INS** can be compared with an actual soccer game. The soccer game is one of the most popular sports in the world, and we can analyze and evaluate this system based on the analysis of several different relays of soccer games⁵.

⁵ In fact, the method of the relay of soccer game is quit different in each country.

2.1 RoboCup-3D:three dimensional viewer - SPACE

Two leagues of real and software exist in RoboCup[4][5]. We have created a first three-dimensional viewer for **MARS** based on the analysis of the relay of a soccer game in the real world.[6]⁶. Because **MARS** has a two-dimensional viewer as a standard viewer, this three-dimensional viewer is an optional viewer for **MARS** now. We named this three-dimensional viewer **SPACE**.

Specifications. We used OpenInventor as a development tool of OpenGL. OpenGL is the graphic library for creating the three-dimensional virtual environment developed by SGI. Though Windows NT and SGI(IRIX) are the platforms of OpenInventor, there are some differences in these two versions. For this reason, **SPACE** runs only on SGI machines.

The following are lists of necessary environments for **SPACE**.

For development.

- IRIS Development Option 5.3
- C++ Translator 4.0
- ProDev/Workshop 2.5.1
- OpenInventor2.0.1

For execute only.

- OpenInventor execution only environment⁷.

Fig. 1 shows the screen shot of **SPACE**. The left window is a bird's-eye view. This left window usually show the whole field's conditions, and not moved automatically. However, the operator can operate this movement by hand. The right window is a camera view which moves automatically, and the operator cannot operate this movement.

Points of the creation. Following is the list of points which became problems.

1. The ratio of the soccer field to the soccer player:

In the real world, the ratio of the soccer field and the soccer player is very big. When we use the real world's ratio in **SPACE**, we will never be able to look at the state of the RoboCup game in detail. A two-dimensional viewer, which is attached to **MARS**, uses the original ratio, which is different from the real world. Though this change directly influences the total impression of the viewer, we decided the ration based on the total impression of the scene.

⁶ MARS is the server for RoboCup software league created by ETL.

⁷ This is a free software, and you can download this from SGI home page.



Fig. 1. Three-dimensional viewer - **SPACE**

2. The reality of the object:

Texture mapping is the typical method for describing the object in virtual three-dimensional world. However, this method needs a lot of data processing power. When we use texture mapping for describing each object, the total speed of **SPACE** decreased excessively. As a result, **SPACE** cannot describe the whole conditions of the soccer field at each time. Based on these results, we adopted the coloring of the polygon instead of texture mapping. Fig. 2 shows the soccer field. The right one is described by the texture mapping, and the left one is described by the coloring of the polygon. Compared with texture mapping, the coloring of the polygon doesn't need much power. However, this expression does not have a good appearance.

The motion of the player is also an important thing. We created two motions for 'Kick' and 'Run', at first, and we created the detailed motions, and applied these data to **SPACE**. Unfortunately, when **SPACE** used these motions, the total speed of **SPACE** went down dramatically. As a result, the motion of the soccer player in the present **SPACE** is very simple⁸

3. Camera:

There are five cameras on the soccer field. Two cameras are arranged at the back of each soccer goal. The other camera is arranged at the top of the field. The screen shot which is displayed in the right window of **SPACE** is only provided by one of these cameras. The position of each camera is fixed. As mentioned above, in general, the operation of the camera in virtual three-dimensional space is very difficult, because the operator must catch the soccer ball based on the information which shows on the two-dimensional screen.

⁸ At first, we created 30 flames per each motion such as 'Kick' and 'Run'. Finally, however, we used only 4 flames per each motion.

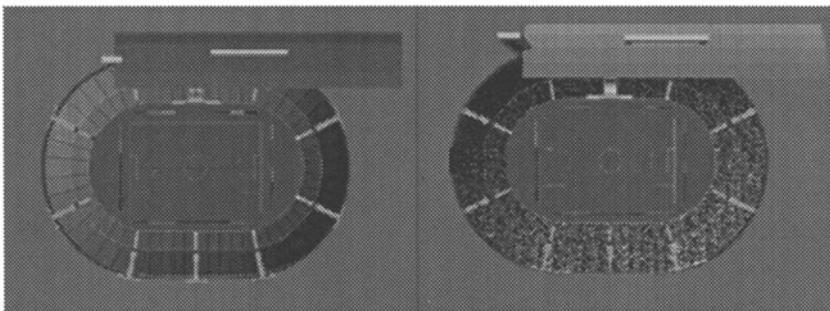


Fig. 2. The soccer field.

When the operator loses the soccer ball, the operator must look for the ball based on this information. For assisting this operation, we created two functions in **SPACE**. One is the function of the camera. Each camera always follows the ball automatically, and the camera which provides the scene to the right window is switched automatically. Thus, basically, **SPACE** always displayed the scene of the field in detail automatically. This is a fundamental method of **INS**. The other is the use of a bird's-eye view. The operator can get the whole condition of the field from this left window(See Fig.1). Basically this window provides the some information as the two-dimensional viewer. However, the operator can operate this camera for its purpose, and get the information which it wants. Because the position of this camera which provides the scene of this bird's-eye window is always fixed, the operator can only use some functions such as 'Zooming' and 'Targeting'. Fig. 3 is the sample of this function.

3 Discussion

3.1 Comparison the two-dimensional viewer and Space

We must compare the two-dimensional viewer and **SPACE** for answering the following question: 'What kind of advantage does **SPACE** have in RoboCup?'. MARS has a two-dimensional viewer by default. Because the present RoboCup software league only deals with the two-dimensional field, the movement of the soccer ball is two-dimensional. As a result, all data which exists on the field can be shown in this two-dimensional viewer. In fact, the movement of the soccer ball in **SPACE** is two dimensional.

A two-dimensional viewer has the following advantage in comparison with a three-dimensional viewer.

- **The necessary calculation is small.**

Compared with the three-dimensional world, the amount of calculation in

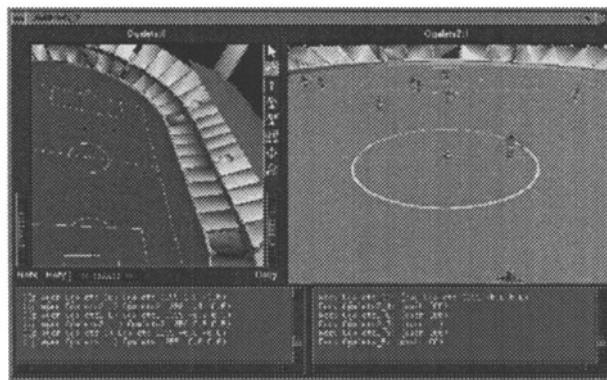


Fig. 3. Space(when the operator operate the left window by hand.)

the two-dimensionoanl world is obviously small. Since the calculation ability of today's computer is limited, this difference is important for describing the field's information.

- **The operator can see the whole of the field's information easily.** Because the three-dimensioanl viewer cannot show the whole of the field's information in detail at the same time, the operator cannot catch the movements of soccer players easily. When the operator wants to analyze movements of soccer players, the two-dimensional viewer is more suitable than the three-dimensional viewer.

For the operator, the two-dimensional viewer is most suitable because the two-dimensional viewer is only a data showing system, and does not have any extra functions. Compared with this, the three-dimensional viewer must fascinate the audience by its own created scene. This is exactly the main purpose of **INS**. **INS** and the three-dimensional viewer is not a system for the operator, but a system for the audience.

Furthermore, the above two disadvantages do not show the limitation of the virtual three-dimensional viewer. The first one will be solved by the development of hardware and software, such as OpenGL optimizer, or next generation's VRML. The last one will be solved by the research of **INS**. When these problems are solved, the three-dimensional viewer will be a most suitable system for the operator, and will be a standard viewer for **MARS**.

3.2 Points of constructing INS

Based on points described in the second chapter, we put together the problems as follows.

Implementation:

This point is based on No.1 and No.2 in the second chapter. I will put the problems in order as follows.

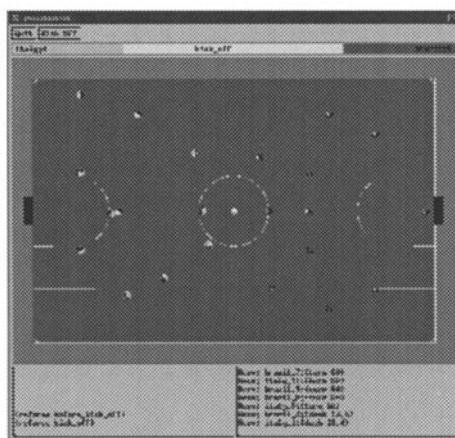


Fig. 4. The two-dimensional viewer

(a):The law.

This is the problem about relations between the real world and the virtual three-dimensional world. This problem has a relationship with No.1 in the second chapter especially. While the real world has restrictions according to the laws of nature, a conclusive law does not exist at all in the virtual three-dimensional world. When the world's constructor does not apply the laws of nature in its constructed virtual world, the operator cannot apply its common sense to this virtual world. This kind of world's construction may lead to confusion for the operator. Therefore, the virtual world needs some kind of the law of nature. In fact, some programming language which is suitable for constructing the virtual world supported this kind of function. For example, VRML2.0 supported some laws, such as collision detect and three-dimensional sound system. Though these functions are useful so that the operator grasps the virtual environment, these functions need enormous calculation quantity.

Another problem is pointed out about the law in No.1 in the second chapter. **SPACE** disfigured the ratio of the soccer field and the player. What kind of disfigurement is right? If the operator usually loses the soccer ball in **SPACE**, we had better raise the ratio of the soccer ball in **SPACE**.

(b):The description of the object.

What kind of thing is a description which has reality? In the second chapter, I pointed out that **SPACE** does not adopt texture mapping, and adopts coloring of the polygon. Fig. 2 shows the example of this method. Fig. 5 shows that a few polygons are used to build the soccer player. Data size directly influenced the total performance of **SPACE**. While the use of texture mapping or detailed description by many polygons adds reality to the object, these methods make the calculation quantity increase.

(c) The motion of the object.

Virtual Fighter⁹ shows that movements or actions of objects add reality to the objects themselves. What kind of movement gives the object reality? In **SPACE**, the motion of each player was built by hand. This method needs a very long time, and the result of the motion depends on the builder's skill. Compared with this method, the method of motion capture which was used by **Virtual Fighter** makes building of a smooth action or movement possible. However, it is generally difficult to get data for motion capture, because exclusive hardware and high costs are necessary for this method. Furthermore, detailed descriptions of motion make the calculation quantity increase.

Most of these discussion issues such as (b) and (c) seems to not be related to **INS**, but related to the development of hardware and software technology which we already discussed in the first chapter. In this version's **SPACE**, I set several conditions by hand. However, in the RoboCup software league, the various environments for running the **SPACE** are presumed, and it is actually impossible to use hand-tuning in every place.

This discussion applies to (a) as well. The most suitable ratio of the soccer field and the player depends on the audience and the total environment of the screen. Does the operator changes this ratio in every place?

Though **SPACE** does not support these autonomous settings yet, this kind of setting obviously includes **INS**, and this function must be provided by the autonomous agent system.

View:

This issue contains four different factors such as 'viewpoint', 'view position', 'the number of cameras' and 'the switch of the camera'. Each issue is related to the camera, because the scene is only provided by the camera in **SPACE**:

Viewpoint

The viewpoint is the target of the scene held by the camera. In **SPACE**, the viewpoint of each camera is the soccer ball, and each camera follows the soccer ball automatically. In the actual relay of the soccer game, the camera operates by hand, and the viewpoint doesn't always hold the soccer ball. This means that the scene provided by **Space** supplies less information than the scene which is provided by the actual relay of the soccer game. Therefore, the operator(and the audience) of **SPACE** cannot catch the whole scene on the field. However, this viewpoint, which is provided by **SPACE**, gives priority to the operator who gets the detailed information around the ball. This is needed to re-adjust this setting based on the analysis of the relay of soccer game in the real world.

View position

There are five cameras in the circumference of the soccer field. The 'view po-

⁹ **Virtual Fighter** is the entertainment arcade game created by SEGA Enterprizes,Ltd.

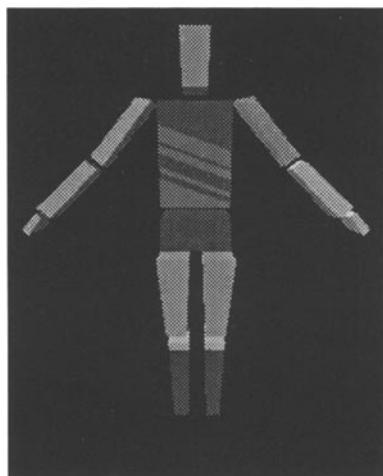


Fig. 5. The soccer player

sition' is the position of each camera. The 'view position' in today's **SPACE** is set based on the analysis of the actual soccer game. Because this 'view position' is fixed, Each camera does not change its 'view position' at all, and the operator also does not do this operation. Because the view position is different in each country, the ideal 'view position' is also different in each country. Thus, we cannot decide the ideal 'view position' based on the relay of the soccer game in the real world.

The number of cameras

Only five cameras exist in **SPACE**. At first, we set seven cameras in **Space**. However, this number required the switching of the camera constantly. As a result, the operator could not catch the position of the ball. Next, we set three cameras in **SPACE**. However, a small number of cameras makes the scene boring. As a result, we set five cameras. However this number depends on other three factors, and cannot be decided independently.

The switch of the camera

In this version's **SPACE**, the operator does not have to switch the camera. The camera which shows the scene on the display is switched automatically, based on the position of the ball. This kind of automatic system is one of the most fundamental systems for **INS**. Thus the important thing about this factor is the value or method for the switching camera. For example, in this version's **SPACE**, switching the camera is based on the position of the soccer ball on the field.

As mentioned above, we set each value by hand. However, each factor is a very complex task, and these factors are influenced by each other. In fact, though this navigation system is a very fundamental one, it is difficult to set each value by

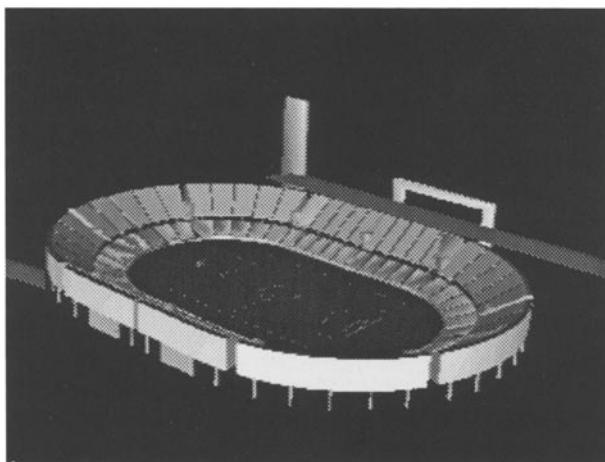


Fig. 6. The soccer field

hand. Furthermore, it needs a total navigation system based on the above four factors to assist human cognition. In this version of **SPACE**, we set this system as follows:

The camera always keep track of the motion of the soccer ball from a fixed position. This means that the ball always moves from the down side to the up side on the screen. The automatic switching function for the camera also supports this system.

Though this is not always a good system for providing the scene, this method was created based on some trial and analysis about the relay of soccer games. Thus, this is one possibility, and the first standard for evaluation. To cope with these complicated problems, we will apply the autonomous agent system. Especially, the standard for evaluation is quite different in each human-being, and the feedback from the audience is an important factor for constructing an ideal **INS**. However, for effective implementaion of this system, more analysis is needed about the actual soccer games which are provided in each country and their computer soccer games.

4 Conclusion

In this paper, we have tried to show the possibilities and the aim of future research of **INS** by discussing the problems of **SPACE**. Our proposed **INS** is basically based on the autonomous agent system, and is related to interactive art and virtual reality. To construct an effective virtual three-dimensional world requires the system's flexibility and the interaction between the system and the audiences or hardware, because effectiveness in this kind of world depends on

the total performance of this system and the total impression of the audiences. To provide an effective scene also requires interaction between the system and audiences, because the standard for evaluation is quite different in each human-being.

Because the standard of evaluation is basically different in each human-being, it is impossible to create a general INS based on a general standard for evaluation. The idealistic INS will only be a personal tool, and a different version of **SPACE** will be used in each RoboCup workshop provided by each national committee.

For developing this research issue, we welcome new research into INS.

5 Future works

This version's **Space** is an early version, and various problems were pointed out. At first, I will find a solution to these problems, and will fix them. After, we will include an autonomous agent system in one of the functions of **SPACE**, and will test this. I will release a new version of **Space** in RoboCup-Paris.

6 Acknowledgments

I would like to express special thanks to Yoko Murakami for her assistance. I also would like to express many thanks to Takemochi Ishii, Michitaka Hirose, Atsuhito Sekiguchi, Masayuki Akamatsu, Hideyuki Morita, Gayle D. Pavola and Hidekazu Hirayu for useful discussions. This research is supported by Softopia Japan.

References

1. 3D Planet.: 3D Assistant. <http://www.3dplanet.com/>
2. Atsushi Shinjoh.: The construction of the interactive space. In *Proceedings of the First International Conference on GeoComputation*, volume-II, pages 739–749, Leeds, UK, September 1996.
3. Gibson, J. J.: *The Perception of the Visual World*. Allen and Unwin:London, 1950.
4. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa.: RoboCup:The Robot World Cup Initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*, Montreal, 1995.
5. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa.: RoboCup:The Robot World Cup Initiative. In *Proceedings of The First International Conference on Autonomous Agent(Agent-97)*, Marina del Ray, The ACM Press, 1997.
6. Itsuki Noda.: Soccer Server Manual. <http://ci.etl.go.jp/~noda/research/kyocho/soccer/manual.newest/main.html>.
7. Maes, P.: Modeling Adaptive Autonomous Agents. *Artificial Life*, **1**(1994) 135–162
8. OpenGL Architecture Review Board.: *OpenGL Programming Guide*. Addison Wesley, 1993.

9. Rodger Lea, Kouichi Matsuda, and Ken Miyashita.: Java for 3D and VRML worlds.
New Riders publishing, 1996.
10. SunMicrosystems,Inc.: Java 3D API Specification. <http://java.sun.com/products/java-media/3D/forDevelopers/3Dguide/j3DTOC.doc.html>, 1997.
11. VRML consortium. VRML97 specification. <http://www.vrml.org/Specifications/VRML97/> .

Generating Multimedia Presentations for RoboCup Soccer Games

Elisabeth André, Gerd Herzog, and Thomas Rist

DFKI GmbH, German Research Center for Artificial Intelligence
D-66123 Saarbrücken, Germany
{andre,herzog,rist}@dfki.de

Abstract. The automated generation of multimedia reports for time-varying scenes on the basis of visual data constitutes a challenging research goal with a high potential for many interesting applications. In this paper, we report on our work towards an automatic commentator system for RoboCup, the Robot World-Cup Soccer. ROCCO (RoboCup-Commentator) is a prototype system that has emerged from our previous work on high-level scene analysis and intelligent multimedia generation. Based on a general conception for multimedia reporting systems, we describe the initial ROCCO version which is intended to generate TV-style live reports for matches of the simulator league.

1 Introduction

Intelligent robots able to form teams and play soccer games are the outspoken vision of the Robot World-Cup Soccer (RoboCup) program which has been initiated by Kitano [12] in order to revitalize and foster research in AI and robotics. From a research point of view the RoboCup program stands for a number of challenges such as agent and multiagent systems, real-time recognition, planning and reasoning, behavior modeling and learning, etc.

The realization of an automated reporting system for soccer games is the long-term vision of a research activity called SOCCER which started in 1986 at University of the Saarland as part of the VITRA (Visual Translator) project. The research challenges addressed in VITRA-SOCCER include the qualitative interpretation of a continuous flow of visual data, and the automated generation of a running report for the scene under consideration. Short sections of video recordings of soccer games have been chosen as a major domain of discourse since they offer interesting possibilities for the automatic interpretation of visual data in a restricted domain. Figure 1 shows a screen hardcopy of the first VITRA-SOCCER prototype which was implemented in 1988 [3]. In the early 90's the focus of interest moved towards the combination of techniques for scene interpretation [3, 7-9] and plan-based multimedia presentation design which has been addressed by DFKI [2, 5]. This was a crucial move in VITRA-SOCCER since it opened the door to an interesting new type of computer-based information system that provides highly flexible access to the visual world [4]. Vice versa, the

broad variety of commonly used presentation forms in sports reporting provides a fruitful inspiration when investigating methods for the automated generation of multimedia reports.

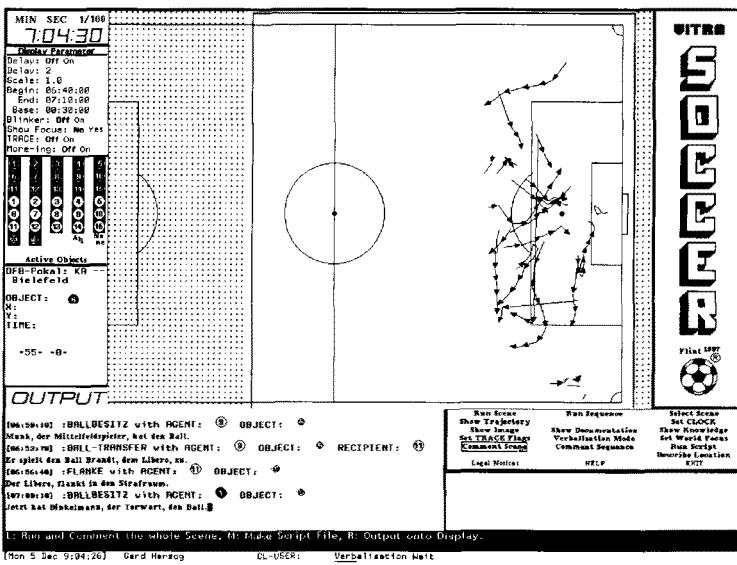


Fig. 1. The basic windows of the first VITRA-SOCCER system

Apparently, the combination of both visions brings into mind the automated reporter that presents and comments soccer games played by robots. But is it worth at all to spend effort on building such a reporting system? From the point of view of VITRA-SOCCER the answer is clearly yes, because RoboCup provides an excellent testbed for both, high-level interpretation of time-varying scenes and the automated generation of multimedia presentations for conveying recognized events.

The first part of this contribution describes the VITRA-SOCCER conception for transforming visual data into multimedia presentations such as TV-style live reports. This conception also forms the basis for the RoboCup commentator system ROCCO which will be presented in more detail in the subsequent sections.

2 From Visual Data to Multimedia Presentations

The automatic description of a time-varying scene by generating elaborated multimedia reports from sensory raw data constitutes a multistage process. In the following, we describe a decomposition of the transformation process into maintainable subtasks. A rough decomposition is depicted in Fig. 2 where the transformation process is subdivided into three distinct levels of processing. The figure

also provides a view on the heterogeneous representation formats as they will be used to bridge between the different steps of the transformation.

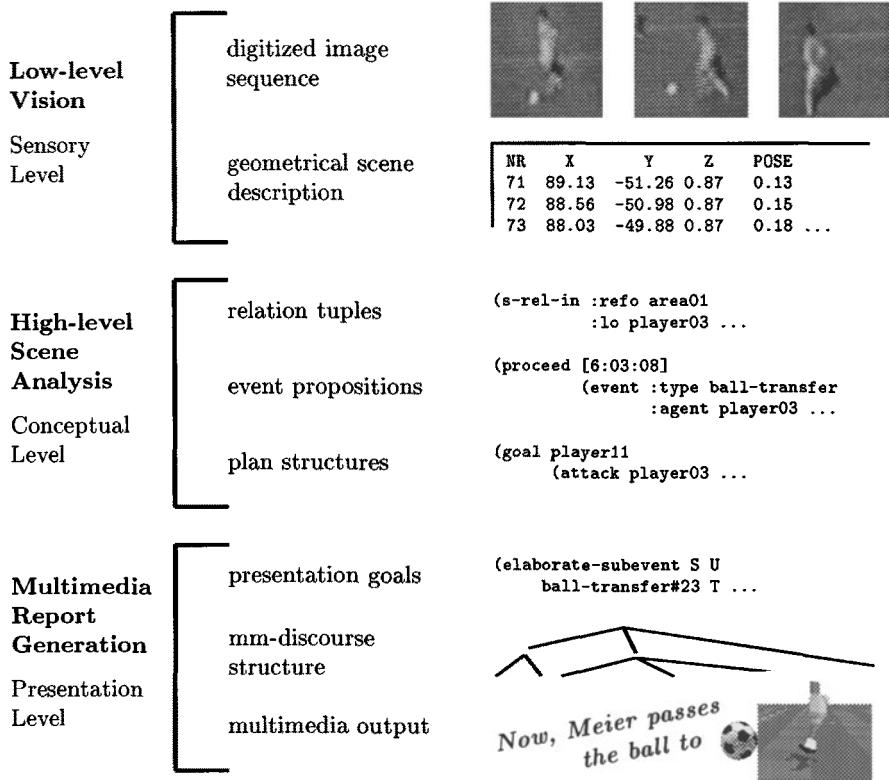


Fig. 2. Levels of representation

2.1 Low-level Vision

The processes on the sensory level start from digitized video frames (cf. Fig. 3) and serve for the automated construction of a symbolic computer-internal representation of the perceived scene.

This processing of image sequences—as it is to be carried out by a vision system—concentrates on the recognition and tracking of visible objects. For VITRA-SOCCER, sequences of up to 1000 images (40 seconds play back time) recorded with a stationary TV-camera during a game in the German professional soccer league, have been evaluated by our project partners at FhG-IITB in Karlsruhe [8]. The employed vision system performed a segmentation and cueing of moving objects by computing and analyzing displacement vector fields. The

calibration of the camera allowed for the transformation of trajectories from the image plane into world coordinates. In the soccer domain, segmentation becomes quite difficult because the moving objects cannot be regarded as rigid and occlusions occur very frequently. The as yet partial trajectories delivered by the vision component (see Fig. 1) required some manual completion in order to ensure a proper assignment of recognized object candidates to previously known players and the ball.



Fig. 3. Image sequence showing an attack in a soccer game

Within the initial version of the system all mobile objects have been represented as centroids since a full 3D-reconstruction of the players from the kind of image sequence shown in Fig. 3 is simply not possible yet (see also [10] for related work using American football scenes). However, the work in [15] describes research on the model-based 3D-reconstruction of the movements of articulated bodies. With this approach, a cylindric representation and a kinematic model of human walking, which is based on medical data, is utilized for the incremental recognition of pedestrians. As can be seen in Fig. 4, the algorithm determines the 3D positions as well as the postures of a single moving person within a real-world image sequence. In addition to treating this example domain in VITRA as well [7], the approach for the geometric modeling of an articulated body has also been adopted in order to represent the players in the soccer domain.



Fig. 4. Automatic recognition of a walking person

As an intermediate representation between low-level image analysis and high-level scene analysis, we rely on the concept of the so-called *geometrical scene description* (GSD) which has been introduced by Neumann [13] as a representation for the intended output of the low-level vision process. The aim of this description format is to represent the original image sequence completely and without loss of information, i.e. the data in the GSD suffice (in principle) to reconstruct the raw images. Basically, the GSD contains information concerning visible objects and their locations over time, together with additional world knowledge about the objects. The GSD constitutes an idealized interface for an intelligent reporting system which is supposed to sit on top of a vision system. In applications, like our VITRA system, the GSD is restricted according to the practical needs. In VITRA, for example, only the trajectories of the moving objects are provided by the vision component. The stationary part of the GSD, an instantiated model of the static background, is fed into the system manually.

2.2 High-Level Scene Analysis

High-level scene analysis provides an interpretation of visual information and aims at recognizing conceptual units at a higher level of abstraction. The information structures resulting from such an analysis encode a deeper understanding of the time-varying scene to be described. They include spatial relations for the explicit characterization of spatial arrangements of objects, representations of recognized object movements, and further higher-level concepts such as representations of behaviour and interaction patterns of the agents observed.

As indicated at the end of Sect. 2.1, the GSD provides the input on which VITRA-SOCCER performs high-level scene analysis. Within the GSD spatial information is encoded only implicitly. In analogy to prepositions, their linguistic counterparts, spatial relations provide a qualitative description of spatial aspects of object configurations [1]. Each spatial relation characterizes a class of object configurations by specifying conditions, such as the relative position of objects or the distance between them.

The characterization and interpretation of object movements in terms of motion events serves for the symbolic abstraction of spatio-temporal aspects of a time-varying scene. In VITRA-SOCCER the recognition of such motion events is based on generic event models, i.e., declarative descriptions of classes of interesting object movements and actions [3]. Besides the question of which events are to be extracted from the GSD, it is decisive how the recognition process is realized. The most distinguishing feature of the VITRA-SOCCER approach is that it relies on an *incremental* strategy for event recognition. Rather than assuming a complete GSD before starting the recognition process (e.g. [13]), VITRA-SOCCER assumes a GSD which will be constructed step by step and processed simultaneously as the scene progresses [8]. An example of an event definition and a sketch of the recognition process is provided in Sect. 4.

For human observers the interpretation of visual information also involves inferring the intentions, i.e. the plans and goals, of the observed agents (e.g., player A does not simply *approach* player B, but he *tackles* him). In the soccer domain

the influence of the agents' *assumed* intentions on the results of the scene analysis is particularly obvious. Given the position of players, their team membership and the distribution of roles in standard situations, stereotypical intentions can be inferred for each situation. Plan-recognition techniques may be employed for the automatic detection of presumed goals and plans of the acting agents [14]. With this approach, each element of the generic plan library contains information about necessary preconditions of the (abstract) action it represents as well as information about its intended effect. A hierarchical organization is achieved through the decomposition and specialization relation. Observable events and spatial relations constitute the leaves of the plan hierarchy.

2.3 Multimedia Report Generation

The presentation component of a reporting system can be regarded as a special instantiation of the class of intelligent multimedia presentation systems as defined in [6].

The interpretation of the perceived time-varying scene together with the original input data provides the required base material for the actual report generation. Depending on the user's information needs the system has to decide which propositions from the conceptual level should be communicated, to organize them and distribute them on several media. To accomplish these subtasks, operator-based approaches have become more and more popular in the User Interfaces community since they facilitate the handling of dependencies between choices (cf. [2, 5]).

The next step is the media-specific presentation of information. In the simplest case, presentation means automatic retrieval of already available output units, e.g. canned text or recorded video clips. More ambitious approaches address the generation from scratch. In VITRA-SOCCER, we were able to base the generation of visual presentations on the camera-recorded visual data and on information obtained from various levels of image interpretation. For example, when generating live reports, original camera data may be directly included in the presentation by forwarding them to a video window. To offer more interesting visualization techniques, we implemented algorithms for data aggregation, display style modifications, and the visualization of inferred information.

The task of natural-language (NL) output generation is usually divided into *text design* and *text realization*. Text design refers to the organization of input elements into clauses. This comprises the determination of the order in which the given input elements can be realized in the text and lexical choice. The results of text design are preverbal messages. These preverbal messages are the input for the *text realization* subtask which comprises grammatical encoding, linearization and inflection. Depending on the selected output mode, formulated sentences are then displayed in a text window or piped to a speech synthesis module. Several approaches to the generation of NL output have been tested within VITRA-SOCCER. Besides templates, we used a text realization component which was based on the formalism of Lexicalized LD/LP Tree Adjoining Grammars (TAG, [11]).

2.4 Task Coordination

An identification of subtasks as described above gives an idea of the processes that a multimedia reporting system has to maintain. The architectural organization of these processes is a crucial issue, especially when striving for a system that supports various presentation styles. For example, the automatic generation of live presentations calls (1) for an incremental strategy for the recognition of object movements and assumed intentions, and (2) for an adequate coordination of recognition and presentation processes. Also, there are various dependencies between choices in the presentation part. To cope with such dependencies, it seems unavoidable to interleave the processes for content determination, media selection and content realization.

3 Rocco: A RoboCup Soccer Commentator System

A practical goal of our current activities is the development of a prototype system called Rocco (RoboCup Commentator) that generates reports for RoboCup soccer games. As a first step, we will concentrate on matches of the simulator league which involves software agents only (as opposed to the different real robot leagues).

Technically, we rely on the RoboCup simulator called SOCCER SERVER [12], which is a network-based graphic simulation environment for multiple autonomous mobile robots in a two-dimensional space. The system provides a virtual soccer field and allows client programs (i.e. software robots) to connect to the server and control a specific player within the simulation (cf. Fig. 5). The SOCCER SERVER environment includes in particular a graphical user interface to monitor object movements and the control messages. In addition, independent components in support of a three-dimensional visualization will be supplied as well (cf. [12]).

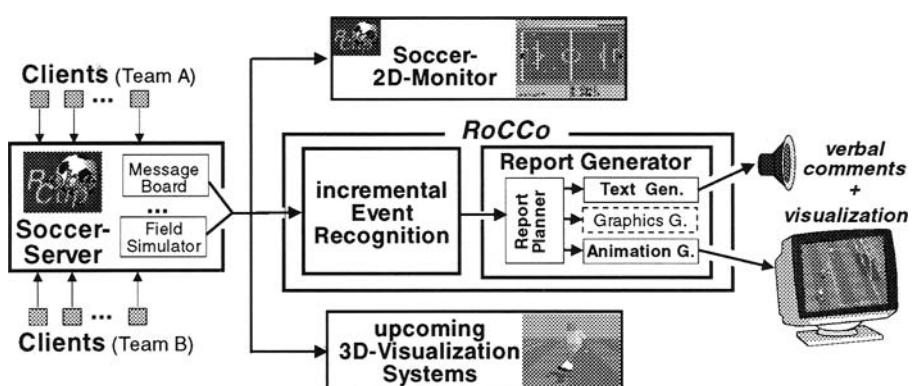


Fig. 5. Connecting SOCCER SERVER and Rocco

The internal communication protocol used between simulator kernel and the monitor component can be exploited to access dynamic information concerning the time-varying scene generated through the simulation. These data serve as the geometrical scene description to be fed into our commentator system.

Depending on basic criteria like the information requirements (complete description vs. summary), the reporting mode (simultaneous vs. retrospective), and the available media for presentation, a broad spectrum of potential presentation styles exists for a system like ROCCO. From the variety of commonly used presentation forms in sports reporting we have selected TV-style live reports as a starting point. To enable this reporting style, ROCCO will comprise modules for the generation of text and 2D-animations (cf. Fig. 5). In its first version, ROCCO will generate a running report of the game and serve as an alternative for the original soccer monitor. For a further version we plan to incorporate a graphics generator in order to support additional reporting styles such as illustrated summaries.

4 Interpretation of RoboCup Soccer Scenes

The requirements for the event recognition component of ROCCO are quite similar to those which have guided the design of VITRA-SOCCER. First of all, ROCCO should be able to produce *live reports*. That is the stream of continuously updated state information about an ongoing soccer game must be processed simultaneously as the game progresses. Moreover, if the presentation is to be focused on what is currently happening, it is very often necessary to describe object motions even while they occur. To enable such descriptions motion events have to be recognized stepwise as they progress and event instances must be made available for further processing from the moment they are noticed first. Therefore, VITRA-SOCCER's approach to the incremental recognition of events seems quite appropriate for the new task, too.

Furthermore, it is desirable to facilitate the definition and modification of event concepts. This is especially important because the set of interesting motion events in the simulation setting are certainly not completely identical to the motion events which we tried to capture from image sequences of real soccer games. Rather than coding event concepts in a pure procedural manner, VITRA-SOCCER allows for declarative descriptions of classes of higher conceptual units capturing the spatio-temporal aspects of object motions. The event concepts are organized into an abstraction hierarchy, grounded on specialization (e.g., *running* is a *moving*) and temporal decomposition. Of course, this kind of modeling event concepts seems also appropriate for ROCCO.

4.1 Definition of Events

Event concepts represent a priori knowledge about typical occurrences in a scene. To define event concepts we use representation constructs as shown in (cf. Fig. 6). The *header* slot indicates the event type to be defined (here *ball-transfer*),

and the number and types of objects which are involved in the occurrence of such an event. The slot *conditions* may be used to provide further constraints on variable instantiations. For example, according to the definition, a *ball-transfer* event involves the ball and two players which must belong to the same team. The slot *subconcepts* lists all involved sub-events. Roughly speaking one can say that the occurrence of the event implies occurrences of all its subconcepts. However, the reverse syllogism does not necessarily hold since the temporal relationships between the occurrences must be considered. For this purpose the event definition scheme comprises the slot *temporal-relations* which allows for the specification of timing information in an interval-based logics. For example, a *ball-transfer* event can only be recognized, if the three specified subevents occur sequentially without interruption.

```

Header: (ball-transfer ?p1*player ?b*ball ?p2*player)
Conditions: (eql (team ?p1) (team ?p2))
Subconcepts: (has-ball ?p1 ?b) [I1]
                  (move-free ?b) [I2]
                  (has-ball ?p2 ?b) [I3]
Temporal-Relations: [I1] :meets [ball-transfer]
                         [I1] :meets [I2]
                         [I2] :equal [ball-transfer]
                         [I2] :meets [I3]
```

Fig. 6. Event model for *ball-transfer*

The decompositional approach to event modeling results in a hierarchical organisation of event definitions. The lowest level of this hierarchy is formed by elementary events, such as *move*, that can be calculated directly from object positions which are recorded within the GSD.

4.2 Event Recognition

The recognition of an event occurring in a particular scene corresponds to an instantiation of the respective generic event concept. In order to enable an incremental recognition of occurrences the declarative definitions of event concepts are compiled into so-called *course diagrams*. Course diagrams are labeled directed graphs which are used to internally represent the prototypical progression of an event. The recognition of an occurrence can be thought of as traversing the course diagram, where the edge types are used for the definition of the basic event predicates.

Since the distinction between events that have and those that have not occurred is insufficient, we have introduced the additional predicates *start*, *proceed* and *stop* which are used for the formulation of traversing conditions in the course diagrams. In contrast to the event specifications described above, course

diagrams rely on a discrete model of time, which is induced by the underlying sequence of digitized video frames. Course diagrams allow incremental event recognition, since exactly one edge per unit of time is traversed. However, by means of constraint-based temporal reasoning, course diagrams are constructed automatically from the interval-based concept definitions [9].

The derived course diagram for the concept *ball-transfer* is shown in Fig. 7. As can be seen from the labels of the edges and the traversing conditions the event starts if a *has-ball* event stops and the ball is free. The event proceeds as long as the ball is moving free and stops when the recipient (who must be a teammate) has gained possession of the ball.

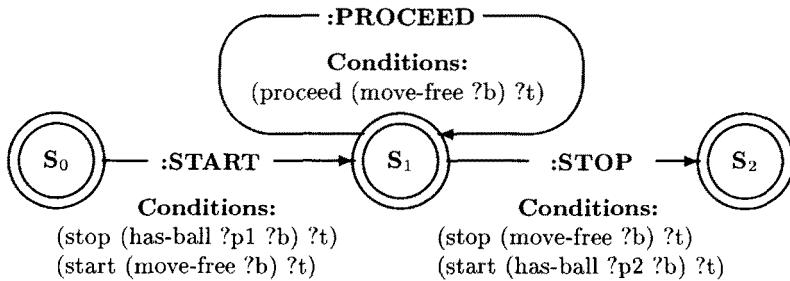


Fig. 7. Course diagram for *ball-transfer*

Course diagrams serve as recognition automata. As soon as new sensory data are provided, the recognition component continues with the traversal of already activated course diagrams and tries to trigger new ones. An activated course diagram corresponds to exactly one potential instantiation of a generic event. Consequently, several recognition automata for the same event type may run concurrently to capture simultaneous occurrences of a certain event type.

5 Report Planning

To plan the contents and the structure of a soccer report, we rely on the plan-based approach which has been originally developed for the WIP presentation system [2, 5]. The main idea behind this approach is to formalize action sequences for composing multimedia material as operators of a planning system. The effect of a planning operator refers to a complex communicative goal (e.g., describe the scene) while the expressions in the body specify which communicative acts have to be executed in order to achieve this goal.

Starting from a presentation goal, the planner looks for operators whose effect subsumes the goal. If such an operator is found, all expressions in the body of the operator will be set up as new subgoals. The planning process terminates if all subgoals have been expanded to elementary generation tasks which are

forwarded to the media-specific generators. For the first prototype of ROCCO the main emphasis will be on the planning of verbal comments which will be realized by a text generator.

To utilize the plan-based approach for RoboCup reports, we had to define additional strategies for scene description. For example, the strategy shown in Fig. 8 may be used to verbally describe a sequence of events by informing the user about the main events (e.g., *team-attack*) and to provide more details about the subevents (e.g., *kick*). While *s-inform* is an elementary act that is directly forwarded to the text generator, *elaborate-subevents* has to be further refined, e.g., by applying the strategy shown in Fig. 9. It informs the user about all salient subevents and provides more details about the agents involved. To determine the salience of an event, factors such as its frequency of occurrence, the complexity of its generic event model, the salience of involved objects and the area in which it takes place are taken into account (see also [3]). All events are described in their temporal order.

```

Header: (Describe-Scene S U ?events)
Effect: (FOREACH ?one-ev
            WITH (AND (BEL S (Main-Ev ?one-ev))
                      (BEL S (In ?one-ev ?events)))
                  (BMB S U (In ?one-ev ?events)))
Applicability-Conditions:
            (BEL S (Temporally-Ordered-Sequence ?events))
Inferior Acts:
            ((FOREACH ?one-ev
              WITH (AND (BEL S (Main-Ev ?one-ev))
                        (BEL S (In ?one-ev ?events)))
                  (S-Inform S U ?one-ev)
                  (Elaborate-Subevents S U ?one-ev)))

```

Fig. 8. Plan operator for describing a scene

The strategies defined in Fig. 8 and Fig. 9 can be used to generate a posteriori scene descriptions. They presuppose that the input data from which relevant information has to be selected are given a priori. Since both strategies iterate over *complete* lists of temporally ordered events, the presentation process cannot start before the interpretation of the whole scene is completed.

However, ROCCO should also be able to generate live reports. Here, input data are continuously delivered by a scene interpretation system and the presentation planner has to react immediately to incoming data. In such a situation, no global organization of the presentation is possible. Instead of collecting scene data and organizing them (e.g., according to their temporal order as in the first two strategies), the system has to locally decide which event should be reported next considering the current situation. Such behavior is reflected by the strategy shown in Fig. 10. In contrast to the strategy shown in Fig. 9, events are selected

Header: (Elaborate-Subevent S U ?ev)

Effect: ((FOREACH ?sub-ev
 WITH (AND (BEL S (Salient ?sub-ev))
 (BEL S (Sub-Ev ?sub-ev ?ev))))
 (BMB S U (Sub-Ev ?sub-ev ?ev))))

Applicability-Conditions:
 (AND (BEL S (Sub-Events ?ev ?sub-events))
 (BEL S (Temporally-Ordered-Sequence ?sub-events)))

Inferior Acts:
 ((FOREACH ?sub-ev
 WITH (AND (BEL S (In ?sub-ev ?sub-events))
 (BEL S (Salient ?sub-ev)))
 (S-Inform S U ?sub-ev)
 (Elaborate-Agents S U ?sub-ev)))

Fig. 9. Plan operator for describing subevents

for their topicality. Topicality is determined by the salience of an event and the time that has passed since its occurrence. Consequently, the topicality of events decreases as the scene progresses. If an outstanding event (e.g., a goal kick) occurs which has to be verbalized as soon as possible, the presentation planner may even give up partially planned presentation parts to communicate the new event as soon as possible.

Header: (Describe-Next S U ?ev)

Effect: ((AND (BMB S U (Next ?preceding-ev ?ev))
 (BMB S U (Last-Reported ?ev))))

Applicability-Conditions:
 (AND (BEL S (Last-Reported ?preceding-ev))
 (BEL S (Topical ?ev *Time-Available*))
 (BEL S (Next ?preceding-ev ?ev)))

Inferior Acts:
 ((S-Inform S U ?ev)
 (Describe-Next S U ?next-ev)))

Fig. 10. Plan operator for simultaneous description

6 Generation of Verbal Comments

VITRA-SOCCER was only able to generate complete, grammatically well-formed sentences. As a consequence, the output was not tuned to live reports that often contain short, sometimes syntactically incorrect phrases. Unlike VITRA-SOCCER, ROCCO aims at the generation of a large variety of natural-language

expressions that are typical of the soccer domain. Since it is rather tedious to specify soccer slang expressions in formalisms like TAG, we decided to use a template-based generator instead of fully-fledged natural-language design and realization components. That is language is generated by selecting templates consisting of strings and variables that will be instantiated with natural-language references to objects delivered by a nominal-phrase generator. To obtain a rich repertoire of templates, 13.5 hours of soccer TV reports in English have been transcribed and annotated. Each event proposition is assigned a list of templates that may be used to verbalize them. For instance, Table 1 lists some templates that may be used to verbalize an instance of a finished *ball-transfer* event in a live report.

Template	Constraints	Vb	Floridity	Sp	Formality	Bias
(?x passes the ball to ?y)	None	8	dry	4	formal	neutral
(?x playes the ball towards ?y)	None	8	dry	4	formal	neutral
(?x towards ?y)	(NotTop ?x)	5	dry	3	slang	neutral
(?x combines with ?y)	None	6	normal	3	slang	neutral
(?x now ?y)	(NotTop ?x)	5	dry	2	slang	neutral
(ball played towards ?y)	None	5	dry	3	colloquial	neutral
(the ball came from ?x)	(NotTop ?x)	6	dry	3	colloquial	neutral
(a touch by ?x)	None	5	normal	2	slang	neutral
(turn from ?x)	None	4	dry	2	slang	neutral
(shot)	None	1	dry	1	colloquial	neutral
(?x)	(NotTop ?x)	2	dry	1	slang	neutral
(now ?y)	(NotTop ?y)	3	dry	1	slang	neutral
(?y was there)	None	4	dry	1	slang	neutral
(?y)	(NotTop ?y)	2	dry	1	slang	neutral
(well done)	None	2	dry	0	colloquial	positive
(good work from ?x)	None	5	normal	1	colloquial	positive
(a lovely ball)	None	3	flowery	1	colloquial	positive

Table 1. Some templates for the event *ball-transfer*

Constraints specify whether a certain template can be employed in the current context. For instance, “*Now Miller*” should not be uttered if Miller is already topicalized. Currently, we assume that the whole field is visible. Thus we do not get any constraints referring to the visual focus. To select among several applicable templates, the following features are considered:

Verbosity: The verbosity of a template depends on the number of words it contains. While instantiated slots correspond to exactly one word, non-instantiated slots have to be forwarded to the nominal phrase generator which decides what form of nominal phrase is most appropriate. Since the length is not yet known at that point in time, ROCCO assumes a default word number of two for non-instantiated slots.

Floridity: We distinguish between dry, normal and flowery language. Flowery language is composed of unusual ad hoc coinages, such as “*a lovely ball*”. Templates marked as normal may contain metaphors, such as (*finds the gap*), while templates marked as dry, such as (*playes the ball towards ?y*) just convey the plain facts.

Specificity: The specificity of a template depends on the number of verbalized deep cases and the specificity of the natural-language expression chosen for the action type. For example, the specificity of (*?x looses the ball to ?y*) is 4 since 3 deep cases are verbalized and the specificity of the natural-language expression referring to the action type is 1. The specificity of (*misdone*) is 0 since none of the deep cases occurs in the template and the action type is not further specified.

Formality: This attribute can take on the values: slang, colloquial and normal. Templates marked as formal are grammatically correct sentences which are more common in newspaper reports. Colloquial templates, such as “*ball played towards Meier*”, are simple phrases characteristic of informal conversation. Slang templates are colloquial templates peculiar to the soccer domain, such as “*Miller squeezes it through*”.

Bias: Biased templates, such as (*well done*) contain an evaluation of an action or event. Bias may be positive, negative or neutral.

While the features listed above don't change within a game and can be computed before generating a report, *dynamic features*, such as *How often mentioned?* and *Last mentioned?*, have to be continuously updated during the game.

To select a template, Rocco first determines all templates associated with an event concept, checks for which of them the constraints are satisfied and subsequently performs a four-phase filtering process. Only the best templates of each filtering phase will be considered for the next evaluation step.

1. *Compute the effectivity rate of all applicable templates:*

Set the effectivity rate of all templates to 0 and punish or reward them by increasing or decreasing their effectivity rate considering their length and specificity. If the system is under time pressure, a template will be punished for its length, but rewarded for its specificity. In phases with little activity, all templates will get a reward both for their length and specificity. In all other situations, only the specificity of a template will be rewarded.

2. *Compute the variability rate of the most effective templates:*

Punish templates that have been recently or frequently been used. Unusual templates, i.e., templates that seldom occur in a soccer report, get an additional punishment.

3. *Convey the speaker's partiality:*

Select templates which convey the speakers partiality best. That is if the system is in favor of team X, prefer positive templates for describing activities of X to neutral templates, and neutral templates to negative templates.

4. *Select templates that convey the style of presentation best:*

For example, prefer slang templates to colloquial templates and colloquial templates to normal templates in case of a live report.

For illustration, let's suppose that ROCCO is not in favor of a particular team, under time pressure and has to verbalize the following event proposition:

*(ID123 :Type ball-transfer :Agent sp1 :object Ball :Recipient sp2
:State Finished :Start 5 :Finish 10)*

Let's further assume that *sp1* is topicalized. ROCCO first selects all applicable templates from the table shown as Table 1. (*?x towards ?y*), (*the ball came from ?x*), (*?x now ?y*), and (*?x*) are not applicable because *sp1* is topicalized. The remaining templates are checked for their effectiveness. Only four templates pass this test: (*ball played towards ?y*), (*shot*), (*now ?y*) and (*?y*). For the purpose of this example, we assume that the system was under permanent time pressure and that (*shot*), (*now ?y*) and (*?y*) have already been used very often. Therefore, after the variability test only the candidate (*ball played towards ?y*) is left, and there is no need to apply further filters. The remaining template is taken for verbalization and the value for the non-instantiated slot *?y* is forwarded to the nominal phrase generator that takes into account the discourse context to decide whether to generate a pronoun or a noun phrase (with modifiers).

7 Conclusion

In this contribution, we have reported on our efforts towards the development of ROCCO, a system that observes RoboCup soccer simulations in order to provide informative multimedia reports on recognized occurrences. The conception of ROCCO builds on the experience gained from our long-term research in the context of VITRA-SOCCER. The new system ROCCO follows an incremental strategy for the recognition of higher-level concepts, such as spatial relations and motion events, and relies on a plan-based approach to communicate recognized occurrences with multiple presentation media (currently written text, spoken language, and 2D-visualizations).

Given our specific interest in high-level scene analysis and automated multimedia report generation, we will benefit in particular from the RoboCup initiative. The intrinsic difficulty of low-level image analysis leads to limited availability of suitable data material which has always been hindering our experimental investigations in the context of VITRA-SOCCER. RoboCup promises to ameliorate this situation. The broad use of a common simulation environment will certainly contribute to a much better research infrastructure which we can use as a flexible testbed for our current and future work on the automated generation of multimedia reports for time-varying scenes.

Acknowledgements

We would like to thank Hiroaki Kitano for encouraging us to start the Rocco project. Furthermore, we are grateful to Dirk Voelz for transcribing and annotating the soccer TV live reports and his work on the implementation of the Rocco system.

References

1. E. André, G. Bosch, G. Herzog, and T. Rist. Coping with the Intrinsic and the Deictic Uses of Spatial Prepositions. In K. Jorrard and L. Sigurev, editors, *Artificial Intelligence II: Methodology, Systems, Applications*, pages 375–382. North-Holland, Amsterdam, 1987.
2. E. André, W. Finkler, W. Graf, T. Rist, A. Schauder, and W. Wahlster. WIP: The Automatic Synthesis of Multimodal Presentations. In M. T. Maybury, editor, *Intelligent Multimedia Interfaces*, pages 75–93. AAAI Press, Menlo Park, CA, 1993.
3. E. André, G. Herzog, and T. Rist. On the Simultaneous Interpretation of Real World Image Sequences and their Natural Language Description: The System SOC-CER. In *Proc. of the 8th ECAI*, pages 449–454, Munich, Germany, 1988.
4. E. André, G. Herzog, and T. Rist. Multimedia Presentation of Interpreted Visual Data. In P. Mc Kevitt, editor, *Proc. of AAAI-94 Workshop on "Integration of Natural Language and Vision Processing"*, pages 74–82, Seattle, WA, 1994. Also available as Report no. 103, SFB 314 – Project VITRA, Universität des Saarlandes, Saarbrücken, Germany.
5. E. André and T. Rist. Towards a Plan-Based Synthesis of Illustrated Documents. In *Proc. of the 9th ECAI*, pages 25–30, Stockholm, Sweden, 1990.
6. M. Bordegoni, G. Faconti, S. Feiner, M. T. Maybury, T. Rist, S. Ruggieri, P. Trahanias, and M. Wilson. A Standard Reference Model for Intelligent Multimedia Presentation Systems. *Computer Standards & Interfaces*, 1998. To appear in the Special Issue on Intelligent Multimedia Presentation Systems.
7. G. Herzog and K. Rohr. Integrating Vision and Language: Towards Automatic Description of Human Movements. In I. Wachsmuth, C.-R. Rollinger, and W. Brauer, editors, *KI-95: Advances in Artificial Intelligence. 19th Annual German Conference on Artificial Intelligence*, pages 257–268. Springer, Berlin, Heidelberg, 1995.
8. G. Herzog, C.-K. Sung, E. André, W. Enkelmann, H.-H. Nagel, T. Rist, W. Wahlster, and G. Zimmermann. Incremental Natural Language Description of Dynamic Imagery. In C. Freksa and W. Brauer, editors, *Wissensbasierte Systeme. 3. Int. GI-Kongreß*, pages 153–162. Springer, Berlin, Heidelberg, 1989.
9. G. Herzog and P. Wazinski. VIIsual TRAnslator: Linking Perceptions and Natural Language Descriptions. *AI Review*, 8(2/3):175–187, 1994.
10. S. Intille and A. Bobick. Visual Tracking Using Closed-Worlds. In *Proc. of the 5th Int. Conf. on Computer Vision*, pages 672–678, Cambridge, MA, 1995.
11. A. Kilger. Using UTAGs for Incremental and Parallel Generation. *Computational Intelligence*, 10(4):591–603, 1994.
12. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. RoboCup: A Challenge Problem for AI. *AI Magazine*, 18(1):73–85, 1997.
13. B. Neumann. Natural Language Description of Time-Varying Scenes. In D. L. Waltz, editor, *Semantic Structures: Advances in Natural Language Processing*, pages 167–207. Lawrence Erlbaum, Hillsdale, NJ, 1989.
14. G. Retz-Schmidt. Recognizing Intentions, Interactions, and Causes of Plan Failures. *User Modeling and User-Adapted Interaction*, 1:173–202, 1991.
15. K. Rohr. Towards Model-based Recognition of Human Movements in Image Sequences. *Computer Vision, Graphics, and Image Processing (CVGIP): Image Understanding*, 59(1):94–115, 1994.

Football in Recent Times: What We Can Learn From the Newspapers

Ian Frank

Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, Japan 305
`ianf@etl.go.jp`

Abstract. This paper uses a basic statistical analysis of newspaper articles to gain an insight into the game of football. Basic features of the game are established and examined in a way that should give insights to the designers of RoboCup teams and also suggest future developments for the regulations that determine the RoboCup environment. As concrete examples of possible Soccer Server modifications, we suggest touchline coaching, a revised model of stamina, and the inclusion of substitutions.

1 Introduction

Formalising human expertise is a useful technique in many areas of AI. In expert systems, for example, it takes the form of *knowledge elicitation* (e.g., see [1]). In automated reasoning, designers of theorem-provers attempt to capture techniques used by human mathematicians (e.g., [2]). In game-playing, designers commonly incorporate features such as opening books and pattern-recognition elements into their programs, based on moves that experts would make in given positions (e.g., see [3]).

For the designers of teams in RoboCup, formalisations of expert knowledge should also be a useful tool when designing teams of players. However sources of such information are not immediately obvious: access to football professionals is limited, and books by experts rarely concentrate on the high-level, declarative descriptions of the game that would be most useful to RoboCup designers. There are many actual football games shown on TV, but although some national teams are rumoured to record games using pitch diagrams, the lack of a generally accepted transcription language means that most records remain largely visual.

In this paper, we generate our own declarative information on football by drawing on a novel source of information: newspaper match reports and features. Such articles have only become accessible to the public in a convenient form recently, with the growth of on-line newspapers. As our particular source, we chose the Internet pages of the Times newspaper, from which we downloaded and analysed the entire collection of football articles.

Our basic supposition is that any factor that is important for the success of a footballing team should be mentioned at some time by football reporters. We carry out a basic statistical analysis of the Times database in a way that we hope will:

1. Identify the important features of the game,
2. Help give more detailed objectives than ‘scoring goals’ to the designers of teams in RoboCup.
3. Maybe suggest directions in which the RoboCup regulations could develop. (For example, the feature of rear motion detectors was incorporated into the Soccer Server as a direct result of one of the quotes uncovered from the Times archive.)

In §2 we give an overview of the database itself, and use this to motivate the selection of a limited number of primary features of the game of football. §3 then discusses each of these features in turn, illustrating the discussion with example passages taken from the database of articles. Finally, §4 discusses the concept of *beauty* within the game of football, before we conclude in §5.

2 Database Overview

The total database contains 2402 football articles downloaded from the WWW site of the Times Newspapers Ltd, at <http://www.the-times.co.uk/>. These articles cover the 1996-1997 football season from before its start (7 August 1996) until the date that this paper was first drafted (25 Feb 1997). This represents 174 days of newspaper coverage, since the Times itself does not have a Sunday edition. In their raw format, the files containing the articles occupy 21.6 Mbytes of disk space, but after deleting the HTML headers and footers this reduces to 9.8 Mbytes.

Table 1 shows the most commonly occurring words in the database, after the removal of words with no obvious relation to football (*e.g.*, purely grammatical words), and also proper names such as those of players or clubs. We have arbitrarily chosen 400 occurrences as the cut-off point for inclusion in this table. It is a sign of the times that *million* (828) occurs so often in the database, as well as variations such as *millions* (31), *millionaire* (15), and even *multimillionaire* (13).

Some of the words in this table appear simply because they are the names of things commonly associated with the sport (*e.g.*, *league* (2454), *game* (2298), *cup* (2150), *club* (1979), *football* (1965), *division* (1880), *etc.*). Other words, however, can more easily be seen to correspond to a particular feature of the game itself. For example, the words *manager* (2435), *players* (1980), *team* (1772) and *coach* (445) all suggest the notion of ‘teamwork’. Similarly, *free* (765) (from ‘free kick’) and *penalty* (606) suggest the importance of ‘set pieces’ in football.

In total we identified eight basic features corresponding to the words in Table 1. These features appear next to the relevant words in the table and, in order of first appearance, are: teamwork, basic skills, playing conditions, objectives, attack, set pieces, midfield, and defence. The remainder of this paper will be devoted to examining each of these features in more detail.

Word	Feature
<i>said</i> (3743)	Teamwork
<i>league</i> (2454)	
<i>manager</i> (2435)	Teamwork
<i>game</i> (2298)	
<i>cup</i> (2150)	
<i>players</i> (1980)	Teamwork
<i>club</i> (1979)	
<i>football</i> (1965)	
<i>division</i> (1880)	
<i>ball</i> (1829)	Basic Skills
<i>goal</i> (1798)	
<i>season</i> (1784)	
<i>team</i> (1772)	Teamwork
<i>time</i> (1671)	
<i>back</i> (1609)	
<i>minutes</i> (1446)	
<i>match</i> (1414)	
<i>off</i> (1302)	
<i>play</i> (1213)	
<i>side</i> (1112)	
<i>away</i> (1110)	Playing Conditions
<i>player</i> (1109)	Teamwork
<i>win</i> (1099)	Objectives
<i>referee</i> (952)	
<i>goals</i> (939)	
<i>million</i> (828)	
<i>shot</i> (826)	Attack
<i>games</i> (814)	
<i>man</i> (767)	Basic Skills
<i>free</i> (765)	Set Pieces
<i>minute</i> (736)	
<i>played</i> (731)	
<i>midfield</i> (700)	Midfield
<i>scored</i> (697)	Attack
<i>forward</i> (686)	Attack
<i>long</i> (682)	Basic Skills
<i>playing</i> (676)	
<i>won</i> (622)	
<i>think</i> (618)	
<i>penalty</i> (606)	Set Pieces
<i>goalkeeper</i> (597)	Defence
<i>defence</i> (591)	Defence
<i>people</i> (586)	
<i>supporters</i> (583)	Playing Conditions
<i>matches</i> (570)	
<i>lost</i> (519)	
<i>chance</i> (470)	Basic Skills
<i>run</i> (470)	Basic Skills
<i>move</i> (455)	Basic Skills
<i>coach</i> (445)	Teamwork
<i>teams</i> (440)	Teamwork
<i>defeat</i> (437)	
<i>cross</i> (436)	Attack
<i>say</i> (421)	Teamwork
<i>post</i> (419)	
<i>injury</i> (416)	
<i>points</i> (412)	Objectives
<i>squad</i> (408)	Teamwork
<i>later</i> (408)	
<i>thought</i> (405)	
<i>early</i> (401)	Basic Skills
<i>draw</i> (400)	Objectives

Table 1. Most commonly occurring words in Times football articles

3 Game Features

Each of the following subsections examines one of the features of football identified in the previous section (with some of the features being broken down into further sub-categories). More examples of word frequencies from the database of news articles are given, and illustrative quotes are also selected and discussed. Possible lessons for the Soccer Server are emphasised by highlighting game features that cannot be modelled (or can only be modelled with difficulty) within the framework of the current implementation.

3.1 Teamwork

Database examples: *manager* (2435), *players* (1980), *team* (1772), *coach* (445), *squad* (408), *captain* (281).

The overall notion of teamwork is well described by the following quotes:

'People work in teams because together they have the potential to create something they cannot create alone. By maximizing the quality of the relationship between team members, teams maximise their performance.' The words of John Syer, a psychologist who works in sport and business, in his latest book, *How Team-work Works*. [4]

...you can never expect that any single player will go out and win the game by himself. Each individual needs collective support. [5]

In this paper 'maximising the quality of the relationship between the team members' will be interpreted as optimising the way that a team, as a whole, deals with each of the game features discussed in the following subsections. Note, though, that whilst the maxim that 'no player functions alone' generally holds true, it does just occasionally get disproven:

In the 87th minute of the match between AC Milan and Verona, George Oppong Weah transcended any solo goal in the imagination. He defended a Verona corner, he ran 85 metres, he outpaced, out-thought, outclassed seven gentlemen of Verona, and within 14 seconds, 30 strides and 14 touches of the ball, he scored. [6]

A goal such as this would be highly unlikely under the current implementation of the Soccer Server, since the model of stamina makes it difficult for one player to outrun other players for an extended period. We will return to this issue in the section on 'Basic Skills'.

Communication. A natural sub-topic of 'teamwork' is 'communication'. Already, a number of designers of teams participating in the Pre-RoboCup simulator league have reported using the say and hear protocols of the Soccer Server to increase each player's information about the state of the game. For example, a player in a good position to receive a pass can broadcast this information to a team-mate that has the ball. Also, players that are not directly involved in the play can keep team-mates up to date with information such as the positions of opponents who may be approaching them from behind.

In addition to spreading information about the environment, communication also offers an avenue for feedback and learning during the course of a game. For example, when a player chooses one action from between two or more alternatives, it may actually be one of his team-mates that is in the best position to judge whether the outcome has actually improved the position of their team. Also, decision-making can be improved during a game by using communication to express *displeasure* at the actions of team-mates:

I tell you, Paddington Bear could learn something from the Ranieri hard stare. A couple of times, when a colleague failed yet again to understand his elevated conception of football, he dealt a look that almost sliced his head off. [7]

It should further be noted that managers as well have a role in communicating feedback and advice to players:

[United's manager] came down to the bench and, after much encroachment to the touchline, United's tactics evolved so that *they* put the emphasis on attack.... [8]

Francis appeared to have hot cinders in his training shoes at this point, hopping around excitedly a metre or so from the dugout as he waved players forward. [9]

Currently, the rules of the simulator league allow programmers to modify their software at half-time. This encourages the design of software that utilises concepts high-level enough to be tinkered with during such a small time-span. However, the above quotes show that real football managers can also have an effect on their team's performance *during* the play itself. This suggests an increased role for the current 'coaching mode' of Soccer Server.

Specifically, teams could be allowed to use the 'coach' to monitor play in real time and shout advice to players. Since the coach has a global view, it should be easier to determine something about the play from this perspective, compared to the restricted vision of the individual players. Of course, to prevent the coach actually directing the players during play itself, coaching advice should only be allowed during breaks (*e.g.*, between the scoring of a goal and the next kickoff or at throw-ins).

Increasing the role of the coaching mode would facilitate research on one of the more difficult (and interesting) areas of RoboCup: enabling a team of clients to monitor — and meaningfully react to — the strategy and tactics of another team. Although a number of researchers are already using machine learning techniques to develop client programs strong enough to play in RoboCup, there seem to be very few (if any) teams that include a component that makes a serious attempt to learn *during* a match. It could be that tackling the 'easier' problem of using the coach to analyse the play will encourage people down the harder road of getting the actual players to do the same task for themselves.

Note that one further part of the manager's role that does not feature in the current implementation of the Soccer Server is squad selection and substitution. In real life, this plays an important part in determining the outcome of a game (*e.g.*, *substitute* (195), *substitutes* (59), *substituted* (26), *substitution* (11)). We will return to this issue when we discuss the issue of stamina in the following subsection.

3.2 Basic Skills

Database examples: *kick* (384), *work* (350), *turned* (330), *pass* (307), *position* (300), *training* (296). *control* (192) *touch* (174)

Touch. Ball control and touch is a recurring theme when talking about the skills of players:

You could see the way Cantona touches the ball what a great player he could be [10]

Despite being talked about frequently, however, ‘touch’ is not an easy concept to define. Perhaps in the context of RoboCup it relates to the fine control of the ball necessary to carry out more complicated actions such as passing (or intercepting, dribbling, turning, shooting *etc.*).

Passing. As for passing itself, it is often not just important to get the ball to another team-mate, but to do so in a way that actually advances the position of the team.

His first touch is sure, his passing elegant, and he has the courage to eschew the obvious for the enterprising. [11]

Position. Actually getting players into a good position to receive passes (and to intercept the opponents’ passes) therefore becomes a key to successful interplay:

There is no footballing equivalent of ‘street-smart’, but there should be. Wright is supremely ‘pitch-smart’, or ‘ball-smart’. [4]

A lack of ‘running off the ball’ was one of the most striking features of many of the teams participating in the Pre-RoboCup tournament. This area is particularly ripe for progress since it also offers the benefits of increasing the freedom of the players who actually do have the ball:

Roger Hunt was darting about in a series of decoy runs, ‘diving in different directions and their defenders were being pulled and all the time I was allowed to go further and further. I thought, if they let me go another half a dozen yards I’m going to have a dip.’ [12]

Improving the use of space and the passing of RoboCup teams should see the demise of the ‘rugby effect’: the result of a strategy that sees teams constantly send four or five players after the ball, resembling a pack of rugby forwards. A good passing side should easily be able to outmanoeuvre teams that concentrate their players in this way.

Possession. The combination of touch, passing and positioning allows a team to control possession:

One-touch or two-touch, short-range or long-range, they appear to know instinctively where their best-placed team-mate is positioned. Possession is treasured, not to be discarded lightly. [13]

Although not mentioned particularly frequently in the database (*possession* (95)), possession is one of the key aims during many stages of football play:

'If you give the ball away at this level, you cut your throat.' (Alex Ferguson, quoted in [14])

Turns. Once in possession, one of the ways (other than passing) of keeping possession when challenged by an opponent is to get past the opponent:

....[Okocha] teased with his silky control, he drew Gary Neville close to him. Tighter, tighter, the African seemed to invite the England right back; and then, with remarkable balance, Okocha turned full circle, swept the ball with a delicate little chip out of the reach of the now bemused Neville, and skipped past. [15]

Within the RoboCup environment, deciding when to pass and when to try to turn an opponent may be a critical element of a player's abilities: a team that never takes on the opponents may never make any real forward progress, but a team that tries to run at the opponents too often may give away too much possession.

Work, Training & Stamina. The following quote describes a player called Lucas Neill:

Speed, stamina and skill are his assets, he says, and one can believe him; though he is hard on himself when he says that his touch needs to improve. [16]

We have already discussed touch, but not speed, stamina or skill. Of these three, skill is the easiest to deal with, as it has been covered to a certain extent by our discussion of ball control and touch, and is in any case a general concept applying to many aspects of the game (*skill* (89), *skills* (87), *skilful* (18), *skilled* (8), *skilfully* (4)). Skill is also used to describe types of ball control that are not reproducible within the Soccer Server environment:

...he made a football, of all the unwieldy objects, behave as a stump-shattering late in-ducker. He seemed to possess an almost painterly talent, his brush the swerve-inducing, ecstatically-sponsored Predator boots.[17]

Speed and stamina, however, both feature as parameters within the Soccer Server. They are also an integral part of the modern game, and they form the backbone of the style of play that has become traditional in England:

'Look at the World Cup final video of 1970 between Brazil and Italy,' Olsen said, 'and you see it was almost a walking game [even allowing for the heat and altitude]. The game has advanced so much in speed and fitness.' [18]

Ask any foreign player to draw an identikit picture of the typical English footballer and the image would be readily identifiable. Pearce is easy to caricature and easy to despise, too, the ultimate product of a system that reveres physique over technique, stamina over skill. [19]

In the current Soccer Server, each player has a maximum stamina (the default is 200 units) which decreases by the dash power whenever a player runs (the dash power may be up to 100). On each simulation cycle, stamina is recovered by a preset amount (the default is 20 units). This clearly models *short-term* exhaustion very effectively, but we suggest here that it may also be beneficial to incorporate a slightly more sophisticated *long-term* model of stamina into the environment.

The modelling of long-term stamina within the Soccer Server would greatly broaden the scope of the tactical and resource management issues in the simulation. For example, envisage the situation where each player starts the game with a finite pool of *reserves* that decreases in proportion to their movements (like the current stamina), but never gets replenished during the game. As a player's reserves becomes more and more depleted, his overall performance would start to be affected. To play effectively under this system, every player would clearly have to ration their efforts throughout the entire game — a longer-term notion of resource management than the current implementation of stamina requires, and also one which does not inherently bias against the kind of solo goal described in §3.1.

At any one time in the game, each of the players would generally have used a different amount of their reserves. This difference between players (where currently there is essentially none) expands greatly the opportunities for tactical reasoning. For example, players could be tactically re-shuffled, to concentrate the players with most reserves in the areas where they are most needed. Also, examining the opposing players for signs of weakness would become more profitable. Further, a meaningful use of substitution suggests itself: the coach client could be allowed to swap players who are running low on their reserves for a fresh player with full reserves. Choosing whether to station these fresh players in defence, midfield or attack would depend on the current game situation.

Vision. Finally in this section on basic skills, we give two quotes that illustrate the importance of being able to 'read the game'; the ability to predict future events and, ultimately, to dictate them.

Leboeuf has a fine touch and excellent distribution, but best of all is the way he sees things about ten minutes before they actually happen. He is an education. [20]

Injuries and his own immaturity have tarnished Gascoigne's career, but a trace of greatness is still visible. His superiority to the men around him is not just a matter of technique, of weight of pass and refinement of touch. It is intentions as much as skills that separate such a figure from the array of competent professionals. [21]

3.3 Playing Conditions

Database examples: *away* (1110), *supporters* (583), *stadium* (338), *pitch* (335), *crowd* (301), *fans* (217), *atmosphere* (50), *weather* (39), *rain* (36), *wind* (32), *rained* (12), *snow* (21).

These statistics from the database seem to indicate that the main external conditions that affect the outcome of a game are the supporters, and the atmosphere they create, rather than other factors such as the weather or the state of the pitch. Indeed one British club (Wimbledon) went as far as to hold a seminar about its atmosphere:

So concerned is the club at keeping the pot boiling, it is even staging a 'crowd atmosphere' seminar on December 2, at which the gathering will discuss ways of transforming Selhurst Park into a seething cauldron. [22]

Of course, crowd atmosphere is unlikely to ever have a formal role within RoboCup. In any case, it is often debated whether a crowd's efforts can actually sway the results of a game in a particular direction or whether they simply encourage *all* the players to raise their performance. As the Liverpool and England player Steve McManaman says about a trip to rivals Newcastle:

Personally, even though it can be quite intimidating there, I enjoy it immensely. At this level, few players freeze in such an atmosphere, and so it tends to give both sides a lift. There is nothing like playing in front of a passionate crowd, because you can't fail to be anything but inspired. [23]

3.4 Objectives

Database examples: *win* (1099), *points* (412), *draw* (400), *lead* (376), *beat* (372), *hope* (301), *injured* (186).

It is commonplace for real football teams to tailor their tactics and their mode of play to fit the objectives they have set for a particular game. Teams may try to slow down a game to silence the voices of a hostile away crowd, they

may try to play for points, or they may need to win by a certain number of goals to secure their position in a league. Currently in RoboCup it is unlikely that teams are sophisticated enough to modify their play significantly, but given the round-robin style of the tournaments, such an ability would certainly represent an advantage. Note also that a team's objectives can change dynamically during a game:

Rangers had enough ideas and possession to deserve at least a draw and probably would have got one had not Palace read the game so well; but Palace realised, after only 15 minutes of watching their opponents finessing the ball to no good purpose, that here was a match for the taking. [24]

One further twist is that once opposing teams start to observe each other's strategies it is possible for them to attempt to gain an advantage by misleading each other about their intentions:

Apparently our defence was so weak because Poland had cunningly tricked us. 'No, we will not be trying to score goals,' they announced beforehand, with their crossed fingers hidden behind their backs. 'Goals are overrated, in our opinion.' And we fell for it. Good grief. [25]

3.5 Attack

Database examples: *cross* (436), *header* (372), *area* (309), *attack* (229), *opening* (186), *opportunity* (155), *wing* (119), *winger* (116), *opportunities* (66), *deflection* (35), *one-two* (7), *one-twos* (2), *goalpoacher* (1).

The basic purpose of attack is to score goals by creating openings and opportunities. As a demonstration of the possible ways of creating chances, we have the following quote from Kevin Keegan:

Where were the crosses, where was the invention, where were the cute one-twos strikers thrive on? [26]

One aspect of real football that may not translate so well into the RoboCup domain is the use of wingers to attack down the sides of the field. In real games, such attacks often end with crosses, but in RoboCup the inability to kick the ball into the air may decrease the effectiveness of such balls.

It took him until ludicrously late to appreciate that the best way to negotiate a packed defence is to go round the back, using the wings. [27]

The delivery from the flanks, Hoddle acknowledges, is vital. [28]

It may be more interesting to see how the following argument carries over into RoboCup:

The controversy surrounding the merits of the long-ball theory and the possession game, exemplified by Brazil and Liverpool, is unresolved. Olsen argues that direct play is the most effective route to goal, the majority of goals stemming from three passes or less. [18]

3.6 Set Pieces

Database examples: *penalty* (606), *corner* (369), *set* (289), *free kick* (261).

By the very nature of set pieces, teams have more freedom to organise players than under normal game situations. Many teams in the Pre-RoboCup tournament, however, made poor use of set pieces. This is in contrast to real football, where some teams try to survive almost exclusively on set pieces alone, for example as in this description of Lincoln City:

Their only real ploy appears to be to pump the ball into intentionally soaked corners of the pitch, to win corners, free kicks and throw-ins around the penalty area. It is Pomo, the position of maximum opportunity, gone mad... [29]

3.7 Midfield

Database examples: *midfield* (700), *central* (218), *playmaker* (22)

The midfield is particularly important for the possession style of football mentioned in §3.5. Denying the opponents' possession in midfield is a question of controlling the space and of timing:

Of Ince, the ball winner and anchor in midfield, Hoddle said: 'He's appreciated when to tackle, when to hold off. Once he stepped into a higher level in Italy [with Internazionale] he learnt the right time; otherwise players there would be around and away from you.' [30]

Midfield can also give rise to specialised positions such playmakers who instigate attacks either by directly creating opportunities or directing the play into promising patterns:

There is therefore the renewed scope, Hoddle thinks, for the playmaker... Yet, he believes the game has moved away from the individual; that the contemporary formation of 3-5-2 — which he favours — calls for flexible midfield creativity. [31]

The following also describes another specialised role in midfield:

You're not outstanding at anything but you're very good at everything. You can tackle, you've got skill on the ball, you're good in the air and you read the game well. I've got the perfect position for you; the holding slot in front of the back four. I want you to dictate, get the ball, play it, set everything up, keep it all ticking over, but never get in front of the ball. Wherever it is, I want you behind it.' [32]

3.8 Defence

Database examples: *defender* (371), *clear* (280), *mark* (251), *save* (244).

One important feature of defending highlighted by our use of the Times database was the importance of being able to sense the presence of players and other objects even when they are behind you. Rear motion detection was incorporated into Soccer Server after the author posted the following quote to the RoboCup mailing list:

One of the many alarming things that can happen on a football field is disorientation. As the ball moves about, you can lose your sense of direction and the precise understanding of where everybody else is. That is why you constantly see defenders reaching out to grope the man they are marking. You need to know where he is, and where the ball is at the same time. [33]

The ability keep other players close helps to implement the general idea of defence as the complement of attack: the closing down of attacking players in an effort to prevent them creating the chances and opportunities needed to score goals. However, in the modern game, defence has become something more important than this. The ‘total football’ ideology preaches that defenders should have the ability to come forward as attackers. Indeed, echoing back to our previous classification of touch, passing and vision, there are those who advocate treating defence as the beginning of attack:

defenders are expected to be proficient in more than jumping, tackling and clearing their lines... [34]

‘Nowadays, at the back, it’s important that you’ve got a good touch, that you can pass the ball, read the game,’ Matteo said. There lies the key to Hoddle’s new road: he wants to build the game up from defence. [35]

In the final analysis, if there is one area in which an otherwise outclassed team should concentrate its attention it should be defence:

Ever since the radical change in the offside law in 1925 — from three defenders between opponent and goal to two — tactics have continually evolved. What has latterly changed most, however, is the defensive organisation of lesser teams and their advanced physical fitness, counteracting those which once dominated by technique. [18]

4 The Beautiful Game

It was Pelé, arguably the greatest player ever, who described football as ‘the beautiful game’. Looking at some word frequencies in our database, it is possible to partially support this epithet: *e.g.*, *beauty* (45), *beautiful* (42), *beautifully* (15), *beautifully-struck* (2), and *beautifully-weighted* (2). Is it too much to

hope that RoboCup programs will produce ‘beautiful’ moments, such as the following description of a shot that

...followed the trajectory of a ping-ponger’s forehand loop, from centre circle to goal in a gloriously walloped arcing topspin lob.
Beautiful indeed. [17]

This is a difficult question to answer. It is also debatable whether the goal of beautiful play is in fact a useful one. Even the journalists themselves express doubts about this:

Perhaps a player can only ride the wave of beauty so far. Perhaps other qualities — Linekeresque opportunism, Shearer-like aggression — are more lasting. [17]

The problems of declaratively describing football itself are even more apparent when dealing with beauty:

You can move the salt and pepper pots how you like to show the chess-like manoeuvring that led to the great strike, but that is to miss the point. You do not stroke your chin and say: hmm, that was a *good* goal. It is a cry of wonder from deep inside your guts. [12]

Note also that the essentially 2-D nature of the RoboCup will also hinder the learning of ‘beautiful’ play from examples in the 3-D world of real football games.

In real life, it has to be admitted that many teams succeed without playing beautifully, and in some cases by actively aiming for the opposite. For example, the following quote describes Wimbledon — an unfashionable team (‘no one likes us, we don’t care’) who are nevertheless enjoying great success this season, lying fourth in the English Premiership table at the time of writing.

They don’t play the beautiful game; they have perfected the hideous game, something like volleyball with a no-hands rule. [36]

In the final analysis, the old adage ‘nothing succeeds like success’ may well be the best guiding principle when designing RoboCup teams. Witness the following:

And of course, when it comes to goals, they all count, there are no marks for artistic impression and football is about winning, not about aesthetics. To get seduced by beauty is one of the most dangerous traps in football... [17]

This may be the bottom line:

...as a golf pro once vividly explained to me, there are no pictures on a scorecard. [36]

5 Summary

This paper has used an analysis of newspaper articles on football to identify basic features of the game. These basic features were examined in a way that should give insights to the designers of RoboCup teams and also inform the future development of the Soccer Server environment itself. We ourselves plan to use the basic breakdown of game features to design an effective fitness landscape over which evolutionary computation techniques can be used to develop client programs. As concrete examples of future Soccer Server developments, we suggested touchline coaching, a revised model of stamina, and the inclusion of substitutions.

We finished by looking at the concept of ‘beauty’ within the game of football. Perhaps the last words on this subject, and in this paper, should go to the ex-England manager Don Revie (as quoted in [17]):

As soon as it dawned on me that we were short of players who combined skill and commitment, I should have forgotten all about trying to play more controlled, attractive football and settled for a real bastard of a team.

References

1. D. A. Waterman. *A Guide to Expert Systems*. Addison-Wesley, 1986.
2. Alan Bundy. A science of reasoning. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 178–198. MIT Press, 1991. Also available from Edinburgh as DAI Research Paper 445.
3. D. Levy, editor. *Computer Games II*. Springer Verlag, 1988.
4. Simon Barnes. Newcastle toppled by Arsenal united. In *The Times*, London, December 2 1996.
5. Kevin McCarra. Pressure on Laudrup to open up Russians. In *The Times*, London, August 8 1996.
6. Rob Hughes. A goal fit to set before the football gods. In *The Times*, London, September 16 1996.
7. Simon Barnes. Middlesbrough’s birds of paradise have feathers ruffled. In *The Times*, London, December 30 1996.
8. Rob Hughes. New breed delivers best of both worlds. In *The Times*, London, February 24 1997.
9. Mark Hodgkinson. Hunt rescues point for bemused Francis. In *The Times*, London, Aug 26 1996.
10. Rob Hughes. United discover true measure of champions. In *The Times*, London, September 13 1996.
11. Mark Hodgkinson. Basement better than annexe for Rochdale. In *The Times*, London, February 21 1997.
12. Simon Barnes. Boyhood hero who gave England his best shot. In *The Times*, London, September 18 1996.
13. Russell Kempson. Bristol spoil a football fantasy. In *The Times*, London, February 24 1997.

14. Rob Hughes. Ferguson optimistic of further progress. In *The Times*, London, September 12 1996.
15. Rob Hughes. United on song for Europe. In *The Times*, London, October 17 1996.
16. Brian Glanville. Neill adds decisive ingredient to nostalgia at New Den. In *The Times*, London, September 2 1996.
17. Simon Barnes. Real beauty of Beckham's goal in life. In *The Times*, London, January 15 1997.
18. David Miller. The ills afflicting football's national health. In *The Times*, London, February 3 1997.
19. Andrew Longmore. Hard man now allying brain to brawn. In *The Times*, London, January 11 1997.
20. Simon Barnes. Injustice seen to be done by everyone but officials. In *The Times*, London, August 26 1996.
21. Kevin McCarra. Gascoigne underlines firm grasp of need for creativity. In *The Times*, London, September 30 1996.
22. Russell Kempson. Coventry show character to salvage a lost cause. In *The Times*, London, November 18 1996.
23. Steve McManaman. Winning way to avoid the holiday blues. In *The Times*, London, December 23 1996.
24. Simon Wilde. Rangers forget basic principles. In *The Times*, London, February 3 1997.
25. Lynne Truss. Seaman's goal was so busy, Poland's goalkeeper was reading a magazine'. In *The Times*, London, October 11 1996.
26. Andrew Longmore. Dream pairing in dire need of fine tuning. In *The Times*, London, August 19 1996.
27. Brian Glanville. Arsenal slow to use dismissals as credit cards. In *The Times*, London, September 30 1996.
28. Rob Hughes. Hoddle hopes Le Saux can outflank Italy. In *The Times*, London, January 31 1997.
29. David Maddock. Berkovic makes sure justice is done. In *The Times*, London, November 13 1996.
30. Rob Hughes. Hoddle not afraid to tackle the refereeing debate. In *The Times*, London, October 8 1996.
31. David Miller. Hoddle aiming to create the right blend. In *The Times*, London, October 9 1996.
32. Russell Kempson. Family man mixing amid the glitterati. In *The Times*, London, January 25 1997.
33. Simon Barnes. Ferguson brings old-fashioned values to the fore. In *The Times*, London, December 9 1996.
34. Richard Hobson. Giant killer seeks one more Cup victim. In *The Times*, London, November 15 1996.
35. Rob Hughes. Hoddle plans new national health service. In *The Times*, London, October 3 1996.
36. Lynne Truss. Wimbledon remain the true princes at Palace. In *The Times*, London, January 24 1997.

The Value of Project-Based Education in Robotics

Igor M. Verner

Technion - Israel Institute of Technology, Dept. of Education in Technology & Science,
32000 Haifa, Israel
ttrigor@tx.technion.ac.il

Abstract. This article deals with the educational values of the robotics tournaments as a form of project-oriented learning in engineering. The features of the project-oriented education, namely motivation, problem solving, time limits, interdisciplinary approach, team-work cooperation and applicable outcome, are discussed in relation to the Robotics area context. The RoboCup survey data about the participants, their motivation for participating in the program and preferences in the prospective summer school curriculum are summarized. Suggestions for incorporating educational activities in the RoboCup program are proposed.

1 Introduction

A number of robotics tournaments, such as the robot world cup initiative RoboCup [1], [2], [3], were held with great success during the past few years. The fielders at these robot football games are autonomous mobile robots designed by the competing university teams.

Robotics competitions attract an increasing number of universities and are arousing wide public interest. It becomes evident that perspectives of RoboCup development are not only in AI and intelligent robotics research but also in education. The reasons for the need to consider educational aspects of programs like the RoboCup are as follows:

- The majority of the team-members are students participating in the program during their studies;
- RoboCup projects may offer to the participants a wide range of modern technologies;
- Efforts for arranging robotics tournaments may be justified by their educational benefit.

The purpose of this article is to stress the educational values of the robotics tournaments as a form of project-oriented learning in robotics.

Project-based education currently attracts the keen interest of engineering educators. It is a way of teaching/learning by using key professional problems as the stimulus and focus of student activities [4].

Kolmos [5] distinguishes between three types of project works. In the *assignment project* the problem, the subject and the methods are defined by the supervisor. In

the *subject project* the students have a free choice either of problem or methods as to the subject defined by the supervisor. In the *problem project* the students have to analyze the problem, specified by the supervisor in general terms, and to determine the adequate disciplines and methods. The conclusion is that all three types of projects are necessary, since they lead to different kinds of knowledge and skills.

While aspiring to achieve the project goals, the student acquires important professional skills such as self-directed learning, problem solving, work planning and communication. Thus project-oriented learning is expected to increase the quality of learning [6] and to promote skills of effective professional retraining to new areas of expertise in response to technology changes, cross-functional teamwork and market dynamics [7].

In this article the features of project-based education in the area of robotics will be discussed. Statistical data about the participants and their attitudes towards RoboCup will be summarized. Some suggestions for incorporating educational activities in the RoboCup program will be proposed.

2 Features of Project-Oriented Learning in Robotics

Projects in robotics relate to professional experimentation in design, building and programming of robots. Making mechanical devices that are capable of performing human tasks or behavior has become a favorite subject for graduate student projects. Some universities are introducing projects in robotics into their undergraduate engineering programs [8,9]. Therefore, the definition of educational objectives, curricula and strategies for project-based education in robotics is now required.

The following features of project-oriented learning are emphasized: Motivation, problem solving, strict time limits, interdisciplinary approach, teamwork cooperation and an applicable outcome (Fig. 1). Let us consider some of these features in relation to the project-based education in robotics.

A. High Motivation to Learn. Motivation largely determines the student's behavior, especially the processes of active learning, that are central for project-based education. Motivation to learning may be divided into six factors [10]:

- M1:* A positive attitude towards the learning situation, the subject and the method;
- M2:* Awareness of the practical needs and values of the learning content;
- M3:* Extrinsic stimulation of learning results;
- M4:* Positive emotions within the learning process;
- M5:* Achieving the reward of competence;
- M6:* Reinforcement of the learner's abilities and skills.

Specification and analysis of these general factors in the context of project-based education in robotics is required in order to give recommendations on motivating RoboCup team-members. In this article we will make a first step in this direction - personal motives of the RoboCup participants will be statistically analyzed.

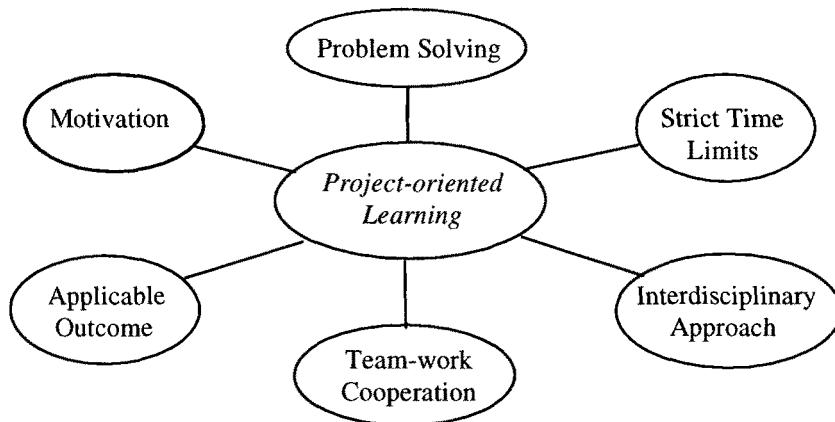


Fig. 1. The following features of project-oriented learning

B. The Emphasis on Problem Solving. Problem solving is the core of the developmental learning process. Developing the ability to recognize various approaches to the problem, generate and optimize solutions in relation to multiple criteria is crucial to the formation of engineers. Many educators believe that there is an essential difference between technological and general problem solving. To show this difference, the Polya's definition of the general problem solving steps can be compared with that proposed by Dekker for technological problem solving.

The Polya's GPS steps:

- (1) Define
- (2) Think about it
- (3) Plan
- (4) Carry out the plan
- (5) Look back

The Dekker's TPS steps:

- (1) Recognize a need
- (2) Accept the Challenge
- (3) Define the problem
- (4) Collect Information
- (5) Synthesize & Ideate
- (6) Analyze & Optimize
- (7) Evaluate
- (8) Implement

The reasonable position is that solving problems through all the steps given in the model of Dekker, from recognizing the need up to the implementation is more suitable for training in practical thinking and design skills.

RoboCup is stated as an AI and robotics research program focused on a specific problem - performing actual robot soccer competitions. It deals with challenging problems of robot design, control and behaviors, as well as adapting a wide range of modern technologies [3]. The “layered” building-up approach to constructing robots, implementing behaviors and their testing at the RoboCup competitions [11], [2] makes the program an ideal learning environment for involving students in problem solving activities. Special educational consideration on using this advantage is required.

C. Working Within Strict Time Limits. An important didactic goal of the project-based education is to develop the skills of time-planning and completing assignments on time. In this connection participation in the RoboCup program requires from the student careful planning of his project as part of the teamwork in order to stick to the schedule of the robot competitions.

D. The Interdisciplinary Approach. Robotics is an interdisciplinary subject. While building and programming robots, students acquire various theoretical and practical experiences in engineering science, including mechanics, electronics, mathematics and physics, computers and artificial intelligence. While participating in robotics projects, the students acquire basic technological knowledge and skills which they usually lack. Thus the RoboCup program can help eliminate the disparity of a strong science background vs. lack of basic technology skills, which is typical for a significant group of undergraduate engineering students.

E. Close Team-Work. Developing team-work cooperation skills is considered an important component of the engineering education process [7]. The project education framework gives the students initiative in work planning and decision making, thus stimulating close team cooperation. Being an international program of robot competitions, the RoboCup exceeds the usual project cooperation framework. The students, are involved in the following forms of cooperation:

- Collaboration with team-mates in developing the autonomous robots-“players”;
- Programming team cooperation behavior of the autonomous robots;
- Rule-guided competition with other teams;
- Programming rule-guided strategies of robot competition.

Openness and readiness for cooperation shown by the team members during the RoboCup competitions were mentioned by observers, including the author of this article. A study of the stimulating effect of participation in competitive team projects on developing team-work skills is required.

F. Applicable Outcome. Projects in robotics result in making mechanical devices ready to perform some specific assignments or behaviors. Robot competitions provide a way to evaluate objectively the project results.

The RoboCup outcomes are used as new methods and technologies of AI and intelligent robotics not intended for direct economic application. The argument in favor of focusing the RoboCup on *realistic* but not real world problems [3] is that the real problems need taking into account the specific domain and socio-economic factors, while many of the academic groups focus on fundamental research.

We accept this argument, but should mention that from the educational point of view, solving real practical problems is an inevitable component in formation of an engineer. Therefore it is required to find a rational combination of real and realistic problems for student projects performed in the frame of the RoboCup program.

Summing up the discussion on framework and objectives for project-oriented learning, we may conclude that projects in robotics made in the framework of the

competitive problem-centered programs like RoboCup provide the students with a direct insight into theoretical and practical engineering.

3 An Operative Questionnaire for RoboCup Team-Members

From the educational point of view RoboCup presents a new phenomenon in relation to the learning subject and teaching methods. Therefore it was decided to use a questionnaire in which we asked the team-members for some details on their participation in the RoboCup. The questionnaire was presented on the last day of the robot competitions. Answers were obtained from 77 respondents - the majority of the team-members.

The questionnaire includes four sections. The first section relates to general data (name, country and position). The second describes the participants' experience in RoboCup: Period of participation, form of participation (league) and role in the team.

In the third section the participant is asked about his/her motivation for participation in the RoboCup. The question is formulated in the following open-ended form: "To what extent are the factors mentioned below important for your participation in the RoboCup (from 1 to 5, 5 - very important)". The mentioned factors are:

- RoboCup challenge,
- Applying and testing your ideas,
- Practical training,
- The soccer idea
- Prestige of the program,
- Team work,
- Prospective research field,
- Perspective career opportunities,
- Game playing,
- Other motivation factors (name).

In the forth section the participant was asked about his/her preferences in the prospective RoboCup summer school curriculum. The open-ended question was the following: "To what extent is it important to include the following subjects in the summer school curriculum, in your opinion?" The list of subjects included:

- Overall lectures,
- Robot building workshop,
- Simulator programming workshop,
- Paper presentation,
- Panel discussions,
- Education.

4 Questionnaire Results

4.1 General Data and Experience in the RoboCup

Distribution of the team-members according to their positions in the universities is presented in Table 1.

Table 1. Positions of the team-members

Undergraduate students	Graduate students	Postgraduate students	Research & ing.	assist.	Professors
15	18	12	11		18

One can see that the teams are recruited mainly from students. Many of them are graduate students though the numbers of undergraduate and postgraduate students are also significant.

The distribution of the respondents according to the period of participation in the robot competitions is given in Table 2.

Table 2. Period of participation in RoboCup

Less than half a year	From half to 1 year	From 1 to 2 years	Over 2 years
37	25	11	4

According to the table, the majority of the team members have been involved in the RoboCup for less than a year. Almost all the undergraduate and many graduate students have participated in the program for less than half a year. Their successful performance during the robot competitions in Nagoya indicated that the task of design and building robots for a robot-soccer competitions may be completed by graduate or even undergraduate students within one/two semesters of team project work.

The RoboCup competitions were held in middle size, small size and simulation leagues. The number of respondents participating in each league is given in Table 3.

Table 3. League

Middle size	Small size	Simulation
29	14	36

4.2 Motivation for Participating in RoboCup

The questionnaire results related to the personal motivation factors are summarized in Table 4. In the first (left) column of the table the motivation factors mentioned in

the questionnaire are listed in descending order, according to the average grade (for all the respondents) given to each factor.

The average grades are listed in the second column (AG/all). The third column (VI) presents the number of respondents who assigned the highest grade 5 to each motivation factor, i.e. consider it as very important for their participation in the RoboCup. The fourth column (AG/Prof.) includes the average grades attributed to each factor by a certain part of the respondents, namely professors.

The motivation factors in the questionnaire specify the general motivation factors (categories *M1* - *M6*) mentioned above. The general motivation factors specified by each of the specific motivation factors are given in the last (right) column (Cat.).

One can see that the motive mentioned as very important by most of the respondents is the opportunity to apply and test their own ideas. This evidence enables us to assume that many team-members participate in the RoboCup mainly for the purpose of doing research.

Table 4. Motivation for participation in RoboCup

Motivation factors	AG/all	VI	AG/Prof.	Cat.
1. Applying and testing your ideas	4.41	47	4.65	M6
2. Team work	3.98	32	4.12	M6
3. Prospective research field	3.84	27	4.06	M1
4. RoboCup challenge	3.75	23	3.94	M5
5. Practical training	3.51	22	3.53	M6
6. Game playing	3.08	9	2.47	M4
7. Prestige of the program	3.02	12	2.82	M1
8. The soccer idea	2.87	11	2.71	M4
9. Prospective career opportunities	2.76	5	2.23	M2
10. Others	2.48	12	2.23	M3

However, the educational motivation factors are also important. In particular, “applying and testing ideas” together with “team work” and “practical training” specify the general motivation factor of reinforcement of personal abilities and skills.

The list of motivation factors given in the questionnaire, of course, does not exhaust all possible reasons for personal participation in the RoboCup. Therefore the respondents were asked about importance of other motivation factors and requested to name them. As follows from Table 4, the average grade (for all the respondents) of the other motivation factors is quite low (2.48), though these factors were considered by some respondents as very important. The mentioned “other” factors, such as travel grants, visiting Japan, meeting students from other countries, relate mainly to the “extrinsic motivation” category.

The following features of the questionnaire data on personal motivation should be mentioned:

- ⇒ High level motivation - almost all respondents assigned “very important” to some of the motivation factors.
- ⇒ Average grades given by the professors do not differ essentially from those given by all the respondents.
- ⇒ Some motivation factors have relatively low average grades, though each of them is very important for a certain part of the respondents.
- ⇒ The unjustified low average grade (2.76) was given to the motivation factor of “Prospective career opportunities”.

Personal interviews indicated that some of the students are not aware of prospective career opportunities in the area of Robotics and Artificial Intelligence. As an example of successful professional employment of the graduates we would like to quote the interview of a female graduate from the U.S. She said: “I have a Bachelor of Science in Math and Computer Science with a special technical area of interest in Robotics and Artificial Intelligence. I am currently a Robotics Engineer at ... in California. After graduating in May 97, I wanted to find a job in my field. I feel that my education prepared me for a variety of job opportunities in the field of computer science. There is a very small job market of available in Robotics and AI. However, with my background I was able to find a position with a mobile robotics company. I really enjoy working in this field of research and keeping up with the ever-expanding technologies. My motivation for pursuing a career in Robotics was to gain experience in a variety of engineering aspects: Mechanical, electrical, and computer science. This had allowed me to be competitive on the job market.”

4.3 Subjects for a Prospective Robot Summer School

The results for this section of the questionnaire are summarized in Table 5. The form of this table is similar to that of Table 4 but it includes only three columns. In the first (left) column the subjects mentioned in the questionnaire are listed in descending order, according to the average grades (for all respondents). These average grades (AG/all) are listed in the second column. The third (right) column (AG/Prof.) presents average grades given by the professors.

Table 5. Preferences in the prospective summer school curriculum

Subjects for 1998 Summer School	AG/all	AG/Prof.
1. Simulator programming workshop	4.06	3.94
2. Robot building workshop	4.04	4.06
3. Education	3.69	3.59
4. Paper presentation	3.41	2.94
5. Panel discussion	3.32	3.24
6. Observing lectures	3.05	3.31

The following characteristic features of the data are:

- ⇒ Workshops on simulator programming and robot building were mentioned as the most important subjects by the majority of respondents, including professors.
- ⇒ In fact, the respondents who participated in the simulation league noted the importance of a robot building workshop, and vice versa, the participants at the small and middle size leagues showed interest in the simulator programming workshop.
- ⇒ Graduate and postgraduate students more than others are interested in presenting papers and participating in panel discussions.
- ⇒ A high average grade was assigned to education, especially by the students.

Personal interviews indicated that incorporating education into the RoboCup program will help to attract new participants. The undergraduate and graduate students performing specific team assignments are interested in learning more about the whole subject, while the postgraduate students and professors want to discuss teaching methods, curricula and evaluation.

5 Challenge for High-School Technology Education

Technology education in high school is undergoing a world-wide reform in relation to its status, goals and teaching/learning strategies. As a result of changes in the high

school curriculum, which were caused by the reform, robotics is becoming part of the science-technology discipline in junior and senior high-schools. Junior high school curricula may include short-term projects in robotics dealing with the basics of Lego construction and programming. Senior high-school projects can last one or several terms, and be based on elements of robot design and programming intelligent tasks. A pilot course "Robotics and real time systems" which was conducted in a number of senior high schools in Israel and several assignments that were implemented through the school projects are described in our article [12].

Incorporating such projects into competitive programs like RoboCup will stimulate the interest of high school students in the project-based science-technology education. A possible direction for active participation of high school students in the RoboCup is implementing specific elementary skills and diverse service functions around the football game. These service tasks may be game field care, ball pick up, champion reward, etc. Service functions should be performed by means of robots, including elements of AI.

An additional opportunity to participate in project-oriented learning, particularly in robotics, may be offered by universities to interested high school students. For example, an annual international summer research program for youth is conducted by the Technion (SciTech home page www.technion.ac.il/pub/projects/scitech/).

Conclusions

1. RoboCup is principally a research program in Robotics and AI. However, its value as a competitive project-oriented engineering education program is also important and should be taken into account.
2. The task of basic-level design, building robots and programming behaviors for robot soccer competitions is a suitable project assignment for graduate and even undergraduate student teams.
3. The results of the operative questionnaire (RoboCup Survey) present objective data about the participants, their motivation for participating in the program and preferences in the prospective summer school curriculum, that can be used for evaluation and planning.

Acknowledgments

The author would like to thank Hiroaki Kitano and the RoboCup-97 Workshop Committee for supporting the RoboCup Survey initiative. This work was funded in part by the E. Schaver Research Fund.

References

1. Kitano, H., Kuniyoshi, Y., Noda, I., Asada, M., Matsubara, H., Osawa, E.: RoboCup: A Challenge Problem for AI, *AI Magazine*, 18(1) (1997) 73-85
2. Kitano, H., Veloso, M., Matsubara, H., Tambe, M., Coradeschi, S., Noda, I., Stone, P., Osawa, E., Asada, M.: The RoboCup Synthetic Agent Challenge 97. Proc. of 15th International Joint Conference on Artificial Intelligence, Vol. 1 (1997) 24-29
3. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: RoboCup: The Robot World Cup Initiative. Proc. of the First International Conference on Autonomous Agents (1997) 340-347
4. Schachterle, L., Vinther, O.: Introduction: The Role of Projects in Engineering Education, *European Journal of Engineering Education*, 21(2) (1996) 115-120
5. Kolmos, A.: Reflections on Project Work and Problem-based Learning, *European Journal of Engineering Education*, 21(2) (1996) 141-148
6. Gibbs, G.: Improving the Quality of Learning, Bristol, Technical & Educational Services (1992)
7. Gates, A., Della-Piana, C., Bernat, A.: Affinity Groups: A Framework for Developing Workplace Skills. Proc. of 1997 Frontiers in Education Conference, 1 (1997) 53-56
8. Manseur, R.: Development of an Undergraduate Robotics Course. Proc. of 1997 Frontiers in Education Conference, 2 (1997) 610-612
9. Mehrl, D., Parten, M., Vines, D.: Robots Enhance Engineering Education. Proc. of 1997 Frontiers in Education Conference, 2 (1997) 613-618
10. Wlodkowski, R. J.: Enhancing adult motivation to learn. San Francisco, CA: Jossey Bass Publ. (1984)
11. Brooks, R.: Elephants don't play chess. In: Maes, P. (ed.): *Designing Autonomous Agents*, MIT Elsevier (1991) 3-15
12. Verner, I., Waks, S., Kolberg E.: High School Sci-Tech Project - An Insight into Engineering. Proc. of 1997 Frontiers in Education Conference, 2 (1997) 949-954

The CMUnited-97 Small Robot Team

Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim

Computer Science Department

Carnegie Mellon University

Pittsburgh, PA 15213

{veloso, pstone, kwunh, sorin}@cs.cmu.edu

Abstract. Robotic soccer is a challenging research domain which involves multiple agents that need to collaborate in an adversarial environment to achieve specific objectives. In this paper, we describe CMUnited, the team of small robotic agents that we developed to enter the RoboCup-97 competition. We designed and built the robotic agents, devised the appropriate vision algorithm, and developed and implemented algorithms for strategic collaboration between the robots in an uncertain and dynamic environment. The robots can organize themselves in formations, hold specific roles, and pursue their goals. In game situations, they have demonstrated their collaborative behaviors on multiple occasions. The robots can also switch roles to maximize the overall performance of the team. We present an overview of the vision processing algorithm which successfully tracks multiple moving objects and predicts trajectories. The paper then focusses on the agent behaviors ranging from low-level individual behaviors to coordinated, strategic team behaviors. CMUnited won the RoboCup-97 small-robot competition at IJCAI-97 in Nagoya, Japan.

1 Introduction

As robots become more adept at operating in the real world, the high-level issues of collaborative and adversarial planning and learning in real-time situations are becoming more important. An interesting emerging domain that is particularly appropriate for studying these issues is Robotic soccer, as first proposed by [9] and actively pursued within the RoboCup initiative [7, 1]. Although realistic simulation environments exist [10, 11] and are useful, it is important to have some physical robotic agents in order to address the full complexity of the task.

Robotic soccer with real robots is a challenging domain for many reasons. The fast-paced nature of the domain necessitates real-time sensing coupled with quick behaving and decision making. Furthermore, the behaviors and decision making processes can range from the most simple reactive behaviors, such as moving directly towards the ball, to arbitrarily complex reasoning procedures that take into account the actions and perceived strategies of teammates and opponents. Opportunities, and indeed demands, for innovative and novel techniques abound.

One of the advantages of Robotic Soccer is that it enables the direct comparison of different systems: they can be matched against each other in competitions.

In particular, the system described here was designed specifically for RoboCup97 in which several robotic teams competed on an “even playing field.” [6]. The scientific opportunities involved in this effort are enormous. Our particular scientific focus is on multiagent systems coupled with collaborative and adversarial learning in an environment that requires real-time dynamic planning.

This paper describes the overall architecture of our robotic soccer system. The combination of robust hardware, real-time vision, and intelligent control represented a significant challenge which we were able to successfully meet. The work described in this paper is fully implemented as our CMUnited-97 RoboCup team. CMUnited-97 won the RoboCup-97 small-robot competition at IJCAI-97 in Nagoya, Japan. Our team scored a total of thirteen goals and only suffered one. Figure 1 shows a picture of our robotic agents.

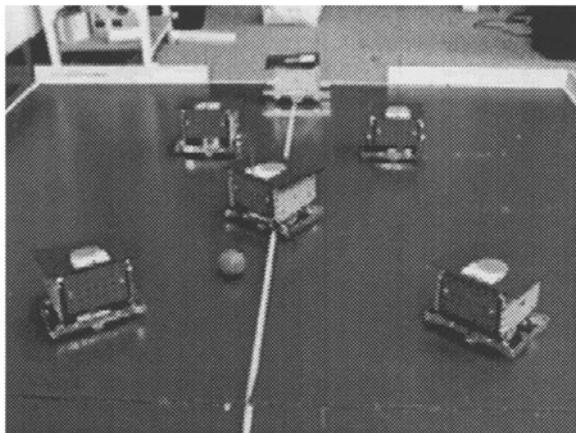


Fig. 1. The CMUnited-97 robot team that competed in RoboCup-97.

The specific contributions of the CMUnited-97 robot team, as presented in this paper, include:

- The complete design and development of robots with robust navigation and communication hardware.
- Reliable perception through the use and extension of a Kalman-Bucy filter. Sensing through our vision processing algorithm allows for (i) tracking of multiple moving objects; (ii) and prediction of object movement, particularly the ball, even when inevitable sharp trajectory changes occur.
- Multiagent strategic reasoning. Collaboration between robots is achieved through: (i) a flexible role-based approach by which the task space is decomposed and agents are assigned subtasks; (ii) a flexible team structure by which agents are organized in *formations*, homogeneous agents flexibly switch roles within formations, and agents switch formations dynamically; and (iii) alternative plans allowing for collaboration (e.g. passing or shooting), are controlled by pre-defined metrics for real-time evaluation.

2 Overall Architecture

The architecture of our system addresses the combination of high-level and low-level reasoning by viewing the overall system as the combination of the robots, a vision camera over-looking the playing field connected to a centralized interface computer, and several clients as the minds of the small-size robot players. Figure 2 sketches the building blocks of the architecture.

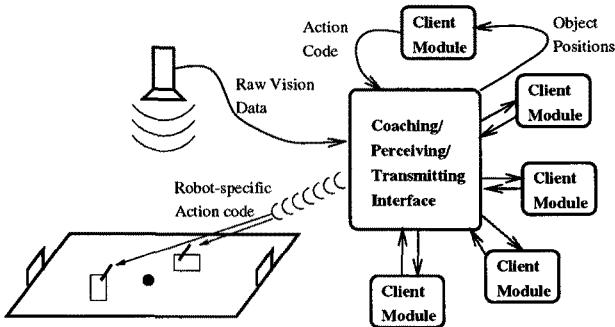


Fig. 2. CMUnited-97 Architecture with Global Perception and Distributed Reaction.

The complete system is fully autonomous consisting of a well-defined and challenging processing cycle. The global vision algorithm perceives the dynamic environment and processes the images, giving the positions of each robot and the ball. This information is sent to an off-board controller and distributed to the different agent algorithms. Each agent evaluates the world state and uses its strategic knowledge to decide what to do next. Actions are motion commands that are sent by the off-board controller through radio communication. Commands can be broadcast or sent directly to individual agents. Each robot has an identification binary code that is used on-board to detect commands intended for that robot.

The fact that perception is achieved by a video camera that over-looks the complete field offers an opportunity to get a global view of the world state. Although this setup may simplify the sharing of information among multiple agents, it presents a challenge for reliable and real-time processing of the movement of multiple moving objects – in our case, the ball, five agents on our team, and five agents on the opposing team (see Section 3).

The robots are built using two coreless DC motors (differential drive) with built-in magnetic encoders. The various speed and acceleration values are obtained by two specialized motion control processors (PID) which are controlled by a 8-bit micro-controller. The robots are equipped with an on-board two-way radio link, which allows for data exchange with an off-board processing unit. This wireless communication link also enables sending and receiving data to and from other robots at speeds of up to 40 Kbit/s.

One of the interesting characteristics of our robots is that the core electronics is separated from the outside frame. The robot dimensions are as follows:

- Length: limited by the size of the circuit board, which is 9.4cm; actual total length depends on the frame.
- Width: limited by the size of the wheels and motors' axis, which is 9.4cm.
- Frame: we have three sizes of frames, namely a base frame – 12cm × 12cm, the elongated frame – 18cm × 10cm, and a wide frame – 15cm × 12cm.
- Height: 12 cm; Weight: 1.5 lb

The base frame allows for various configurations of the final external frame. The frames are articulated at the edges and made of perforated steel strips and sheets. The flexible frame structure allows for the easy access to the components and easy use of variations of frames, as a function of the purpose of the robots.

Around 60% of the robot weight is made up by the battery packs. The on-board electronics include an 8-bit micro-controller, an on-board memory of 512 bytes RAM, and 2 Kbyte EEPROM, and a half-duplex FM 418 MHz radio.

The dimensions of the circuit boards could be adapted to fit different shapes and sizes of the main frame. More and smaller boards can be stacked inside the main frame, making it possible to incorporate other components if required.

We created a *command server* for handling commands from the individual off-board robot reasoning processes. The radio control processor is connected to the server computer via one serial link. Thus individual "brains" from networked machines must communicate to their robot "bodies" through the server computer. One of the command server's roles is to collect and translate these commands and to send them to the radio control processor.

3 Real-Time Perception for Multiple Agents

The vision requirements for robotic soccer have been examined by different researchers [12, 13]. Systems with on-board and off-board types have appeared in recent years. All have found that the reactivity of soccer robots requires a vision system with a high processing cycle time. However, due to the rich visual input, researchers have found that dedicated processors or even DSPs are often needed [2, 12]. We currently use a frame-grabber with frame-rate transfer from a 3CCD camera. A 166MHz Pentium processor is dedicated to the vision processing.

3.1 Color-based Detection

The RoboCup rules specify well defined colors for different objects in the field and these are used as the major cue for object detection. Our vision-processing is therefore color based. Teammates and opponents are identified by blue and yellow circles. We add an additional pink patch for each of our robots to determine teammate orientation. The ball is an orange golf ball (see Figure 1).

Noise is inherent in all vision systems. False detections in the current system are often of a magnitude of 100 spurious detections per frame. The system eliminates false detections via two different methods. First, color patches of size not matching the ones on the robots are discarded. This technique filters off most “salt and pepper” noise. Second, by using a minimum-distance data association mechanism, all false detections are eliminated.

3.2 Data Association

The color-based detection algorithm returns an unordered list of robots for each frame. To be able to control the robots, the system must associate each detected robot in the field with a robot identification.

Each of the robots is fitted with the same color tops and no attempts are made to differentiate them via color hue. Experience has shown that, in order to differentiate five different robots by hue, five significantly different hues are needed. However, the rules of the RoboCup game eliminate green (field), white (markings), orange (ball), blue and yellow (team and opponent) from the list of possibilities. Furthermore, inevitable variations in lighting conditions over the area of the field and noise in the sensing system are enough to make a hue-based detection scheme impractical.

With each robot fitted with the same color, visually, all robots on the same team appear identical to the visual system. Data association addresses the problem of retaining robot identification in subsequent frames. We devised an algorithm to retain association based on the spatial locations of the robots.

We assume that the starting positions of all the robots are known. This can be done trivially by specifying the location of the robots at the start time. However, as subsequent frames are processed, the locations of the robots change due to robot movements (due to controlled actions or adversarial pushes). Association can be achieved by making two complementary assumptions: 1) Robot displacements over consecutive frames are local; 2) The vision system can detect objects at a constant frame rate. By measuring the maximum robot velocity, we know that in subsequent frames, the robot is not able to move out of a 3cm radius circular region. This knowledge provides the basis of our association technique.

These assumptions provide the basis to the minimum distance scheme that we devised to retain association between consecutive frames. During consecutive frames, association is maintained by searching for objects within a minimum displacement. Current robot positions are matched with the closest positions from the previous frame. Our greedy algorithm searches through all possible matches, from the smallest distance pair upwards. Whenever a matched pair is found, it greedily accepts it as a matching pair.

Due to noise, it is possible for the detection system to leave a robot undetected. In this case, the number of robots detected on the past frame is larger than the number of robots in the current frame. After the greedy matching process, the remaining unmatched positions from the past frame will be carried over to the current frame. The robots corresponding to the unmatched locations will be assumed to be stationary. Note that if the missing rate of the

detection algorithm were high and frequent, this minimum distance association technique would easily lose the robots. However, this greedy algorithm was used in RoboCup-97 successfully, showing its reliability when combined with our accurate and robust vision detection system.

3.3 Tracking and Prediction

In the setting of a robot soccer game, the ability to merely detect the locations of objects on the field is often not enough. Like for real soccer players, it is often essential for robots to predict future locations of the ball (or of the other players). We have used an Extended Kalman-Bucy filter (EKF) [5] for ball movement prediction. The EKF is a recursive estimator for a possibly non-linear system. It involves a two-step iterative process, namely *update* and *propagate*. The current best estimate of the system's state and its error covariance is computed on each iteration. During the update step, the current observations are used to refine the current estimate and recompute the covariance. During the propagate step, the state and covariance of the system at the next time step are calculated using the system's equations. The process then repeats, alternating between the update and the propagate steps.

We capture the ball's state with five variables: the ball's x and y location, the ball's velocities in the x and y direction and a friction parameter (λ_k) for the surface. The system is represented by the following non-linear difference equations:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \dot{x}_{k+1} \\ \dot{y}_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k \cdot \Delta t \\ y_k + \dot{y}_k \cdot \Delta t \\ \dot{x}_k \cdot \lambda_k \\ \dot{y}_k \cdot \lambda_k \\ \lambda_k \end{bmatrix} \quad \begin{aligned} \dot{x}_{k+1} &= \dot{x}_k \cdot \lambda_k \\ \dot{y}_{k+1} &= \dot{y}_k \cdot \lambda_k \\ \lambda_{k+1} &= \lambda_k \end{aligned}$$

The equations model the ball with simple Newtonian dynamics, where λ_k is a friction term which discounts the velocity at each time step, and Δt is the time-step size.

The prediction equations are:

$$\begin{aligned} x_{k+n} &= x_k + \dot{x}_k \cdot \Delta t \cdot \alpha_{kn} \\ y_{k+n} &= y_k + \dot{y}_k \cdot \Delta t \cdot \alpha_{kn} \\ \alpha_{kn} &= \begin{cases} 1 & \text{if } \lambda_k = 1 \\ (1 - (\lambda_k)^n) / (1 - \lambda_k) & \text{otherwise} \end{cases} \end{aligned}$$

The prediction equations are derived by solving the recursive equation obtained by substituting the value of x_{k+i} where i decreases from n to 1. We are only interested in the predicted spatial location of the ball thus we do not explicitly calculate the predicted velocity.

Through a careful adjustment of the filter parameters modelling the system, we were able to achieve successful tracking and, in particular prediction of the ball trajectory, even when sharp bounces occur.

Our vision processing approach worked perfectly during the RoboCup-97 games. We were able to detect and track 11 objects (5 teammates, 5 opponents and a ball) at 30 frames/s. The prediction provided by the EKF allowed the goal-keeper to look ahead in time and predict the best defending position. During the game, no goals were suffered due to miscalculation of the predicted ball position.

4 Multiagent Strategy Control

We achieve multiagent strategy through the combination of accurate individual and collaborative behaviors. Agents reason through the use of persistent reactive behaviors that are developed to aim at reaching team objectives.

4.1 Single-agent Behaviors

In order to be able to successfully collaborate, agents require robust basic skills. These skills include the ability to generate a path to given location, and the ability to handle the ball, namely to direct the ball in a given direction, and to intercept a moving ball. All of these skills must be executed while avoiding obstacles such as the walls and other robots.

Non-holonomic path generation

The non-holonomic path planning problem has been addressed by many researchers, e.g., [8, 4]. However, most of the algorithms deal with static worlds and generate pre-planned global paths. In the robot soccer domain, this is not possible as the domain is inherently dynamic and response times need to be very high. Furthermore, the world dynamics include also possible interference from other robots (e.g., pushing), making precisely mapped out paths ineffective and unnecessary.

We devised and implemented a reactive controller for our system, which is computationally inexpensive, deals with dynamic environments, and recovers from noisy command execution and possible interferences. A reactive controller also has possible disadvantages, as it may generate sub-optimal paths, due to local minima. We introduced a failure recovery routine to handle such failures.

The navigational movement control for our robots is hence done via reactive control. The control rules described below are inspired by the Braitenburg vehicle [3]. The Braitenburg *love vehicle* defines a reactive control mechanism that directs a differential-drive robot to a certain target. A similar behavior is required in the system; however, the love vehicle's control mechanism is too simplistic and, in some start configurations, tends to converge to the goal very slowly. We devised a modified set of reactive control formulae that allows for effective adjustment of the control trajectory:

$$(v, \dot{\theta}) = \begin{cases} (\alpha \cdot \sin \theta, \beta \cdot \cos \theta) & \text{if } |\theta| < \frac{\pi}{4} \text{ or } |\theta| > \frac{3\pi}{4} \\ (0, \text{sgn}(\theta) \times \beta_0) & \text{otherwise} \end{cases}$$

where v and $\dot{\theta}$ are the desired translational and rotational velocities, respectively, θ is the direction of the target relative to the robot ($-\pi < \theta < \pi$), β_0 is the in-place rotational velocity, and α and β are the base translational and rotational velocities, respectively. The translational and rotational velocities can be translated to differential drive parameters via a simple, invertible linear transform. This set of control formulae differs from the love vehicle in that it takes into account the orientation of the robot with respect to the target and explicitly adds rotational control. This set of control rules implicitly allows for heading independence, i.e., the control rules allow for both forward and backward movements, whichever one is most efficient to execute. Figure 3 shows an actual run of the reactive control algorithm described above.

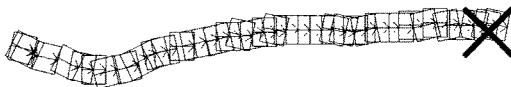


Fig. 3. Sample trace of the execution of the reactive control algorithm. The target point is marked with a cross.

Ball handling

If a robot is to accurately direct the ball towards a target position, it must be able to approach the ball from a specified direction. Using the ball prediction from the vision system, the robot aims at a point on the far side of the target position. The robots are equipped with two methods of doing so:

- **Ball collection:** Moving behind a ball and knocking it towards the target.
- **Ball interception:** Waiting for the ball to cross its path and then intercepting the moving ball towards the target.

When using the ball collection behavior, the robot considers a line from the target position to the ball's current or predicted position, depending on whether or not the ball is moving. The robot then plans a path to a point on the line and behind the ball such that it does not hit the ball on the way and such that it ends up facing the target position. Finally, the robot accelerates to the target. Figure 4(a) illustrates this behavior.

When using the ball interception behavior (Figure 4(b)), on the other hand, the robot considers a line from *itself* to the target position and determines where the ball's path will intersect this line. The robot then positions itself along this line so that it will be able to accelerate to the point of intersection at the same time that the ball arrives.

In practice, the robot chooses from between its two ball handling routines based on whether the ball will eventually cross its path at a point such that the robot could intercept it towards the goal. Thus, the robot gives precedence to the ball interception routine, only using ball collection when necessary. When using

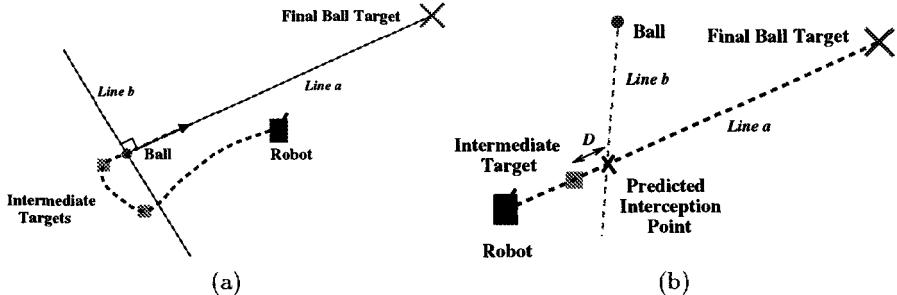


Fig. 4. Single-agent behaviors to enable team collaboration (a) Ball collection (aiming for a pass or to the goal); (b) Ball interception (receiving a pass).

ball collection, it actually aims at the ball's predicted location a fixed time in the future so as to eventually position itself in a place from which it can intercept the ball towards the target.

Obstacle avoidance

In the robotic soccer field, there are often obstacles between the robot and its goal location. Our robots try to avoid collisions by planning a path around the obstacles. Due to the highly dynamic nature of this domain, our obstacle avoidance algorithm uses closed-loop control by which the robots continually replan their goal positions around obstacles. In the event that an obstacle blocks the direct path to the goal location, the robot aims to one side of the obstacle until it is in a position such that it can move directly to its original goal. Rather than planning the entire path to the goal location at once, the robot just looks ahead to the first obstacle in its way under the assumption that other robots are continually moving around. Using the reactive control described above, the robot continually reevaluates its target position. For an illustration, see Figure 5. The robot starts by trying to go straight towards its final target along line a.

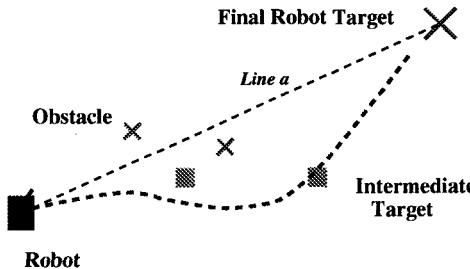


Fig. 5. Obstacle avoidance through dynamic generation of intermediate targets.

When it comes across an obstacle within a certain distance of itself and of line a, it aims at an intermediate target to the side, and slightly beyond the obstacle. The robot goes around the obstacle the short way, unless it is at the edge of the

field. Using reactive control, the robot continually recomputes line a until the obstacle is no longer in its path. As it comes across further obstacles, it aims at additional intermediate targets until it obtains an unobstructed path to the final target.

Even with obstacle avoidance in place, the robots can occasionally get stuck against other robots or against the wall. Particularly if opponent robots do not use obstacle avoidance, collisions are inevitable. When unable to move, our robots identify the source of the problem as the closest obstacle and “unstick” themselves by moving away. Once free, normal control resumes.

4.2 Multiagent Behaviors

Although the single-agent behaviors are very effective when just a single robot is on the field, if all five robots were simultaneously chasing the ball and trying to shoot it at the goal, chaos would result. In order to achieve coordinated multiagent behavior, we organize the five robots into a flexible team structure.

The team structure, or *formation*, defines a set of roles, or *positions* with associated behaviors. The robots are then dynamically mapped into the positions. Each robot is equipped with the knowledge required to play any position in each of several formations.

The positions indicate the areas of the field which the robots should move to in the default situation. There are also different *active modes* which determine when a given robot should move to the ball or do something else instead. Finally, the robot with the ball chooses whether to shoot or pass to a teammate using a passing evaluation function.

These high-level, multiagent behaviors were originally developed in simulation and then transferred over to the robot-control code. Only the run-time passing evaluation function was redefined. Further details, particularly about the flexible team structures, are available in [14].

Positions, Formations, and Active Modes

Positions are defined as flexible regions within which the player attempts to move towards the ball. For example, a robot playing the “right-wing” (or “right forward”) position remains on the right side of the field near the opponents’ goal until the ball comes towards it. Positions are classified as defender, midfielder, forward based on the locations of these regions. They are also given behavior specifications in terms of which other positions should be considered as potential pass-receivers.

At any given time each of the robots plays a particular position on the field. However, each robot has all of the knowledge necessary to play any position. Therefore the robots can—and do—switch positions on the fly. For example, robots A and B switch positions when robot A chases the ball into the region of robot B. Then robot A continues chasing the ball, and robot B moves to the position vacated by A.

The pre-defined positions known to all players are collected into formations, which are also commonly known. An example of a formation is the collection

of positions consisting of the goalkeeper, one defender, one midfielder, and two attackers. Another possible formation consists of the goalkeeper, two defenders and two attackers. For illustration, see Figure 6.

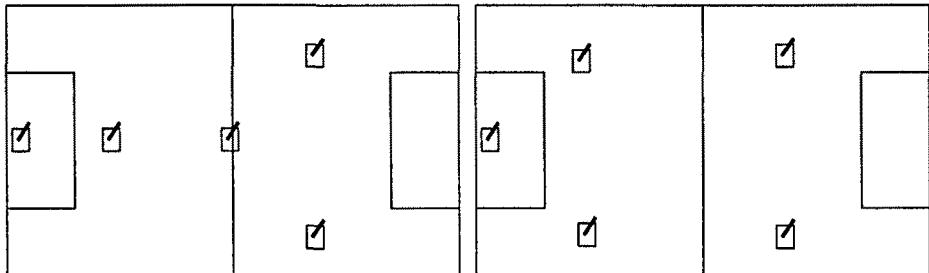


Fig. 6. Two different defined formations. Notice that several of the positions are reused between the two formations.

As is the case for position-switches, the robots switch formations based on pre-determined conditions. For example, if the team is losing with very not much time left in the game, the robots would switch to a more offensive formation. On the other hand, if winning, they might choose a defensive formation. The precise conditions for switching positions and formations are decided upon in advance, in what we call a “locker-room agreement,” [14] in order to eliminate the need for complex on-line negotiation protocols.

Although the default action of each robot is to go to its position and face the ball, there are three *active modes* from which the robot must choose. The default position-holding behavior occurs when the robot is in an *inactive* state. However, when the ball is nearby, the robot changes into an *active* state. In the active state, the robot moves towards the ball, attempting either to pass it to a teammate or to shoot it towards the goal based on an evaluation function that takes into account teammate and opponent positions. A robot that is the intended receiver of a pass moves into the *auxiliary* state in which it tries to intercept a moving ball towards the goal. Our current decision function sets the robot that is closest to the ball into the active state; the intended receiver robot (if any) into the auxiliary state; and all other robots into the inactive state.

Run-time Evaluation of Collaborative Opportunities

One of CMUnited-97’s main features is the robots’ ability to collaborate by passing the ball. When in active mode, the robots use an evaluation function that takes into account teammate and opponent positions to determine whether to pass the ball or whether to shoot. In particular, as part of the formation definition, each position has a set of positions to which it considers passing. For example, a defender might consider passing to any forward or midfielder, while a forward would consider passing to other forwards, but not backwards to a midfielder or defender.

For each such position that is occupied by a teammate, the robot evaluates the pass to that position as well as evaluating its own shot. To evaluate each possible pass, the robot computes the *obstruction-free-index* of the two line segments that the ball must traverse if the receiver is to shoot the ball (lines b and c in Figure 7). In the case of a shot, only one line segment must be considered (line a). The *value* of each possible pass or shot is the product of the relevant obstruction-free-indices. Robots can be biased towards passing or shooting by further multiplying the values by a factor determined by the relative proximities of the active robot and the potential receivers to the goal. The robot chooses the pass or shot with the maximum value. The obstruction-free-index of line segment l is computed by the following algorithm (variable names correspond to those in Figure 7):

1. *obstruction-free-index* = 1.
2. For each opponent O :
 - Compute the distance x from O to l and the distance y along l to l 's origin, i.e. the end at which the ball will be kicked by the robot (See Figure 7).
 - Define constants *min-dist* and *max-denominator*. Opponents farther than *min-dist* from l are not considered. When discounting *obstruction-free-index* in the next step, the y distance is never considered to be larger than *max-denominator*. For example, in Figure 7, the opponent near the goal would be evaluated with $y = \text{max-denominator}$, rather than its actual distance from the ball. The reasoning is that beyond distance *max-denominator*, the opponent has enough time to block the ball: the extra distance is no longer useful.
 - if $x < \text{min-dist}$ and $x < y$,
 $\text{obstruction-free-index} *= x/\text{MIN}(\text{max-denominator},y)$.
3. return *obstruction-free-index*.

Thus the obstruction-free-index reflects how easily an opponent could intercept the pass or the subsequent shot. The closer the opponent is to the line and the farther it is from the ball, the better chance it has of intercepting the ball.

The Goalkeeper

The goalkeeper robot has both special hardware and special software. Thus, it does not switch positions or active modes like the others. The goalkeeper's physical frame is distinct from that of the other robots in that it is as long as allowed under the RoboCup-97 rules (18cm) so as to block as much of the goal as possible. The goalkeeper's role is to prevent the ball from entering the goal. It stays parallel to and close to the goal, aiming always to be directly even with the ball's lateral coordinate on the field.

Ideally, simply staying even with the ball would guarantee that the ball would never get past the goalkeeper. However, since the robots cannot accelerate as fast as the ball can, it would be possible to defeat such a behavior. Therefore, the goalkeeper continually monitors the ball's trajectory. In some cases it moves to

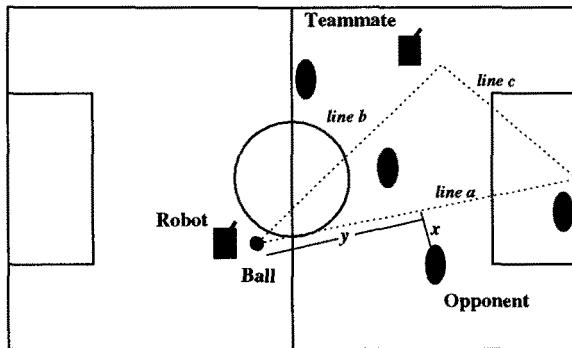


Fig. 7. Run-time pass evaluation is based on position of opponents.

the ball's predicted destination point ahead of time. The decision of when to move to the predicted ball position is both crucial and difficult, as illustrated in Figure 8. Our goalkeeper robot currently take into account the predicted velocity and direction of the ball to select its moves.

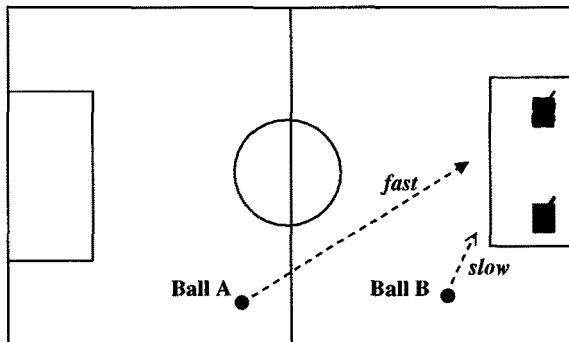


Fig. 8. Goalkeeping.

5 Discussion and Conclusion

CMUnited-97 successfully demonstrated the feasibility and effectiveness of teams of multiagent robotic systems. Within this paradigm, one of the major challenges was to "close the loop," i.e., to integrate all the different modules, ranging from perception to strategic multiagent reasoning. CMUnited is an example of a fully implemented multiagent system in which the loop is closed. In addition, we implemented interesting strategic behaviors, including agent collaboration and real-time evaluation of alternative actions.

It is generally very difficult to accumulate significant scientific results to test teams of robots. Realistically, extended runs are prohibited by battery limitations and the difficulty of keeping many robots operational concurrently. Furthermore, we only had the resources to build a single team of five robots, with one spare so far. Therefore, we offer a restricted evaluation of CMUnited based on the results of four effective 10-minute games that were played at RoboCup-97. We also include anecdotal evidence of the multiagent capabilities of the CMUnited-97 robotic soccer team. Table 1 shows the results of the games at RoboCup-97.

Opponent	Score
NAIST, Institute of Science and Technology, Japan	5-0
MICROB, University of Paris VI , France	3-1
University of Girona, Catalonia, Spain	2-0
NAIST, Japan (finals)	3-0
TOTAL	13-1

Table 1. The scores of CMUnited's games in the small robot league of RoboCup-97. CMUnited-97 won all four games.

In total, CMUnited-97 scored thirteen goals, allowing only one against. The one goal against was scored by the CMUnited goalkeeper against itself, though under an attacking situation from France. We refined the goalkeeper's goal behavior, as presented in this paper, following the observation of our goalkeeper's error.

As the matches proceeded, spectators noticed many of the team behaviors described in the paper. The robots switched positions during the games, and there were several successful passes. The most impressive goal of the tournament was the result of a 4-way passing play: one robot 1 passed to a second robot 2, which passed back to robot 1; then robot 1 passed to a third robot 3, which shot the ball into the goal.

In general, the robots' behaviors were visually appealing and entertaining to the spectators. Several people attained a first-hand appreciation for the difficulty of the task as we let them try controlling a single robot with a joystick program that we developed. All of these people (several children and a few adults) found it quite difficult to maneuver a single robot well enough to direct a ball into an open goal. These people in particular were impressed with the facility with which the robots were able to autonomously pass, score, and defend.

We are aware that many issues are clearly open for further research and development. We are currently systematically identifying them and addressing them towards our next team version. In particular, we are planning on enhancing the robot's behaviors by using machine learning techniques. We are currently developing techniques to accumulate and analyze real robot data.

Acknowledgements

This research is sponsored in part by the Defense Advanced Research Projects Agency (DARPA), and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-95-1-0018 and in part by the Department of the Navy, Office of Naval Research under contract number N00014-95-1-0591. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the Air Force, of the Department of the Navy, Office of Naval Research or the United States Government.

References

1. Minoru Asada, Yasuo Kuniyoshi, Alexis Drogoul, Hajime Asama, Maja Mataric, Dominique Duhamel, Peter Stone, and Hiroaki Kitano. The RoboCup physical agent challenge: Phase-i. *To appear in Applied Artificial Intelligence Journal*, 1998.
2. Minoru Asada, Shoichi Noda, Sukoya Tawaratumida, and Koh Hosoda. Purposeful behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.
3. V. Braatenburg. *Vehicles – experiments in synthetic psychology*. MIT Press, 1984.
4. Kikuo Fujimura. *Motion Planning in Dynamic Environments*. Springer-Verlag, 1991.
5. Kwun Han and Manuela Veloso. Physical model based multi-objects tracking and prediction in robosoccer. In *Working Note of the AAAI 1997 Fall Symposium*. AAAI, MIT Press, 1997.
6. Hiroaki Kitano, Yasuo Kuniyoshi, Itsuki Noda, Minoru Asada, Hitoshi Matsubara, and Eiichi Osawa. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.
7. Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The RoboCup synthetic agent challenge 97. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, San Francisco, CA, 1997. Morgan Kaufman.
8. Jean-Claude Latombe. *Robot motion planning*. Kluwer Academic Publishers, 1991.
9. A. K. Mackworth. On seeing robots. In A. Basu and X. Li, editors, *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, Singapore, 1993.
10. Itsuki Noda. Soccer server : a simulator of RoboCup. In *Proceedings of AI symposium '95*, pages 29–34. Japanese Society for AI, December 1995.
11. Michael K. Sahota. Real-time intelligent behaviour in dynamic environments: Soccer-playing robots. Master's thesis, University of British Columbia, 1993.
12. Michael K. Sahota, Alan K. Mackworth, Rod A. Barman, and Stewart J. Kingdon. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3690–3663, 1995.
13. Randy Sargent, Bill Bailey, Carl Witty, and Anne Wright. Dynamic object capture using fast vision tracking. *AI Magazine*, 18(1):65–72, Spring 1997.
14. Peter Stone and Manuela Veloso. The CMUnited'97 simulator team. In H. Kitano, editor, *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Springer Verlag, Berlin, 1998, in this volume.

Development of Self-Learning Vision-Based Mobile Robots for Acquiring Soccer Robots Behaviors

Takayuki Nakamura

Nara Inst. of Science and Technology Dept. of Information Systems
8916-5, Takayama-cho, Ikoma, Nara 630-01, Japan
takayuki@is.aist-nara.ac.jp

Abstract. An input generalization problem is one of the most important ones in applying reinforcement learning to real robot tasks. To cope with this problem, we propose a self-partitioning state space algorithm which can make non-uniform quantization of the multidimensional continuous state space. This method recursively splits its continuous state space into some coarse spaces called tentative states. It begins by supposing that such tentative states are regarded as the states for Q-learning. It collects Q values and statistical evidence regarding immediate rewards r and Q values within this tentative state space. When it finds out that a tentative state is relevant by the statistical test on minimum description length criterion, it partitions this coarse space into finer spaces. These procedures can make non-uniform quantization of the state space. Our method can be applied to non-deterministic domain because Q-learning is used to find out the optimal policy for accomplishing the given task. To show that our algorithm has generalization capability, we apply our method to two tasks in which a soccer robot shoots a ball into a goal and prevent a ball from entering a goal. To show the validity of this method, the experimental results for computer simulation and a real robot are shown.

Key Words: Self-organizing algorithm, Reinforcement learning, Vision-based mobile robots, Soccer robots.

1 Introduction

RoboCup (The World Cup Robot Soccer) gives a number of research issues for AI and robotics researchers. Such issues involve (1) learning scheme by agents, (2) real-time planning, (3) real-time image processing, (4) coordination or co-operation between agents and so on [1, 2]. Among these technical issues, we currently focus on self-learning scheme by individual agents.

Recently, many researchers in robotics [3] have paid much attention to reinforcement learning methods by which adaptive, reflexive and purposive behavior of robots can be acquired without modeling its environment and its kinematic parameters. A problem in applying reinforcement learning methods to real robot tasks which have continuous state space is that the value function ¹ must be

¹ *value function* is a prediction of the return available from each state and is important because the robot can use it to decide a next action. See section 2.1 for more details.

described in a domain consisting of real-valued variables, which means that it should be able to represent the value in terms of infinitely many state and action pairs. For this reason, function approximators are used to represent the value function when a closed-form solution of the optimal policy is not available.

One approach that has been used to represent the value function is to quantize the state and action spaces into a finite number of cells and collect reward and punishment in terms of all states and actions. This is one of the simplest forms of generalization in which all the states and actions within a cell have the same value. In this way, the value function is approximated as a table in which each cell has a specific value (e.g., [3]). However, there is a compromise between the efficiency and accuracy of this table that is difficult to resolve at design time. In order to achieve accuracy, the cell size should be small to provide enough resolution to approximate the value function. But as the cell size gets smaller, the number of cells required to cover the entire state and action spaces grows exponentially, which causes the efficiency of the learning algorithm to become worse because more data is required to estimate the value for all cells. Chapman et. al [4] proposed an input generalization method which splits an input vector consisting of a bit sequence of the states based on the already structured actions such as “shoot a ghost” and “avoid an obstacle.” However, the original states have been already abstracted, and therefore it seems difficult to be applied to the continuous raw sensor space of real world. Moore et. al [5] proposed a method to resolve the problem of learning to achieve given tasks in deterministic high-dimensional continuous spaces. It divides the continuous state space into cells such that in each cell the actions available may be aiming at the neighboring cells. This aiming is accomplished by a local controller, which must be provided as a prior knowledge of the given task in advance. The graph of cell transitions is solved for shortest paths in an online incremental manner, but a minimax criterion is used to detect when a group of cells is too coarse to prevent movement between obstacles or to avoid limit cycles. The offending cells are split to higher resolution. Eventually, the environment is divided up just enough to choose appropriate actions for achieving the goal. However, the restriction of this method to deterministic environments might limit its applicability since the real environment is often non-deterministic.

Another approach for representing the value function is to use other types of function approximators, such as neural networks (e.g., [6]), statistical models [7, 8, 9, 10] and so on. The approach consists of associating one function approximator to represent the value of all the states and one specific action. Many researchers have experimented with this approach. For examples, Boyan and Moore [11] used local memory-based methods in conjunction with value iteration; Lin [6] used back-propagation networks for Q-learning; Watkins [12] used CMAC for Q-learning; Tesauro [13] used back-propagation for learning the value function in backgammon. Asada et al. [9] used a concentration ellipsoid as a model of cluster (state) of input vectors, inside which a uniform distribution is assumed. They define a state as a cluster of input vectors 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. However, actual distributions are not always uniform. Ideally, situations that input vectors to be included in their model are not included and vice versa should be avoided.

This paper proposes a new method for incrementally dividing a multidimensional continuous state space into some discrete states. This method recursively splits its continuous state space into some coarse spaces called tentative states. It begins by supposing that such tentative states are regarded as the states for Q-learning. It collects Q values and statistical evidence regarding immediate rewards r and Q values within this tentative state space. When it finds out that a

tentative state is relevant by the statistical test on minimum description length (hereafter, MDL) criterion [14], it partitions this coarse space into finer spaces. These procedures can make non-uniform quantization of the state space. Our method can be applied to non-deterministic domain because Q-learning is used to find out the optimal policy for accomplishing the given task.

The remainder of this article is structured as follows: In the next section, We outline the generalization techniques for reinforcement learning algorithm and give our motivation to the approach described in this paper. In section 3.2, we describe our method to automatically construct the sensor spaces. In section 5, we show the results of the experiments with a simple computer simulation and real robot in which a vision-based mobile robot tries to shoot a ball into a goal and tries to prevent a ball from entering a goal. Finally, we give discussion and concluding remarks.

2 Generalization Techniques for Reinforcement Learning Algorithm

2.1 Basics of Reinforcement Learning Algorithm

One step Q-learning [12] has attracted much attention as an implementation of reinforcement learning because it is derived from dynamic programming [15]. Here, we briefly review the basics of Q-learning [16].

In Q-learning algorithm, it is assumed that the robot can discriminate the set S of distinct world states, and can take one from the set A of actions on the world. The world is modeled as a Markovian process, making stochastic transitions based on its current state and the action taken by the robot. Let $T(s, a, s')$ be the probability that the world will transit to the next state s' from the current state-action pair (s, a) . For each state-action pair (s, a) , the *reward* $r(s, a)$ is defined. The general reinforcement learning problem is typically stated as finding a policy that maximizes discounted sum of the reward received over time. A policy f is mapping from S to A . The value function $V^f(s_t)$ associated with a given policy f is defined as:

$$V^f(s_t) \equiv E \left[\sum_{n=0}^{\infty} \gamma^n r_{t+n} \right],$$

where s_t is the state of the system at step t and r_t is the reward received at step t given that the agent started in state s_t and executed policy f . γ is the discounting factor, it controls to what degree rewards in the distant future affect the total value of a policy and is just slightly less than 1. The value function measures the expected discounted sum of rewards or expected rewards the robot will receive when it starts from the given state and follows the given policy.

Given definitions of the transition probabilities and the reward distribution, we can solve the optimal policy, using methods from dynamic programming [15]. A more interesting case occurs when we wish to simultaneously learn the dynamics of the world and construct the policy. Watkin's Q-learning algorithm gives us an elegant method for doing this. Let $Q^*(s, a)$ be the expected return or *action-value function* for taking action a in a situation s and continuing thereafter with the optimal policy. It can be recursively defined as:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a').$$

Because we do not know T and r initially, we construct incremental estimates of the Q values on line. Starting with $Q(s, a)$ at any value (usually 0), every time an action is taken, update the Q value as follows:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r(s, a) + \gamma \max_{a' \in A} Q(s', a') \right],$$

where α is a leaning rate (between 0 and 1) and γ is the discounting factor which controls to what degree rewards in the distant future affect the total value of a policy (between 0 and 1).

2.2 Previous Generalization Techniques and Motivation

As shown in the previous section, basic reinforcement learning algorithm assumed that it is possible to enumerate the state and action spaces and store tables of values over them. In a large smooth state space, we generally expect similar states to have similar values and similar optimal actions. Surely, therefore, there should be some more compact representation than a table. Most problems will have continuous or large discrete state spaces; some will have large or continuous action spaces. The problem of learning in large spaces is addressed through generalization techniques, which allow compact storage of learned information and transfer of knowledge between "similar" states and actions.

One method to allow reinforcement-learning techniques to be applied in large state spaces is to use a function approximator so as to represent the value function by mapping a state description to a value. The following explanation originated from [17]. We follow this literature in order to explain the necessity of non-uniform resolution model for representing a function approximator.

Although there have been some positive examples, in general there are unfortunate interactions between function approximation and the learning rules. In discrete environments there is a guarantee that any operation that updates the value function (according to the Bellman equations) can only reduce the error between the current value function and the optimal value function. This guarantee no longer holds when generalization is used. These issues are discussed by Boyan and Moore [11], who give some simple examples of value function errors growing arbitrarily large when generalization is used with value iteration. Their solution to this, applicable only to certain classes of problems, discourages such divergence by only permitting updates whose estimated values can be shown to be near-optimal via a battery of Monte-Carlo experiments. Several recent results [18, 19] show how the appropriate choice of function approximator can guarantee convergence, though not necessarily to the optimal values. Baird's residual gradient technique [20] provides guaranteed convergence to locally optimal solutions.

Perhaps the gloominess of these counter-examples is misplaced. Boyan and Moore [11] report that their counter-examples can be made to work with problem-specific hand-tuning despite the unreliability of untuned algorithms that provably converge in discrete domains. Sutton [21] shows how modified versions of Boyan and Moore's examples can converge successfully. An open question is whether general principles, ideally supported by theory, can help us understand when value function approximation will succeed. In Sutton's comparative experiments with Boyan and Moore's counter-examples, he changes some aspects of the experiments. Boyan and Moore sampled states uniformly in state space, whereas Sutton's method sampled along empirical trajectories. This change must cause different results. Therefore, more careful research associated with this point is needed.

3 Self-Partitioning State Space Algorithm

3.1 Function Approximator with Non-uniform Resolution Model

There are some reasons why designing non-uniform function approximators may be more beneficial than designing uniform ones.

- In case that the designers know, up to a certain degree, prior knowledge of the system (for example, what regions of the state-action space will be used more often.), it may be efficient to design the function approximator such that it may use many resources in more heavily transited regions than in regions of the state space that are known to be visited rarely.
- If the amount of resources is limited, a non-uniform function approximator may make better performance and learning efficiency than that achieved with a uniform function approximator just because the former is able to exploit the resources more efficiently than the later.
- It may be possible to design function approximators that dynamically allocate more resources in certain regions of the state-action space and increase the resolution in such regions as required to perform on-line.

3.2 Details of Our Algorithm

In this work, we define the sensor inputs, actions and rewards as follows:

- Sensor input \mathbf{d} is described by a N dimensional vector $\mathbf{d} = (d_1, d_2, \dots, d_N)$, each component $d_i (i = 1 \sim N)$ of which represents the measurement provided by the sensor i . The continuous value d_i is provided by the sensor i . Its range $\text{Range}(d_i)$ is known in advance. Based on $\text{Range}(d_i)$, a measurement d_i is normalized in such a way that d_i can take values in the semi open interval $[0, 1]$.
- The agent has a set \mathbf{A} of possible actions $a_j, j = 1 \sim M$. Such a set is called the action space.
- One of the discrete rewards $r = r_k, k = 1 \sim C$ is given to the agent depending on the evaluation of the action taken at a state.

Our algorithm works as follows:

1. It starts by assuming that the entire environment is as if it were one state. Initially, the total number of the states $|S|$ is one.
2. We utilize a segment tree to classify N dimensional input vector. The inner node at i th depth in the j th level keeps the range $b_i(j) = [t_i^{low}, t_i^{high}]$ of a measurement provided by each sensor i . (Actually, j corresponds to the number of iteration of this algorithm.) At each inner node in the j the level, the range of a measurement is partitioned into two equal intervals $b_i^0(j) = [t_i^{low}, (t_i^{low} + t_i^{high})/2]$ and $b_i^1(j) = [(t_i^{low} + t_i^{high})/2, t_i^{high}]$. For example, initially $j = 0$, the range of each dimension i is divided into two equal intervals $b_i^0(0) = [0.0, 0.5]$ and $b_i^1(0) = [0.5, 1.0]$. When sensor input vector \mathbf{d} has N dimensions, a segment tree whose depth is N is built (see Fig.1). The leaf node corresponds to the result of classification for observed sensor input vector \mathbf{d} . As a result, 2^N leaf nodes are generated. These leaf nodes can represent the situations in the agent's environment. The state space represented by the leaf nodes is called "tentative state space" TS . Let $ts_k, k = 1 \sim 2^N$ be the component of the tentative state space which is called "tentative state."

3. Based on this tentative state space TS , our algorithm begins Q-learning. In parallel with this process, it gathers statistics in terms of $r(a_i|ts_k = on)$, $r(a_i|ts_k = off)$, $Q(a_i|ts_k = on)$ and $Q(a_i|ts_k = off)$, which indicate immediate rewards r and discounted future rewards Q in case that individual state is "on" or "off," respectively. In this work, it is supposed that if a N dimensional sensor vector d is classified into a leaf node ts_k , the condition of this node ts_k is regarded as "on," otherwise (this means the case that d is classified into the leaf node except ts_k), it is regarded as "off."
4. After Q-learning based on the tentative state space is converging, our algorithm asks the question whether there are some states in the state description such that the r and Q values for states "on" are significantly different from such values for states "off." When the distributions of statistics of ts_k in case of "on" and "off" are different, it is determined that ts_k is *relevant* to the given task. In order to discover the difference between two distributions, our algorithm performs the statistical test based on MDL criterion. In the section 4.1 and 4.2, these procedures are explained.
5. (a) If there is the state ts'_k adjoining the state ts_k which is shown to be relevant such that the statistical characteristic of Q values and actions assigned at the adjoining state are same, merge these two states into one state.
 (b) Otherwise, skip this step.
6. Each leaf nodes ts_k is represented by a combination of intervals each of which corresponds to the range of a measurement provided by each sensor i . These intervals in ts_k which is shown to be relevant are bisected. As a result, in terms of one ts_k , 2^N leaf nodes are generated and correspond to tentative states.
7. Supposing that these tentative states are regarded as the states in Q-learning, our algorithm performs Q-learning again.
8. Until our algorithm can't find out the relevant leaf nodes, the procedures 2 ~ 7 are repeated. Finally, a hierarchical segment tree is constructed to represent the partitioning of the state space for achievement of a given task.

After the learning, based on Q values stored at leaf nodes, the agent takes actions for accomplishing the given task.

4 The Relevance Test Based on MDL Criterion

Here, we explain how to determine whether a state is *relevant* to the task or not. **Fig. 2** shows the difference between the distributions of r or Q values regarding to the state ts_k in case that ts_k is *relevant* or *irrelevant*. As shown in upper part of this figure, when two peaks of the distributions of r or Q values, which correspond to pair of $r(a_i|ts_k = on)$ and $r(a_i|ts_k = off)$, or pair of $Q(a_i|ts_k = on)$ and $Q(a_i|ts_k = off)$, can be clearly discriminated, it is supposed that ts_k is relevant to the given task because the such state affects the value of state more heavily than the other states does, therefore, it affects how the robot should act at next time step. On the contrary, in case that two peaks are ambiguous as shown in bottom part of this figure, it is considered that ts_k is *irrelevant* to the given task. Actually, we perform the statistical test with respect r and Q values based on MDL criterion [14] in order to distinguish the distribution of such reinforcement values.

In case of $n = 3$

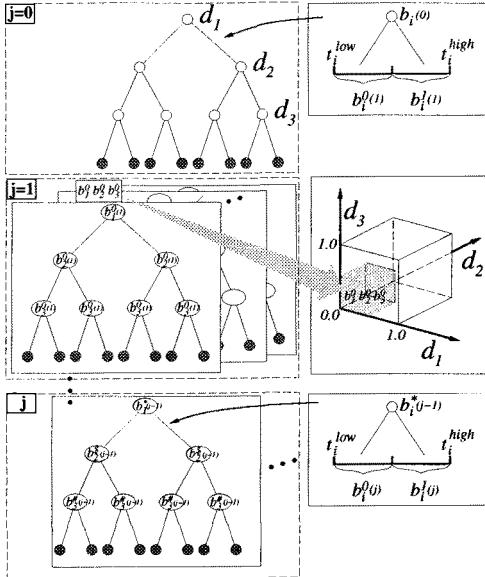


Fig. 1. Representation of state space by a hierarchical segment tree

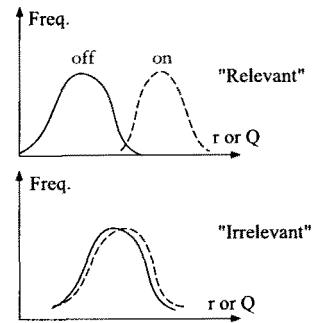


Fig. 2. Criterion for determining the relevance of the state

4.1 The statistical test for r value

Since the immediate reward r_j is given at each trial among one of C mutually exclusive rewards r_j , $j = 1, 2, \dots, C$, the distribution of r_j in the state ts_k follows a multi-nominal distribution. Let n be the number of independent trials, k_i be the number of event E_i and p_i be the probability that the event E_i occurs where $\sum_{i=1}^C p_i = 1$. The probability that E_1 occurs k_1 times, E_2 occurs k_2 times, ... E_C occurs k_C times, can be represented by the multi-nominal distribution as follows:

$$p(k_1, \dots, k_c | p_1, \dots, p_c) = \frac{n!}{k_1! \cdots k_c!} p_1^{k_1} \cdots p_c^{k_c}$$

where, $0 \leq k_i \leq n$ ($i = 1 \sim c$), $\sum_{i=1}^c = n$.

Supposing Table 1 shows the distribution of immediate rewards in case that the state ts_k is "on" or "off," our algorithm tries to find the difference between the distributions of rewards in two cases of ts_k "on" and "off" based on this table.

Table 1. The distribution of rewards in ts_k

Rewards	On	Off
R_1	$n(On, R_1)$	$n(Off, R_1)$
R_2	$n(On, R_2)$	$n(Off, R_2)$
:	:	:
R_C	$n(On, R_C)$	$n(Off, R_C)$
	$n(On)$	$n(Off)$

$n(i)$: the frequency of sample data in the state i ($i = 1, \dots, S$)
 $n(i, r_j)$: the frequency of reward r_j ($j = 1, \dots, C$) given in the state i
 $p(r_j|i)$: the probability that reward r_j is given in the state i
 $\sum_{j=1}^C n(i, r_j) = n(i)$, $\sum_{j=1}^C p(r_j|i) = 1$,
 $(i = 1, \dots, S)$.

The probability $P(\{n(i, r_j)\} | \{p(r_j|i)\})$ that the distribution of the immediate rewards are acquired as shown in Tab. 1 $\{n(i, r_j)\}$, ($i = 1 \dots, S$, $j = 1, \dots, C$), can be described as follows:

$$P(\{n(i, r_j)\} | \{p(r_j|i)\}) = \prod_{i=1}^S \left\{ \frac{n(i)!}{\prod_{j=1}^C n(i, r_j)!} \prod_{j=1}^C p(r_j|i)^{n(i, r_j)} \right\}$$

The likelihood function L of this multi-nominal distribution can be written as follows:

$$L(\{p(r_j|i)\}) = \sum_{i=1}^S \sum_{j=1}^C n(i, r_j) \log p(r_j|i) + K, \text{ where } K = \log \left\{ \frac{\prod_{i=1}^S n(i)!}{\prod_{i=1}^S \prod_{j=1}^C n(i, r_j)!} \right\}.$$

When two multi-nominal distributions in case that each tentative state ts_k is "on" and "off" can be considered to be same, the probability $p(r_j|i)$ that the immediate reward r_j is given in the state i can be modeled as follows:

$$M1: p(r_j|i) = \theta(r_j) \quad i = 1, \dots, S, \quad j = 1, \dots, C.$$

Furthermore, its likelihood function L_1 can be written by

$$L_1(\{\theta(r_j)\}) = K + \sum_{j=1}^C \left[\left\{ \sum_{i=1}^S n(i, r_j) \right\} \log \theta(r_j) \right].$$

Therefore, the maximum likelihood ML_1 can be written by

$$ML_1 = L(\hat{\theta}(r_j)) = L \left(\frac{\sum_{i=1}^S n(i, r_j)}{\sum_{i=1}^S n(i)} \right).$$

where $\hat{\theta}(r_j)$ is the maximum likelihood estimation of $\theta(r_j)$. Therefore, the description length $l_{MDL}(M1)$ of the model $M1$ based on MDL principle is

$$l_{MDL}(M1) = -ML_1 + \frac{C-1}{2} \log \sum_{i=1}^S n(i).$$

On the contrary, when two multi-nominal distributions in case of "on" and "off" can be considered to be different, the probability $p(r_j|i)$ can be modeled as follows:

$$\text{M2: } p(r_j|i) = \theta(r_j|i) \quad i = 1, \dots, S, \quad j = 1, \dots, C.$$

Furthermore, its likelihood function L_2 can be written by

$$L_2(\{\theta(r_j|i)\}) = K + \sum_{i=1}^S \sum_{j=1}^C n(i, r_j) \log \theta(r_j|i).$$

In the same way, the maximum likelihood ML_2 can be written by

$$ML_2 = L(\hat{\theta}(r_j)) = L\left(\frac{n(i, r_j)}{n(i)}\right).$$

Therefore, the description length $l_{MDL}(M2)$ of the model $M2$ based on MDL principle is

$$l_{MDL}(M2) = -ML_2 + \frac{S(C-1)}{2} \log \sum_{i=1}^S n(i).$$

Based on MDL criterion, we can suppose that discovering the difference between two distributions is equivalent to determining which model is appropriate for representing the distribution of data. In this work, the difference between the distributions is found based on the following conditions:

- if $l_{MDL}(M1) > l_{MDL}(M2)$
Two distributions are different.
- if $l_{MDL}(M2) \geq l_{MDL}(M1)$
Two distributions are same.

4.2 The statistical test for Q value

In order to distinguish the distribution of sampled data of Q values, we perform the statistical test based on MDL criterion. Let \mathbf{x}^n and \mathbf{y}^m be the sample data (x_1, x_2, \dots, x_n) and (y_1, y_2, \dots, y_m) , respectively. \mathbf{x}^n and \mathbf{y}^m indicate a history of $Q(a_i|ts_k = \text{on})$ and $Q(a_i|ts_k = \text{off})$, respectively. We'd like to know whether these two sample data \mathbf{x}^n and \mathbf{y}^m come from the two different distributions or the same distribution. Here, we assume the following two model for the distribution of sampled data are $M1$ based on one normal distribution and $M2$ based on two normal distributions.

$$\text{M1: } N(\boldsymbol{\mu}, \sigma^2), \quad \text{M2: } N(\boldsymbol{\mu}_1, \sigma'^2), N(\boldsymbol{\mu}_2, \sigma'^2)$$

The normal distribution with mean μ and variance σ is defined by

$$f(x : \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}.$$

If both \mathbf{x}^n and \mathbf{y}^m follow the model $M1$, the probabilistic density function of these sampled data can be written by

$$\prod_{i=1}^n f(x_i : \mu, \sigma^2) + \prod_{i=1}^m f(y_i : \mu, \sigma^2).$$

The likelihood function LL_1 of the model $M1$ can be written by

$$LL_1 = \sum_{i=1}^n \log f(x_i : \mu, \sigma^2) + \sum_{i=1}^m \log f(y_i : \mu, \sigma^2).$$

Therefore, the maximum likelihood MLL_1 can be derived as follows:

$$MLL_1 = -\frac{n+m}{2}(1 + \log 2\pi\hat{\sigma}^2),$$

where $\hat{\sigma}^2$ is the maximum likelihood estimated variance of σ^2 and can be estimated as follows:

$$\hat{\mu} = \frac{1}{n+m} \left\{ \sum_{i=1}^n x_i + \sum_{i=1}^m y_i \right\}, \quad \hat{\sigma}^2 = \frac{1}{n+m} \left\{ \sum_{i=1}^n (x_i - \hat{\mu})^2 + \sum_{i=1}^m (y_i - \hat{\mu})^2 \right\},$$

where $\hat{\mu}$ is the maximum likelihood estimated mean of μ . Therefore, the description length $l_{MDL}(M1)$ of the model $M1$ based on MDL principle is

$$l_{MDL}(M1) = -MLL_1 + \log(n+m).$$

On the contrary, if x^n and y^m follow the model $M2$, the probabilistic density function of these sampled data can be written by

$$\prod_{i=1}^n f(x_i : \mu_1, \sigma'^2) + \prod_{i=1}^m f(y_i : \mu_2, \sigma'^2).$$

The likelihood function LL_2 of the model $M2$ can be written by

$$LL_2 = \sum_{i=1}^n \log f(x_i : \mu_1, \sigma'^2) + \sum_{i=1}^m \log f(y_i : \mu_2, \sigma'^2).$$

Therefore, the maximum likelihood MLL_2 can be derived as follows:

$$MLL_2 = -\frac{n+m}{2}(1 + \log 2\pi\hat{\sigma}'^2),$$

where $\hat{\sigma}_1^2$ and $\hat{\sigma}_2^2$ is the maximum likelihood estimated variance of σ_1^2 and σ_2^2 , respectively. These statistics can be estimated as follows:

$$\begin{aligned} \hat{\mu}_1 &= \frac{1}{n} \sum_{i=1}^n x_i, & \hat{\mu}_2 &= \frac{1}{m} \sum_{i=1}^m y_i, \\ \hat{\sigma}'^2 &= \frac{1}{n+m} \left\{ \sum_{i=1}^n (x_i - \hat{\mu}_1)^2 + \sum_{i=1}^m (y_i - \hat{\mu}_2)^2 \right\}, \end{aligned}$$

where $\hat{\mu}_1$ and $\hat{\mu}_2$ is the maximum likelihood estimated mean of μ_1 and μ_2 , respectively. Therefore, the description length $l_{MDL}(M2)$ of the model $M2$ based on MDL principle is

$$l_{MDL}(M2) = -MLL_2 + \frac{3}{2} \log(n+m).$$

We can recognize the difference between the distributions based on the following condition:

```

if    $l_{MDL}(M1) > l_{MDL}(M2)$ 
 $x$  and  $y$  arise from the different normal distributions
if    $l_{MDL}(M2) \geq l_{MDL}(M1)$ 
 $x$  and  $y$  arise from the same normal distribution

```

5 Experimental Results

To show that our algorithm has a generalization capability, we apply it to acquire two different behaviors: one is a shooting behavior and the other is a defending behavior for soccer robots.

5.1 Simulation

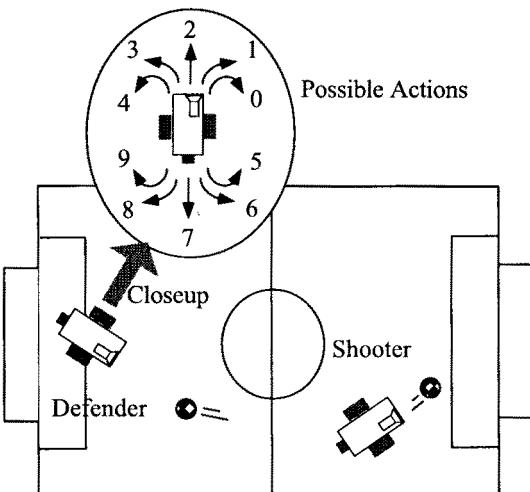


Fig. 3. Simulation environment

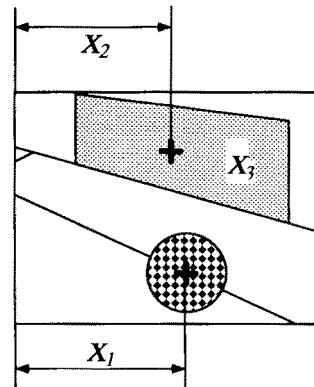


Fig. 4. Input vector as sensor information

We consider an environment shown in **Fig. 3** where the task for a mobile robot is to shoot a ball into a goal or to defend a shot ball. The environment consists of a ball and a goal, and the mobile robot has a single CCD camera. The robot does not know the location and the size of the goal, the size and the weight of the ball, any camera parameters such as focal length and tilt angle, or kinematics/dynamics of itself.

We performed the computer simulation with the following specifications. The field is $1.52m \times 2.74m$. The goal posts are located at the center of the left and right line of the rectangle (see **Fig. 3**) and its height and width are $0.20m$ and $0.5m$, respectively. The robot is $0.10m$ wide and $0.20m$ long and kicks a ball of diameter $0.05m$. The maximum translation velocity is $5cm/s$. The camera is

horizontally mounted on the robot (no tilt) and is in off-centered position. Its visual angle is 36 degrees. The velocities of the ball before and after being kicked by the robot is calculated by assuming that the mass of the ball is negligible compared to that of the robot. The speed of the ball is temporally decreased by a factor 0.8 in order to reflect the so-called "viscous friction." The values of these parameters are determined so that they can roughly simulate the real world.

The robot is driven by two independent motors and steered by front and rear wheels which is driven by one motor. Since we can send the motor control commands such as "move forward or backward in the given direction," all together, we have 10 actions in the action primitive set A as shown in Fig. 3. The robot continues to take one action primitive at a time until the current state changes. This sequence of the action primitives is called an action. Actually, a stop motion does not causes any changes in the environment, we do not take into account this action primitive.

In computer simulation, we take into account the delay due to sensory information processing. The contents of the image processing are color filtering (a ball and a goal are painted in red and blue, respectively), localizing and counting red and blue pixels, and vector calculation. We are using a general graphic workstation for image processing in the real robot experiments. Since it is not specialized on an image processing, it takes about 66 ms to perform these processes. As a result, in the simulation, the robot is assumed to take an action primitive every 66ms.

The size of the image taken by the camera is 256×240 pixels. An input vector \mathbf{x} to our algorithm consists of:

- x_1 : the horizontal position of the ball in the image, that ranges from 0 to 256 pixels,
- x_2 : the horizontal position of the goal in the image ranging from 0 to 256,
- x_3 : the area of the goal region in the image, that ranges from 0 to 256×240 pixels².

After the range of these values is normalized in such a way that the range may become the semi open interval $[0, 1]$, they are used as inputs of our method.

A discounting factor γ is used to control to what degree rewards in the distant future affect the total value of a policy. In our case, we set the value a slightly less than 1 ($\gamma = 0.9$). In this work, we set the learning rate $\alpha = 0.25$. In case that the shooting behavior tried to be acquired by our method, as a reward value, 1 is given when the robot succeeded in shooting a ball into a goal, 0.3 is given when the robot just kicked a ball, -0.01 is given when the robot went out of field, 0 is given otherwise. In the same way, in case that the defending behavior tried to be acquired by our method, as a reward value, -0.7 is given when the robot failed in preventing a ball from entering a goal, 1.0 is given when the robot just kicked a ball, -0.01 is given when the robot went out of field, 0 is given otherwise.

In the learning process, Q-learning continues until the sum of estimated Q values seems to be almost convergent. When our algorithm tried to acquire the shooting behavior, our algorithms ended after it iterated the process (Q-learning + statistical test) 8 times. In this case, about $160K$ trials were required to converge our algorithm and the total number of the states is 246. In case that our algorithm tried to acquire the defending behavior, our algorithms ended after it iterated the process (Q-learning + statistical test) 5 times. In this case, about $100K$ trials were required to converge our algorithm and the total number of the states is 141.

Fig. 5 shows the success ratio versus the step of trials in the two learning processes that one is for acquiring a shooting behavior the other is for a defending behavior. We define the success rate as $(\# \text{ of successes}) / (\# \text{ of trials}) \times 100(\%)$.

As you can see, the bigger the number of iteration is, the higher the success ratio at the final step in each iteration. This means that our algorithm gradually made better segmentation of state space for accomplishing the task.

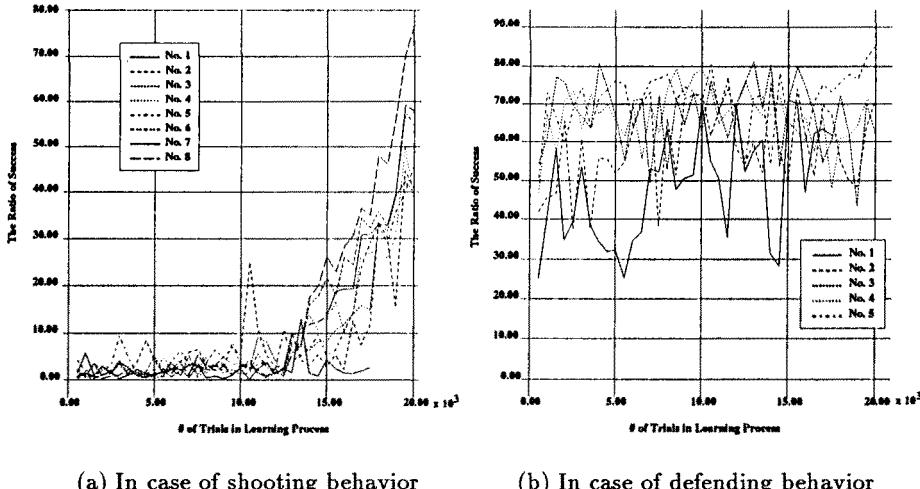


Fig. 5. The success ratio versus the step of trial

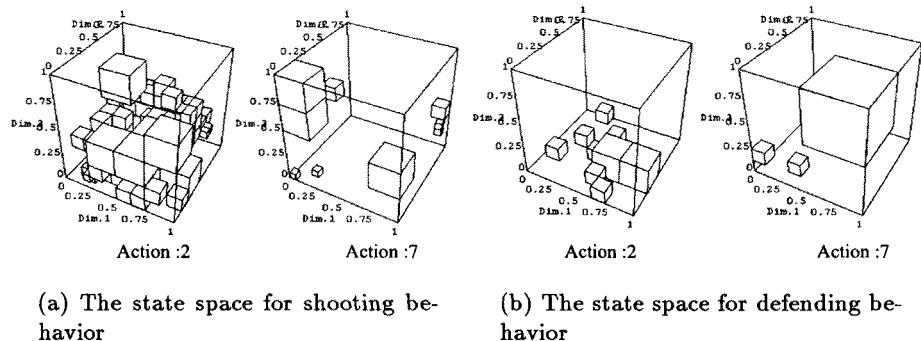


Fig. 6. The partitioned state space

Fig. 6 shows the partitioned state spaces obtained by our method. The upper part of this figure shows the state space for the shooting behavior, the bottom part shows one for defending behavior. In each figure, *Dim.1*, *Dim.2* and *Dim.3* shows the position of ball, the position of goal and the area of goal region,

respectively. Furthermore, in each figure, *Action2* and *Action7* corresponds to “moving forward” and “moving backward,” respectively.

For the sake of readers understanding, one cube in the partitioned state space corresponds to one state. For example, the top left shows a group of the cube where the action 2 is assigned as an optimal action. As shown in this figure, many cubes where forward actions are assigned concentrate around the center of the entire state space. This means that the robot will take an forward action if the ball and goal are observed around the center of field of its view. This shows very natural behavior for shooting a ball into a goal.

In the bottom right figure, there is one large cube. This means that the the robot will take an backward action if large goal are observed around the center of field of its view. This strategy is plausible behavior for preventing a ball from entering a goal because the robot will have to go back in front of own goal after it moved out there in order to kick a ball.

Fig.7 shows a typical sequence of the shooting behavior in the simulation acquired by our method. The bottom part of each figure in the **Fig. 7** shows the processed image where the white rectangle and dark gray circle indicates the goal and ball region, respectively. Once the robot found the ball, the robot ran to the ball quickly. After that, the robot kicked the ball into the goal.

Fig.8 shows the defending behavior in the simulation acquired by our method. The robot initially looked around because the ball was not observed from its initial position. Then, once the robot found the ball, the robot ran to the ball quickly. After that, the robot kicked the ball to prevent a ball from approaching the goal. The robot followed the ball until the area of opponent’s goal grows to a certain extent. Then in order to go back to the front of own goal, the robot took a backward action. It seems that the robot utilizes the area of the opponent’s goal to localize its own position in the field. Note that this behavior is just the result of learning and not hand-coded.

5.2 Real Robot Experiments

Fig. 9 shows our real robot system which we have developed to take part in RoboCup-97 competition where several robotic teams are competing on a field. So, the system includes two robots which have the same structure: one for a shooter, the other for a defender. Off-board computer SGI ONYX (R4400/250MHz) perceives the environment through on-board cameras, performs the decision making based on the learned policy and sends motor commands to each robot. A CCD camera is set at bottom of each robot in off-centered position . Each robot is controlled by SGI ONYX through radio RS232C. The maximum vehicle speed is about 5cm/s. The images taken by the CCD camera on each robot are transmitted to a video signal receiver. In order to process two images (one is sent from the shooter robot, the other from the defender robot) simultaneously, two video signals are combined into one video signal by a video combiner on PC. Then, the video signal is sent to SGI ONYX for image processing. The color-based visual tracking routine is implemented for tracking and finding a ball and a goal in the image. In our current system, it takes 66 ms to do this image processing for one frame.

Simple Color-Based Tracking

Our simple tracking method is based on tracking regions with similar color information from frame to frame. We assume the existence of the color models (CM_{ball} , CM_{goal}) for tracking targets, which are estimated at initialization. Actually, we use only hue values (H_{ball} and H_{goal}) of targets as (CM_{ball} and

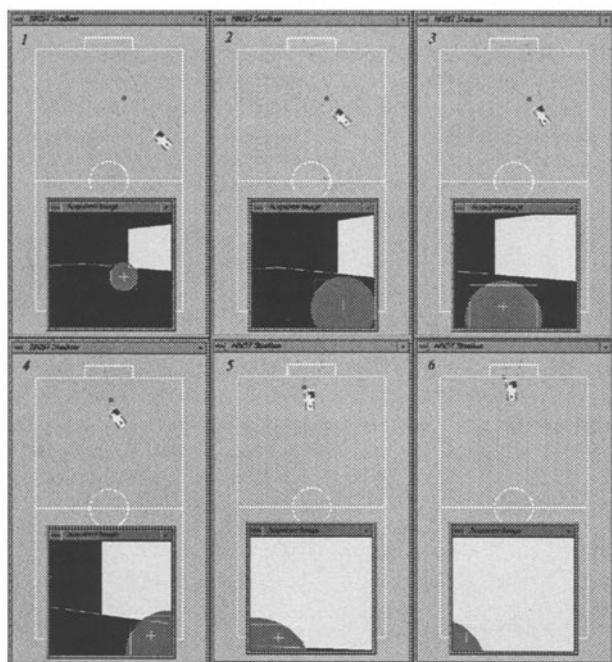


Fig. 7. Shooting behavior in simulation

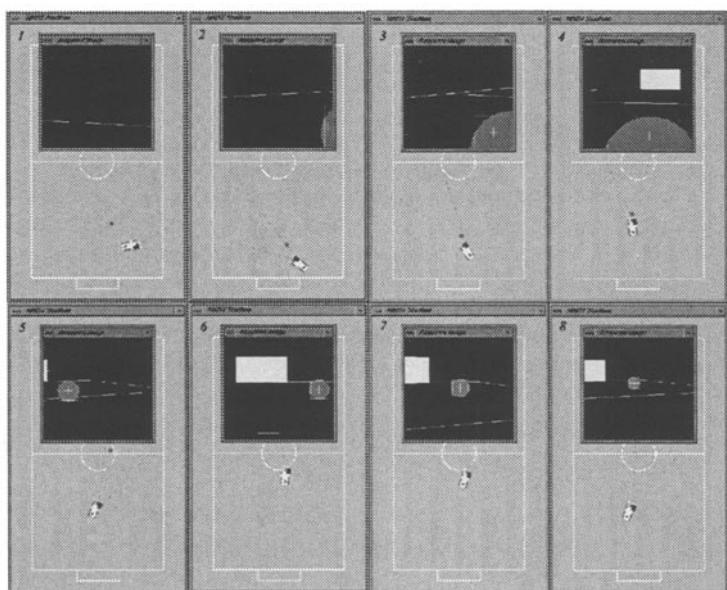


Fig. 8. Defending behavior in simulation

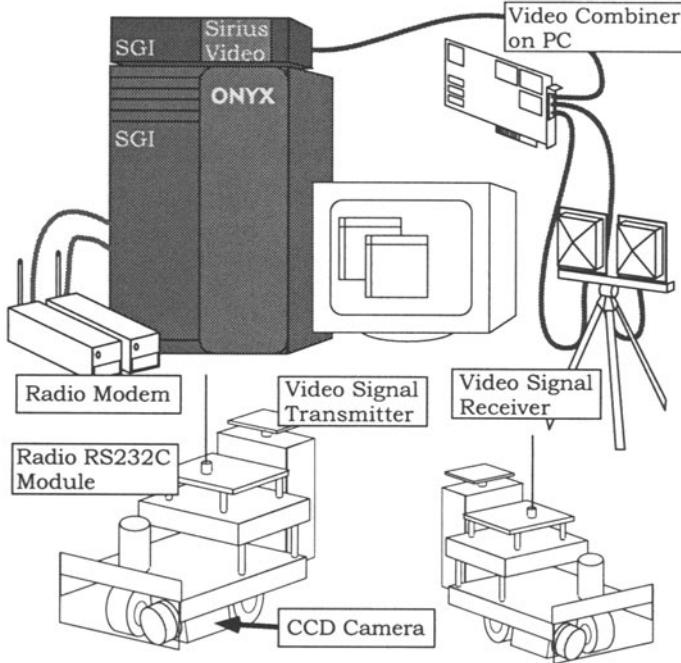


Fig. 9. Our robots and system

CM_{goal}) to reduce its sensitivity due to changes in illumination. We define a fitness function $\Phi_{target}(x, y)$ at a pixel (x, y) as a criterion for extracting a target region in the image,

$$\Phi_{target}(x, y) = \begin{cases} 1 & H_{target} - \sigma_{target} \leq H(x, y) \leq H_{target} + \sigma_{target}, \\ 0 & \text{Otherwise} \end{cases}$$

, where $H(x, y)$ and σ_{target} show hue value at (x, y) and a threshold for extraction, respectively. In our current implementation, we set $\sigma_{ball} = 10$ and $\sigma_{goal} = 20$. These values can be calculated based on the standard deviation of H_{ball} and H_{goal} which were estimated at initial estimation process. Based on $\Phi_{target}(x, y)$, the best estimate $(\hat{x}_{target}, \hat{y}_{target})$ for the target's location is calculated as follows:

$$\hat{x}_{target} = \frac{\sum_{(x_i, y_i) \in R} x_i \Phi_{target}(x_i, y_i)}{\sum_{(x_i, y_i) \in R} \Phi_{target}(x_i, y_i)}, \quad \hat{y}_{target} = \frac{\sum_{(x_i, y_i) \in R} y_i \Phi_{target}(x_i, y_i)}{\sum_{(x_i, y_i) \in R} \Phi_{target}(x_i, y_i)},$$

where R shows the search area. Initially, R implies an entire image plane. After initial estimation for the location of the target, we can know the standard deviations $\sigma(\hat{x}_{target})$ and $\sigma(\hat{y}_{target})$ regarding $(\hat{x}_{target}, \hat{y}_{target})$. Therefore, based on the deviations, R is restricted to a local region during the tracking process as follows:

$$R : \{(x, y) | \hat{x}_{target} - 2.5\sigma(\hat{x}_{target}) \leq x \leq \hat{x}_{target} + 2.5\sigma(\hat{x}_{target}), \\ \hat{y}_{target} - 2.5\sigma(\hat{y}_{target}) \leq y \leq \hat{y}_{target} + 2.5\sigma(\hat{y}_{target})\}.$$

$\sum_{(x_i, y_i) \in R} \Phi_{target}(x_i, y_i)$ shows the area of the target in the image. Based on this value, we judge the appearance of the target. If this value is lower than the pre-defined threshold, the target is considered to be lost, then R is set to be the entire image plane for estimation at next time step. We set this threshold for the target area = $0.05 * S$, where S shows the area of the entire image. **Fig. 10** shows examples of processed images during the task execution. In these figures, a light gray and a dark gray "X" show the estimated location of the ball and the goal, respectively. In the same way, the white rectangles surrounded with dotted line in each image show the estimated area of the ball and the goal.

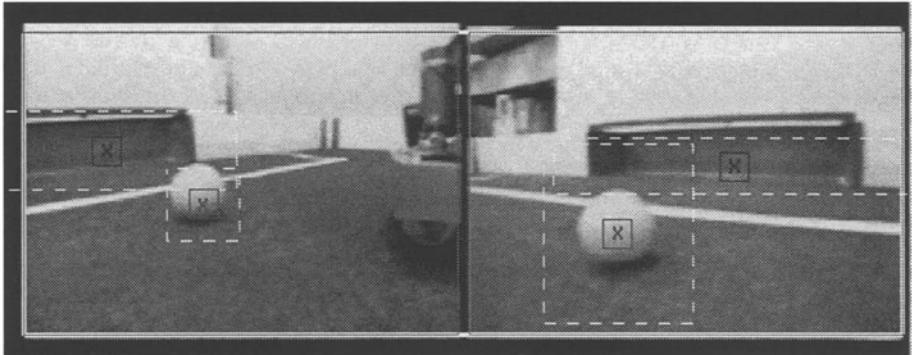


Fig. 10. A example of processed images taken by the robots

Experimental Results

Fig. 11 shows how a real robot shoots a ball into a goal based on the state space obtained by our method. 6 images are shown in raster order from the top left to the bottom right in every 2.0 seconds, in which the robot tried to shoot a ball, but failed, then moved backward so as to find a position to shoot a ball, finally succeeded in shooting. Note that the backward motion for retry is just the result of learning and not hand-coded.

Fig. 12 shows how a real robot prevents a ball shot by the opponent's robot from entering a goal based on the state space obtained by our method. 6 images are shown in raster order from the top left to the bottom right in every 2.0 seconds. This sequence was recorded at the RoboCup-97 competition. In the top left image, the white circle shows the position of the defender robot and the white arrow shows the position of the ball. In the time step 1 and 2, first, the opponent's robot tried to shoot a ball, then the defender robot prevent a ball from entering a goal. In the time step 3 ~ 6, another opponent's robot got a ball cleared by the defender robot and tried to shoot a ball again, then the defender robot defend a goal again. As shown in these figures, the defender robot always moves in front of own goal to find out a shot ball as soon as possible. Note that this behavior is just the result of our learning algorithm and not hand-coded.

6 Concluding Remarks and Discussion

We have proposed a method for self-partitioning the state space which recursively splits continuous state space into some coarse regions based on the relevance test in terms of reward values. Our method utilized a hierarchical segment tree in order to represent the non-uniform partitioning of the state space. This representation has an advantage for approximating the non-uniform distribution of sample data which frequently occurs in real world. We also have shown the validity of the method with computer simulations and real robot experiments at our lab. and the RoboCup-97 competition. We think that segmentation of sensory data from the environment should depend on the purpose (task), capabilities (sensing, acting, and processing) of the robot, and its environment. The state spaces obtained by our method (**Fig.6** indicates such segmentations) correspond to the internal representations of the robot for accomplishing given tasks. Furthermore, we think that the resolution of state spaces indicates the importance that how much attention the robot have to pay to a part of the state space. In these sense, by our method, our robot acquired the the subjective representation for its environment and the given task.

Current version of our algorithm has two kinds of problem:

- If the dimension of the sensor space is higher, the computational time and amount are enormous, and as a result the learning might not converge correctly. The dimension of the sensor space is associated with the number of features which are used to describe all situations in the environment. In our current implementation, we used input vector consisting of three parameters which were enough to acquire the shooting and defending behaviors. However, there is no guarantee that these parameters are enough in case that our method tries to acquire the higher-level behaviors such as strategic coordination between two robots. Generally, selection of features from raw images is really a difficult problem. The selection of feature which is necessary to accomplish a give task might be much harder when such a feature changes depending on situations. Since use of all possible sensory information seems impossible, a learning mechanism for selecting features from the sensory data should be developed. As a result, an effective method for coping with higher dimension space might be developed.
- We currently define the quantization of the action space in advance. Since our robot has only two degrees of freedom (DOFs), abstraction of action space is not very important. However, it is generally important to quantize the action space in case that the dimension of the action space is higher and/or we want to use real-valued motor signals to control the robot precisely. For example, if the robot has an active vision system with pan and tilt mechanism in addition to the two DOFs for driving system, the action space should be abstracted depending on situation in order to coordinate between sensing and driving actions. The abstraction of action space is as necessary as the abstraction of sensor space.

7 Acknowledgments

The main idea of this paper was thought of while I stayed at AI Lab of Comp. Sci. Dept. of Brown University. I would like to thank Prof. L. P. Kaelbling for her helpful comments during my stay. I also would like to thank Prof. M. Imai for providing research fund and S. Morita Japan SGI Cray Corp for lending SGI ONYX to me. Finally, I want to say thank you to my colleague M. Iwase for his help in the development of our robot.

References

1. H. Kitano, M. Tambe, Peter Stone, and et.al. "The robocup synthetic agent challenge 97". In *Proc. of The First International Workshop on RoboCup*, pages 45–50, 1997.
2. M. Asada, Y. Kuniyoshi, A. Drogoul, and et.al. "The robocup physical agent challenge:phase i(draft)". In *Proc. of The First International Workshop on RoboCup*, pages 51–56, 1997.
3. J. H. Connell and S. Mahadevan, editors. *Robot Learning*. Kluwer Academic Publishers, 1993.
4. D. Chapman and L. P. Kaelbling. "Input generalization in delayed reinforcement learning: An algorithm and performance comparisons". In *Proc. of IJCAI-91*, pages 726–731, 1991.
5. A. W. Moore and C. G. Atkeson. "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces". *Machine Learning*, 21:199–233, 1995.
6. Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". *Machine Learning*, 8:293–321, 1992.
7. H. Ishiguro, R. Sato, and T. Ishida. "Robot oriented state space construction". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, 1996.
8. A. Ueno, K. Hori, and S. Nakasuka. "Simultaneous learning of situation classification based on rewards and behavior selection based on the situation". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, 1996.
9. M. Asada, S. Noda, and K. Hosoda. "Action-based sensor space categorization for robot learning". In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, volume 3, 1996.
10. Y. Takahashi, M. Asada, and K. Hosoda. "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 (IROS96)*, volume 3, pages 1518–1524, 1996.
11. J. Boyan and A. Moore. "Generalization in reinforcement learning: Safely approximating the value function". In *Proceedings of Neural Information Processings Systems 7*. Morgan Kaufmann, January 1995.
12. C. J. C. H. Watkins. "Learning from delayed rewards". PhD thesis, King's College, University of Cambridge, May 1989.
13. G. Tesauro. "Practical issues in temporal difference learning". *Machine Learning*, 8:257–277, 1992.
14. J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.
15. R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
16. L. P. Kaelbling. "Learning to achieve goals". In *Proc. of IJCAI-93*, pages 1094–1098, 1993.
17. L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement learning: A survey". *Journal of Artificial Intelligence Research*, 4, 1996.
18. G. J. Gordon. "Stable function approximation in dynamic programming". In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 261–268, 1995.

19. J. N. Tsitsiklis and B. V. Roy. "Feature-based methods for large scale dynamic programming". *Machine Learning*, 22:1, 1996.
20. L. Baird. "Residual algorithms: Reinforcement learning with function approximation". In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37, 1995.
21. R. S. Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In *Proc. of Neural Information Processing Systems 8*, 1996.

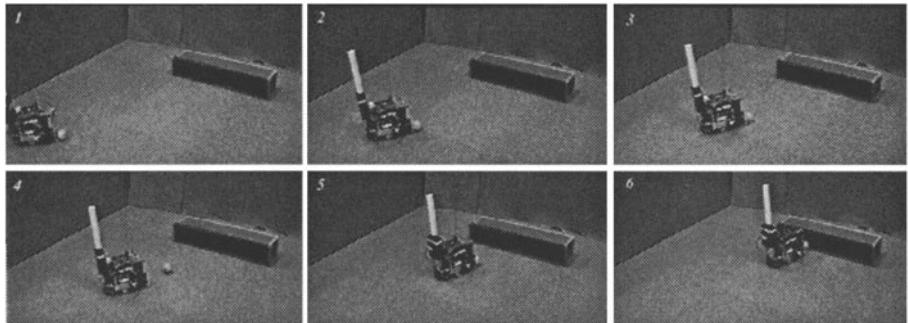


Fig. 11. Shooting behavior on our robot

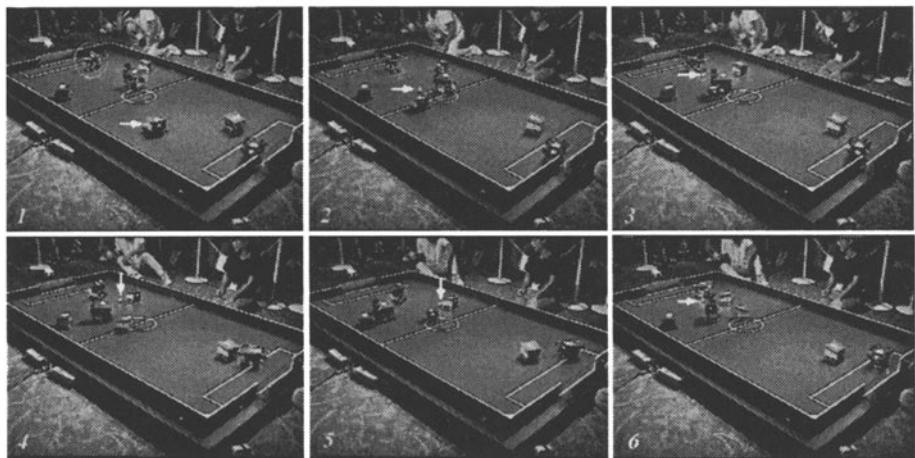


Fig. 12. Defending behavior on our robot

MICROB: The French Experiment in RoboCup

Sébastien Rocher¹, Vincent Idasiak¹; Sébastien Doncker²; Alexis Drogoul² and Dominique Duhaud¹

¹ Laboratoire de Robotique de Paris, Université PARIS 6, UVSQ, CNRS, France

² LAFORIA, Université PARIS 6, CNRS, France

Abstract. In this paper, we present the ongoing research done in the field of robots playing “football”. Several development aspects are discussed such as the hardware system, followed by the software implementation. We will highlight three softwares : the central decision system and the embedded software (used for the real robots small RoboCup league), and the simulation software (used for the simulation RoboCup league). So, we will show and discuss, for each kind of competition, the implemented behaviors.

1 Introduction

The idea of robots playing “football” came at the same time within two fields of research : robotics and computer science [4, 5]. We decided to work on this subject at the end of 1995. The idea was to build and experimental platform : MICROB, to validate on real robots different kind of multi-agents algorithms. Thus, the MICROB platform is well adapted to solve such problems as the football game.

In the first part, we present the team participating for the small league of the RoboCup competition and we focus on the decision system.

In the second part of the paper we present the simulation team. Here, we tried to design a team in which all the players processes the same behaviour. We discuss the limits of the above approaches.

2 The MICROB team for small size competition

Our platform is divided in two parts : the robots and the central decision system (*Fig 1.a*) .

2.1 The robots architecture

The physical structure of the robots (*Fig 1.b*) is divided in four layers (*Fig 1.a*):

- the **mechanical layer** : the robots designed have two independent wheels on the middle of their structure. They have two degrees of freedom in the plane and they are non holonomic. Each wheel is motorized by an actuator consisting on a small motor. The motor power supply is carried by the robot and consist on a small battery 12V.

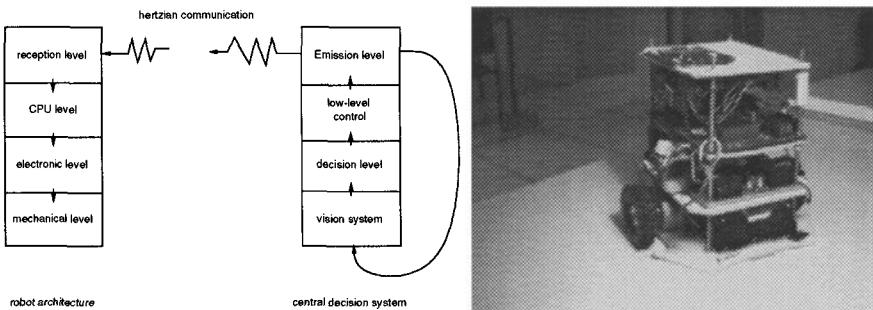


Fig. 1. a) The general platform architecture **b)** One of our small size robot

- the **electronic layer**, which uses a tachometric feed-back.
- the **CPU layer** : the CPU is a 80C552 processor, with a 16 MHz frequency and 64 Kbytes of memory. It has only to decode the data coming from the hertzian receiver of the robot.
- the **communication layer** : it receives orders from the central decision system using an hertzian transmission (9600 bauds).

2.2 The central decision system

For the central decision system, we use an Intel-Pentium 90 CPU with 32 Mbytes of memory and MS-DOS operating system. In this section, we describe the vision system, and the behavior module (the theoretical behavior and the experimental one).

The vision system. The vision system takes a large part in the global system performance [6]. It has to extract pertinent data from video acquisition :

- the **position** and the **orientation** for each robot,
- the **position** and the **velocity vector** of the ball.

Moreover, because of the real-time constraints, the acquisition and the processing of a video frame has to take less than 10 Hertz. The acquisition card used (the IFG Imasys frame grabber card) provides black and white video. The image processing extracts the outline of the objects located on the playing field. To identify the robots, we put various marks on the top of each of them. The directions of the robots are determined using the knowledge of their outline. For the ball, the velocity vector is deduced using the previous detected positions. Due to the time processing, and the resolution of the image (256×256), one pixel maps $1cm^2$, which affects the precision reliability of the system.

The theoretical behavior. A multi-agent architecture (*fig 2.a*) is proposed to define the behavior of the robots (in our study the agents are the robots). This architecture is composed of four main modules : a module representing the

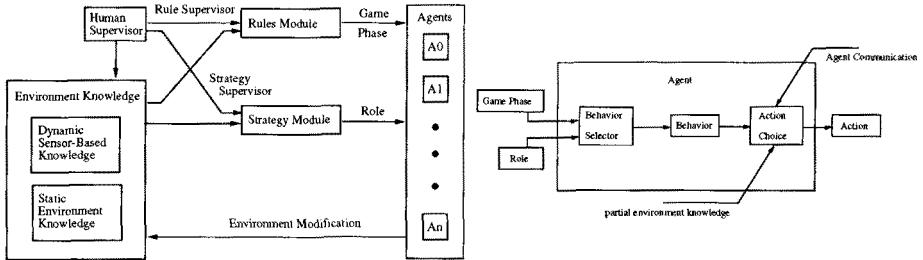


Fig. 2. a) The general Multi-agent ARCHitecture b) The agent model

knowledges about environment, a **rules module**, a **strategy module** and a set of **agents**.

- **The environment knowledge.** In our system, there are two kinds of environment knowledge:

- the **static knowledges**: the limits of the playing field, the kickoff position and the penalty position.
- the **dynamic knowledges**, which are computed on-line : the position and the velocity vector of the ball, the position and the orientation of the robots.

- **The rules module.** The football game is divided in game phases. The different phases we implemented are : the **kick-off** phase, the **free kick** phase, the **penalty kick** phase, the **stop** phase, and the **main game** phase. The rules module allows to define the current phase of the game. It acts mostly under the influence of the human supervisor. Indeed, a human supervisor can decide to stop, to start a game, etc... In certain cases, the module of rules allows to change game phases automatically (for instance, from the kick-off phase, the penalty kick phase or the free kick phase to the main game phase).

- **The strategy module.** The goal of the strategy module is to define the role of each agent within the team. This role can be defined in a static or a dynamic way, according to the environment knowledge. Two main roles are defined : the **goalkeeper** role, the **field player** role. Moreover, specific roles can be added to the main ones. These are : the **penalty kicker** role, the **Kick-Off kicker** role.

At most, one robot can have the **goalkeeper** role during the game. To be efficient, the **field player** role must be different for each robot of the application. The idea is to distinguish the **field player** roles by giving to each robot a specific zone on the playing field. So, we will have sub-roles of the **main field player** role : we will be able to have a **defender player**, a **middle field player**, a **attaquant player**, etc. This will be developed in the next section.

- **The agent behaviors.** In this approach, each robot is an agent. Two functions are used to select the behavior (function s) (1) and to select the action of a robot (function f) (2) :

$$\text{Behavior} = s(\text{Role}, \text{GamePhase}). \quad (1)$$

$$\text{Action} = f(\text{Behavior}, \text{EnvironmentKnowledge}, \text{Communication}). \quad (2)$$

The behaviors are selected according to the role of the agent and the phase of the game (*fig 2.b*). Thus, the selected behavior will define the action to be executed by the robot, according to the environment knowledge and the inter-agent communication. The behaviors will have to obey to the game rules. The different behaviors implemented are : the **specific goalkeeper** behavior, the **specific field player** behavior, the **keep stopped** behavior, the **kickoff player** behavior, the **penalty kick player** behavior.

The experimental behavior. In this section, we will develop the implementation of the main behaviors (the goalkeeper behavior and the specific field player one) for the RoboCup experiment. Two configurations were realized to adjust to the robot number of the opponent team : one game with a five players team and one game with a two players team. Indeed, due to the competition requirements, we played with 2 robots against the Japanese team. Thus, we had to change our strategy.

• **The actions.** We distinguish two kinds of actions : the **high-level** actions and the **low-level** ones. The high-level actions define goals to be realized, such as : replacement, ball kicking, avoidance, move for a position with an orientation. These high-level actions will be divided into low-level actions. These low-level actions are : the **forward gear**, the **reverse gear**, and a **position and orientation low-level servoing**. The third action is the most interesting. To implement it, we used a control system well adapted to our robots configuration, and to our problematic. It allows a position and orientation servoing, using a constant linear velocity of the robot [2].

• **The goalkeeper behavior.** The goalkeeper behavior is very simple : during the game, we place the goalkeeper on a straight segment, between the ball and the goal.

• **The specific field player behavior.** It uses different zones of influence for each different field player behavior. The algorithm of the behavior can be resumed in a pseudo-code like this :

IF (the ball is not in the zone of influence of the agent)

THEN Replacement.

ELSE

IF (the agent is not well placed to kick the ball)

```

    THEN Replacement.
ELSE /* Beginning of contract net */
the agent communicate to other agents the
evaluation of his position to kick the ball.
IF (the agent is the best placed)
    THEN the agent kick the ball.
ELSE
    Replacement.
ENDIF
ENDIF
ENDIF

```

The idea is to have stackables zones of influence in the robot team. So, each time, several robots can kick the ball to attack or to defend. Finally, only one robot will kick the ball (the one which is the best placed), and the other ones execute a replacement.

- ***The five players game.*** For the game with five players, we defined the following roles : one goalkeeper, one defender player, one left middle field player, one right middle field player and one attaquant. To define the zone of influence assigned to each robot, we decided to divide the game field into 9 parts (fig 3). Then, each field player has a specific zone of influence. We also have to define

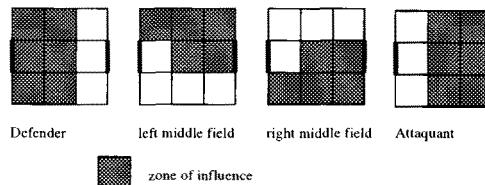


Fig. 3. Zones of influence for each field player robot

the replacement position. In our experiment, the replacement position depends only on the ball position (fig 4). During all the five players game, the agent role

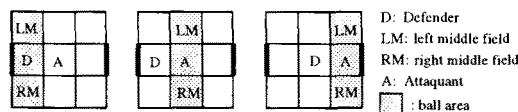


Fig. 4. Replacement positions of robots, according to the position of the ball

was always the same.

- ***The two players game.*** The game with two players involving the Japanese team is different from the one with five players. Indeed it is much

more difficult to kick the ball with two players than with five. So, we used this configuration : one defender/goalkeeper player and one attaquant player. For the game with only two agents, we decided to give the maximum of freedom to our robots. So the zones of influence are the same for the two robots and they represent all the game field. But the behaviors are not exactly the same for the two robots. Indeed, the replacement position is different (*fig 5*). This configuration is

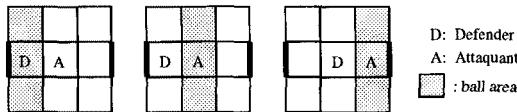


Fig. 5. Replacement positions of robots, according to the position of the ball

particularly interesting, because it allows us to test the role selection function. Indeed, the role of one of the robots is dynamic. It plays as a defender or as a goalkeeper according to the environment knowledge. If the ball is in our camp and if the direction of the ball was toward our goal, then the strategy module imposes goalkeeper role to one agent.

2.3 The limits of the team

The limits of our robots team are due partly to the low quality of equipment used. Actually, to carry out our experiment, we used a cheap equipment : a PC (pentium 90), a black and white acquisition card with an acquisition rate of 18 Hz. For the robots, we used secondhand actuators, and mechanical engineering and the electronic card have been realized by our own. The cost of all the equipments is approximatively 5000 dollars.

The main problem arise from the vision system : the vision system is too slow and the precision of the detection is inaccurate. Indeed, the global precision of our system is about 2 or 3 cm (regarding robots speed). This problem made useless the implementation of evolved actions and so, the behaviors were limited.

The second problem is the low quality of the hertzian transmission between the central decision system and the robots. Now the path processing being computed by the central system, the robots are very dependent on the communication chanel quality.

3 The Microb simulation team : exploring a strict bottom-up approach

Unlike the approach used for the real robots, where the goal was to provide existing robots with coherent behaviors regardless of the methodology employed, the development of the simulation team has been initiated to track the methodological issues raised by the soccer problem (see [1] for a different perspective on

the same issue), and to illustrate some properties of reactive multi-agent systems [3]. The basic idea was to follow a strict bottom-up approach to the problem of organizing a team, beginning with very simple players and supplying them with the behaviors that could appear necessary with regard to their collective play. The hypotheses that have underlined our work, and to which we have remained committed throughout the different versions are :

- **Hypothesis 1** : behavioral homogeneity . All the team mates are identical. As a consequence, the behavior of the team does not rely on a coordination between specific roles (e.g., the goalie, defenders, etc.), which are therefore not described nor used in the implementation. However, even if the agents are not provided with a given role, their position on the field will induce different behavioral responses from their partners.

- **Hypothesis 2** : environmental heterogeneity. The activity of reactive multi-agent systems usually rely on the information the environment can provide them. The computation of a simple topology, which divides the field in eleven zones of influence supplied with different marks, is intended to dispatch dynamically the agents on the playground during the game, regardless of their current behavior).

- **Hypothesis 3** : no direct communication. This restriction lets the agents use the characteristics of the game as a communication-like protocol : passing the ball or entering/leaving a zone is supposed to be sufficiently explicit for the others.

- **Hypothesis 4** : no planning. The agents do not look ahead. They do not plan their behaviors after the current cycle they are engaged in, nor do they anticipate the behaviors of the other participants.

- **Hypothesis 5** : no memory. The agents do not record the previous information they have obtained through their perception. Of course, their velocity, as well as their direction, can be considered as a sort of embedded memory of their former behavior, in the sense that it implicitly influences their current behavior.

- **Hypothesis 6** : no explicit deliberation. Due to the constraints above, the agents do not use explicit deliberative mechanisms for controlling their activity. Rather, a subsumption-like architecture, with a fixed structure, allows them, at each cycle, to behave accordingly to a fixed set of situations they may encounter (fig 6).

- **Hypothesis 7** : a small set of behaviors. The behaviors organized within this architecture are only five : **move to another zone, look at the ball** (i.e. continuously changing direction to follow the position of the ball, but without moving), **chase the ball** (i.e. the same with a movement), **kick the ball** (i.e. kick the ball toward the opposite team), **pass the ball** (i.e., kick the ball toward a team mate).

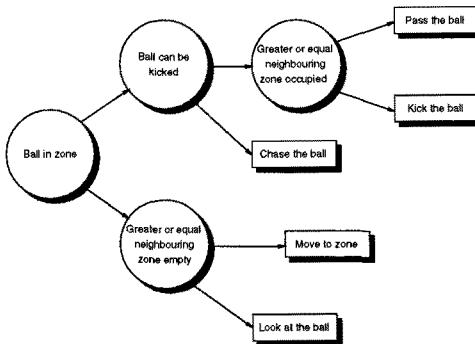


Fig. 6. The tree-like decision algorithm and the associated behaviors

Given these hypotheses, one could have expected the collective game to be fairly poor. Indeed, the behavior of a sole player is easily predictable, and it is quickly defeated by any other opponent (like Ogalets's ones, for example). However, collectively, things appear to be a lot more interesting, with truly coordinated movements emerging from the interactions between the simple behaviors of the players. It is especially the case when the team is attacking, elaborating surprisingly good collective actions that can easily beat any defense. The main reason is that the players, because they do not need to communicate nor plan anything, are much faster at moving or passing the ball than most of the teams they have already been facing. But the behavioral homogeneity allows the players to occupy any zone on the field and thus making the team dynamically and smoothly cover the playground.

Of course, we have detected, during the experiments, numerous drawbacks for such a simple approach :

- The players often miss the ball because they are always heading toward it regardless of its speed or direction. This lack of anticipation is especially painful for the “goalie” (i.e. the player being placed in the corresponding zone), which does not care about moving until the ball reaches its zone.
- During difficult and long actions, four or five players can happen to chase the ball as if they were alone in their zone. It is usually due to the initial dash which made them leave their zone (without them paying attention to it). Because they can not replace themselves while the ball is too close, the result slightly disordered.

These two defaults combined often ruin the interesting collective behaviors that could otherwise emerge. But the interesting thing is that, given the proposed methodological (and somewhat radical) approach to the problem, we now know exactly what to improve in the individuals’ behaviors for dynamically obtaining emergent collective strategies that could remain stable throughout the game. Our approach, of course, which will consist in carefully adding new behaviors, will not prevent us from possible side effects (i.e. unwanted new collective

structures). However, it will be certainly the price paid to understand the link between individual and collective behaviors.

4 Conclusion

From experimental observations, we suggest to give futher enhancements to our design. To have good result for the real robots competition, the hardware must be very efficient. Thinking it is possible to control a mechanical structure which is not optimal, is not a good idea. Having good result means having a good software, but also a good hardware. Each layer is important : control, trajectory generation, behaviours, vision, communication. We tried to use some cheap components thinking that a good behaviour could correct the uncertainty of the system. This is not reasonable. For the next competition we can expect to have the best at each layer.

The future of the simulation team relies in our capacity to explore the link between individual and collective beaviors verifying hypotheses presented above. The important thing is that collective responses to unpredictable situations appear to be often qualitatively surprising, especially when facing good teams. But the issue is that the robots can not take advantage of their experience of these collective achievements, because they have no available mean to adapt their behaviors (both the way they classify the situations and their reaction to them) with respect to the previously encountered situations. As a consequence, learning capabilities, built on top of their current behaviors, are already being provided to our players (this requires us to release some constraints, such as the absence of memory). And we intend to prove in a near future that having adopted a behavioral bottom-up approach greatly eases this process.

References

1. Collinot A., Drogoul A. and Benhamou P., "Agent Oriented design of a Soccer Robot Team", Proceedings of ICMAS 96, AAI Press, Nara, Japan, 1996.
2. S. Delaplace, "Navigation a cout minimal d'un robot mobile dans un environnement partiellement connu" *Ph.D thesis, University of Pierre et Marie Curie (Paris 6)*, France, 1991.
3. Drogoul A., Corbara B., Lalande S. "MANTA: new experimental results on the emergence of (artificial) ant societies", in Gilbert N. and Conte R. (Eds.) *Artificial Societies*, UCL Press, 1995.
4. H. Kitano, Y. Kuniyoshi, I. Noda, M. Asada, H. Matsubara, E. Osawa, "Robocup: a challenge problem for AI," *AI magazine Vol. 18 N°1*, pp 73-85, 1997.
5. M. K. Sahota and A. Mackworth, "Can situated Robots Play Soccer?", in proc. Artificial Intelligence '94, pp. 249-254.
6. R. Sargent, B. Bailey, C. Witty, A. Wright, "The importance of fast vision in winning the first micro-robot world cup soccer tournamennt" *Journal of Robotics and Autonomous System Vol. 21, N° 2*, pp 139-147, 1997.

Description of Rogi-Team*

A. Oller, R. Garcia, J. A. Ramon, A. Figueras, S. Esteva, J. Ll. de la Rosa,
J. Humet, E. Del Acebo

Systems Engineering and Automatic Control Group, eXiT
Applied Informatics Institute (IIIA)

Associated European Laboratory Intelligent Systems and Applied Control (LEA-SICA)
University of Girona
Catalonia

{oller, rafa, josepa, figueras, sesteva, pepluis}@eia.udg.es
{humet,acebo}@cima.udg.es

Abstract. Multi-agent decision-making structures are expected to be extensively applied situations in complex industrial systems (namely distributed systems like energy or material distribution networks, big plants with several stations and difficult processes with huge number of variables). Multi-robotic soccer competitions are a forum where rules and constraints are good to develop, apply, and test those structures. This paper shows new developments under Matlab/Simulink that allow decision-making among agents which, in this case, co-operate to play soccer game.

1 Introduction

To facilitate the automatic control community to develop agents in control systems analysis, design, and implementation, in this work we propose to incorporate agents capabilities and facilities in Matlab/Simulink² (called M/S hereafter). These facilities were initially introduced in [de la Rosa et al.1997] by means of agent-oriented language (AOL) called AGENT0 that brings a language to analyse agents based on belief, goals, commitments and capabilities. Work went on specialising the first approach by introducing the classical structure behaviour supervision control algorithms. Both the multi colour based vision system and the micro-robot control structure are specified in the paper. The micro-robotic system is designed to get up to 1 m/s robotic movements and to control them, in the sense of dynamic systems to require of advanced control technology. Section 2 introduces the robotic system. Section 3 describes further the three level approaches to solve the control and behaviour of robots and their co-operative behaviour, all specially related to the supervision and path planning level. Section 4 is a discussion about the improvements that agents represent to control systems analysis, design and implementation of autonomous robots and section 5 is future work.

* This work has been funded by the CICYT TAP97-1493-E project “Desarrollo de una segunda generación de equipo microróbótico. Consolidación de la competitividad internacional”, of the Spanish government.

² MATLAB (ver. 4.2c.1) and SIMULINK (ver. 1.3c) are trademarks of Mathworks Inc.

2 Materials and Methods

The robots were designed for the MIROSOT'97 competition [KAIST1997] and adapted to Robocup'97 features. Because of MIROSOT rules, robots are limited to the size $7.5 \times 7.5 \times 7.5$ cm. and must run autonomously through a 130×90 cm. sized ground. So far, teams can use a centralised vision system to provide position and orientation of the robots so as ball position. A computer host to calculate individual robotic movements that are send to the robots to be executed uses these parameters. Planning of co-ordinated motions for robots playing a soccer game opens a new deal in motion planning, namely co-ordinated motion planning in team competition. This problem is quite challenging due to the fact that the game situation continuously vary, that is, this kind of game consist of a dynamic system which constantly evolve.

The robots are devised and controlled taking into account two main problems:

- High-level control. Mean tasks are to assign positions to the robots and generate commands to be executed by them. All these tasks are executed on the computer host.
- Low-level control. Mean task is to execute with accuracy the commands generated at the high-level. This level consists of a closed loop control.

2.1 Technical Specifications of the Robots

According to MIROSOT rules, robots have strong limitations on their size so that communication features so as autonomous motion capabilities must be implemented by using small devices. Robots are composed by one receiver, a CPU unit, a power unit, two micro-motors, and batteries for motion, communication and control unit supply voltages.

Here follows a brief description of some important components of the robots:

- **87C552:** This microcontroller contains a non-volatile $8k*8$ read-only program memory, a volatile $256*8$ read/write data memory, 6 configurable I/O ports, two internal timers, 8 input ADC and a dual pulse with modulate output.
- **L293:** Contains two full bridges and drivers, with an external component, which is capable to control the speed and the sense of rotation of both motors.
- **Sensors:** Two photoelectric switches, E3S of OMRON, with adjustable distance of detection. Its outputs are open collector.
- **Motors:** MAXON DC motor, 9V, 1.5W, nominal speed of 12.300r.p.m, MAXON gear (16:1).
- **Encoder:** Digital magnetic encoder with 16 counts per turn.
- **Receiver:** Super-heterodyne receiver with monolithic filter oscillator. MOD. STD 433 SIL. 433,92 MHz. Totem Line. AUR·EL S.p.A
- **Batteries:** 9 batteries of 1.2 V-1000 mA/h supply each robot.

2.2 Overall System

Robots do not have any camera on board so that a global vision system is needed. In this work, the vision system [Khatib1985] is implemented on a Pentium PC-

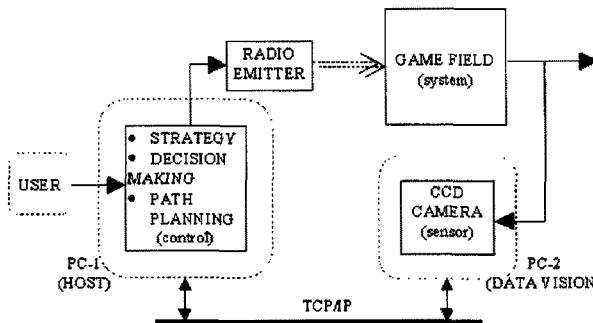


Fig. 1. High-level control structure

computer which process 7 frames/sec. Obtained data are positions of all the objects in the game field (ball and robots) and orientations of the own team robots. Data are shared with another computer using TCP/IP communication protocol. This computer host contains strategy parameters, decision making and path planning algorithms (see Fig. 1) and generate frames to be sent to the robots, as follows:

ID_ROBOT | #commands | comm_1 | comm_2 | ... | behaviour_byte | ETX (1)

where $\text{comm}_X := \text{angle} | \text{distance}$.

This frame allows the robot to execute movements through the ground fast and avoiding obstacles. Since the maximum speed of the robot can be 120 cm/sec (straight way), usual speed values are 50 cm/sec to avoid skidding. In such a way, the robot is made to follow the path based on a guidance matrix obtained by the path planning method explained next. Since obstacles move so as the ball do, this freeway path sometimes becomes wrong. When this problem appears, then on board sensors and 'behaviour-byte' are helpful to the robot in order to react in front of obstacles and walls, or look for the ball.

2.3 Path Planning

The used algorithm is based on Artificial Potential Fields (APF) approach [KAIST1997]. A Cartesian coordinate system is used to define the position of the centre of each square cell, which allocate the value of the associated potential. When the guidance matrix elements have settled, the algorithm will systematically scan the robots three neighbouring cells (see Fig. 2) to search for a potential value that is lower than that of the cell currently occupied by the robot. The trio

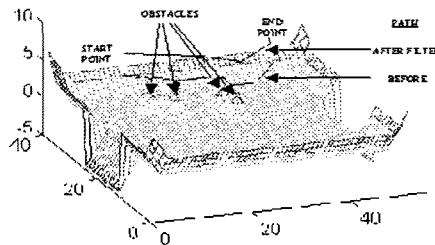


Fig. 2. Example of path finding result with four obstacles

of cells to be scanned depends on the orientation information given by the relative direction from robot to set point position: N, NE, E, SE, S, SW, W, and NW. In such a way, the scanning method is goal finding.

The obtained guidance matrix gives a path that could be translated directly into a sequence of commands described by (1), but the real execution of such path has no sense because has an high number of turns. Therefore, at the end of the algorithm, a filtering process is applied and a smoothed path is obtained, as shown in Fig. 2. Then, frames are sent to the robots.

2.4 Low-level control

This section describes the hardware of the robot and then explain the main program of the robot which allows distance, angle and velocity control.

Hardware description Hardware robot is devised from an 87C552 microcontroller, which has capabilities to change the speed of the robot using two PWM outputs, according to velocity control requirements. Actuators are two DC motors, including gear and encoder in the same axis of the motor. The connection between microcontroller and these motors is done through a L293 integrated circuit. Used sensors consist of two photoelectric ones: one is placed high enough in order to detect obstacles but not the ball, and the other is placed to detect the ball into a closed area. By integrating these two sensors, robot control procedure can efficiently distinguish which kind of thing is detected in front of the robot: ball or obstacle. Finally, communication between computer host and robots is carry out by radio-communication then every robot has one receiver and one identifying code (ID_ROBOT) as shown in expression (1).

Main program When the robot (see expression (1)) receives frames sent by computer host, they are immediately executed. Every command contains the following information:

$$\text{Comm X} := \text{angle} \mid \text{distance}$$

Where distance parameter so as angle one are 4-bits codified. Therefore, the robot is made up to follow paths as those shown in Fig. 5-b with *Distance* and *Angle* values included in the next sets:

$$\text{Distance} = \{0, 5, 10, \dots, 45\} \text{ cm} \rightarrow 10 \text{ values}$$

$$\text{Angle} = \{+90, +67.5, +45, +22.5, 0, -22.5, \dots, -90\} \text{ deg} \rightarrow 9 \text{ values}$$

Robot speed is set up to 100cm/sec using a slope function and, when the last command is executed, another slope function is used to stop the robot. Using three signals carries out the control of the movements: those coming from the encoders (distance closed-loop) and battery voltage. Encoders provide information related to count per turn so that *distance* and *angle* can be measured. However, batteries voltage signal is used to ensure the motors supply: if batteries voltage decrease then motors supply is modified by increasing the weight pulse of the PWM signal.

While commands are executing, sensors can interrupt the main program because photoelectric sensors are directly connected to the interruption lines. When an obstacle or a ball is detected then robot moves accordingly to behaviour.byte.

2.5 Co-operation

The decision making structure is devised in two hierarchical blocks: higher one contains those tasks related to the collectivity (co-operative block), that is to say, communicative actions like inform and request, and lower one where the agent executes its private actions (private block).

- Private Block: Every agent is able to execute several actions according to a set of *capabilities*, in terms of AGENT0 language. In this task, this set is forced to include *persistence* capabilities because the environment is dynamic and continuously changing. This block manages several types of information so that three levels are defined depending on the abstraction degree of this information: agent level (behaviour decisions), supervisory level (path planning –hidden intentions– and set-point calculations), and control level (execution of set points).
- Co-operative Block: The agent level is the connection between the agent itself (autonomous robot) and the collectivity so that it includes communicative actions to *request* or *inform* about decisions. This block is based on Agent-Oriented Programming, and uses the set of communicative primitives defined by AGENT0 [Shoham1990].

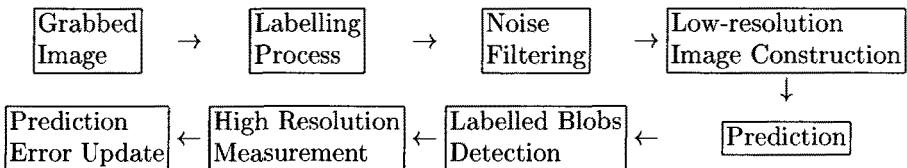
With this structure, decision making procedure can take into account both collective and individual decisions, based on the communication between agents and the exchange of decisions between the behaviour and supervisory levels.

By this way, collective decisions are based on strategy rules, and individualised ones are based on a subjective cost index obtained from the path planning curve properties. This curve is obtained based on the use of Artificial Potential Fields (APF) [Khatib1985] and some modifications proposed by [Clavel et al.1993]. This

index is calculated in the supervisory level and, because it is contained into private block, it contains different parameters for every agent, in other words, different behaviour for every agent that is included in the APF method.

2.6 The Vision System

The vision system is composed of a colour camera, an specific colour processor and a multiple purpose frame-grabber. The camera is located 2 meters over the playing field and the vision system is able to segment and track the moving robots and the ball over a sequence.



Every robot is equipped with two color bands: the first one is the colour of the team and the other one is a robot specific-colour. Finding the center of gravity of both regions allows the system to know the position and orientation of the robots.

The segmentation step is performed by means of an specific processor which performs a real-time conversion from the RGB to the HSI space. This processor uses the discriminatory properties of the three colour attributes: Hue, Saturation and Intensity, in order to segment everyone of the regions in the scene. Selecting different values of these attributes for every object avoids the correspondence problem in multi-tracking environments. The result of the segmentation model is an image 768×576 with several labeled regions on a black background. The cycle time of the segmentation module is 40 ms, that is, video rate on the CCIR standard.

The second step consists on the computation of the position and orientation of every object by using the enhanced image from the segmentation module. As a first step a median filter is applied to the region-labeled image, in order to reduce noise. Then a multiresolution approach is applied : a second image is constructed, taking one pixel over four from the filtered image. This second image is scanned searching for the labeled regions, and when these regions are detected the exact position is computed in the high-resolution image, in order to increase the accuracy of the measurements.

There is a state vector which keeps the information about the position of every robot. At every time step, a new prediction is performed of the next position of every object using a 2D polynomial regressive model over the low-resolution image. This prediction is used in the next time step in order to search for the object, and an error factor is measured to be taken into account for the next prediction. The cycle time of the tracking module depends on the data to be processed and the accuracy of the prediction. Normal cycle time is around 150 ms.

At the end of every processing cycle the vision system sends the state vector to the decision agents by means of a TCP/IP connexion.

3 Design of the Co-operation System

How do include agents in the soccer game? The approach we proposed in [de la Rosa et al.1997] was by using the existing CACSD (Computer Aided Control Systems Design), specially M/S that is extensively introduced in the automatic control field. However, the way we introduce the co-operation among micro-robots is in conventional layers of co-ordination according to standards of CIM (Computer Integrated Manufacturing), because production systems are structured in levels to determine the complexity and specialisation in analysis and design of automatic systems.

Fig. 3 represents the way the information, analysis, design and implementa-

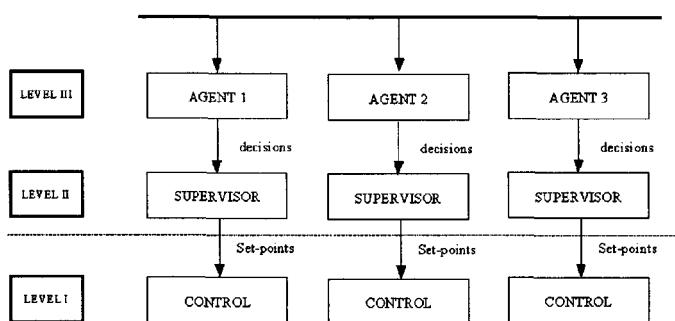


Fig. 3. Layered representation of behaviour, supervision, and control

tion, is distributed in each robot. Also, all co-operations are responsibility of the behaviour level. The AOP and agents could be applied to any level although to start the application of agents in the behaviour level is advisable. Except for this general philosophy we are not applying any protocol from CIM.

Thus, the co-operation of the team of robots is done at the agent (behaviour) level, that is also responsible for the behaviour of each robot by exchanging messages to the supervision level.

Originality of our work is the representation of co-operative behaviour by means of an agent oriented language (AOL) that is developed in a CACSD framework so that the link behaviour-supervisor could be easily implemented with tools from the automatic control field.

However, new difficulties appear in the functional protocol between the *behaviour* and the *supervisor* levels, namely the *persistence* of goals that the AGENT0 language lacks. This lack is overcome by tuning the sample time of agent↔supervisor communication that leads to persistence develop at the supervisor level.

4 Discussion

The previous work [de la Rosa et al.1997] contained important new items such as the adaptation of an AOL (AGENT0) to the M/S application. Although M/S has not any proper language to implement AOL the AGENT0 development was possible although difficult. That implementation contemplated the automatic code generation with RTW (Real time Workshop) toolbox of M/S that was partially solved. Another novelty was the inclusion of fuzzy logic in the behaviour decision process, incorporated to the AGENT0 language.

This work has developed the soccer system that contains the following new features:

- Explicit partition of the behaviour-supervision-control levels, developing mainly the behaviour-supervision levels.
- Exchange of information between behaviour \leftarrow rightarrow supervisor levels, in the sense the supervisor (developed with toolboxes of M/S) executes targets proposed by the behaviour level and the behaviour level obtains suggestions about difficulty in possible targets to feed its co-operative behaviour.
- The exchange of fuzzy evaluations in the Behaviour \leftrightarrow Supervisor communication is included. The fuzzy capabilities in the AOL of the Behaviour can use now fuzzy evaluations of difficulties in possible targets. Now, the vision system is a sensor that provides with information all three levels of the soccer robots, with different sample frequencies from 7 frames/second (at the control level) to 1/2 frames/second (at the behaviour level).

Lacks of the current work

- The persistence problem is not finely solved because the solution of focusing on the supervision level to maintain the decision of the behaviour level is rather rough due to the highly changing nature of the soccer game. The persistence means to maintain previous decisions while it is obvious there is no proper alternative.
- To link supervision \leftrightarrow control levels is also necessary because supervision systems are not so far fully developed to calculate physically good set-points that the control level of the robot could execute. To incorporate this exchange, new controllers technology should be included to robots.
- A new language closer to control engineers and applied mathematics is necessary to develop agents under AGENT0 language.
- Engineers do still not accept the design methodology proposed in previous work [de la Rosa et al.1997] since they do not have the tools to develop it.

The work that was intended but technical problems delayed it to the future work is, AGENT0 implementation with object oriented languages in M/S. This provoked that the agents' librarian, with agent facilities, is not implemented yet

in Simulink. Thus, code generation with the new agents librarian is still unsolved. This is expected to be solved with the newer version 5 of M/S that appeared this spring of 1997.

5 Conclusions and Future Work

The experience in Robocup is about the lack of robustness in our approach. Even though the persistence in agents' plans was a task of the supervision level, the generation of events from the dynamical (continuous state and time variables) is a complex matter. The suggestion to solve it is to apply theory from hybrid systems.

Therefore, although to use agents in control systems design is possible, the developed tools are still rough to call for attention of control engineers and applied mathematics. Good results in analysis and implementation of these ideas so that to develop soccer robots is nice and clear in analysis (but not in design, that takes a lot more time than classical approaches, although then implementation is, more or less, automatic) encourage us to continue in the research.

Future work is to introduce predictive control algorithms in the control level to ease the supervision↔control activity. Also work in hybrid systems for supervision. Furthermore, the PLACA language will be included to compare how does it improve our current work in terms of persistence and analysis representation poverty, after the improvement of the behaviour↔supervision communication contents. By the way, we expect to increase the use of agents in simple automatic control applications with partial development of the agents' librarian within this framework.

References

- [Clavel et al.1993] N. Clavel, R. Zapata, F. Sevilla, "Non-Holonomic Local Navigation by means of Time and Speed Dependent Artificial Potencial Fields", *IFAC Intelligent Autonomous Vehicles*, Southampton, UK, 1993.
- [Khatib1985] O.Khatib, "Real Time Obstacle Avoidance for manipulators and Mobile Robots", *IEEE*, St. Louis (USA), 25-26 March 1985.
- [KAIST1997] Micro-Robot Soccer Tournament, KAIST, Taejon, Korea, 1 - 5 June, 1997.
- [Shoham1990] Y.Shoham, "Agent-Oriented Programming", Technical Report STAN-CS-1335-90, Computer Science Dep., Stanford Univ., Stanford, CA, 1990.
- [Thomas1992] S.R. Thomas, "A logic for representing action, belief, capability, and intention", Stanford Working Document, Stanford, CA, 1992.
- [de la Rosa et al.1997] de la Rosa J. Ll., Oller A., Vehí J., and Puyol J., "Soccer Team based on Agent-Oriented Programming", *Robotics and Autonomous Systems*, Ed. Elsevier, Oct 1997.

Autonomous Soccer Robots

Wei-Min Shen, Jafar Adibi, Rogelio Adobatti, Bonghan Cho,
Ali Erdem, Hadi Moradi, Behnam Salemi, Sheila Tejada
Computer Science Department / Information Sciences Institute
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292-6695
email: {shen,dreamteam}@isi.edu
URL: <http://www.isi.edu/isd/dreamteam>

Abstract. The Robocup 97 competition provides an excellent opportunity to demonstrate the techniques and methods of artificial intelligence, autonomous agents and computer vision. On a soccer field the core capabilities a player must have are to navigate the field, track the ball and other agents, recognize the difference between agents, collaborate with other agents, and hit the ball in the correct direction. USC's Dreamteam of robots can be described as a group of mobile autonomous agents collaborating in a rapidly changing environment. The key characteristic of this team is that each soccer robot is an autonomous agent, self-contained with all of its essential capabilities on-board. Our robots share the same general architecture and basic hardware, but they have integrated abilities to play different roles (goalkeeper, defender or forward) and utilize different strategies in their behavior. Our philosophy in building these robots is to use the least possible sophistication to make them as robust as possible. In the 1997 RoboCup competition, the Dreamteam played well and won the world championship in the middle-sized robot league.

1. Introduction

The Robocup task is for a team of multiple fast-moving robots to cooperatively play soccer in a dynamic environment [6]. Since teamwork and individual skills are fundamental factors in the performance of a soccer team, Robocup is an excellent test-bed for autonomous agents. For this competition each of the soccer robots (or agents) must have the basic soccer skills -- dribbling, shooting, passing, and recovering the ball from an opponent. Each agent then must use these skills in making complex plays according to the team strategy and the current situation on the field.

An agent must be able to evaluate its position with respect to its teammates and opponents, and then decide whether to wait for a pass, run for the ball, cover an opponent's attack, or go to help a teammate; while at the same time following the rules of the game. In the following sections of this paper we will describe the general architecture for an autonomous robot agent and the team strategy, as well as discuss some of the key issues and challenges in the creation of a team of autonomous soccer agents.

2. General Architecture

For this project we define an autonomous agent as an active physical entity intelligently maneuvering and performing in realistic and challenging surroundings [4]. The critical design feature of our autonomous soccer agents is that each agent has all of its essential capabilities on-board, so that every robot is self-contained. Autonomous systems, such as these, need to be physically strong, computationally fast, and behaviorally accurate to survive the rapidly changing environment of a soccer field. In our general architecture, great

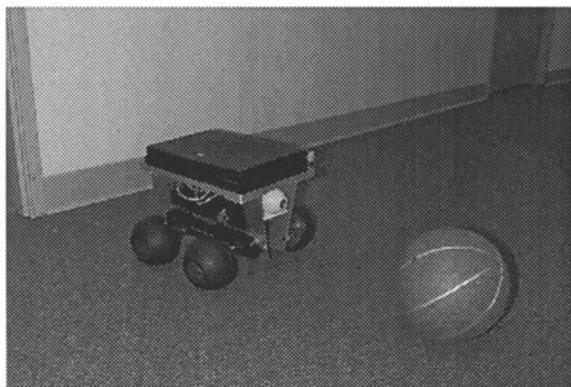


Figure 1. An Autonomous Soccer Robot

importance is given to an individual robot's ability to perform on its own, without outside guidance or help of any kind. Each robot agent bases its behavior on its own sensor data, decision-making software, and eventually communication from other teammates. There are many techniques in robot agent modeling and design [1,2,3,7]; however, much of the work in this area has focused on single agent performing and sometimes under external supervision. We believe that the further of robot agents lies in total autonomous, with the capability of learning and adaptation to the environment [4,5]. Moreover, agents have to be intelligent enough to cooperate among themselves.

Each robot consists of a single 586 based computer on a 30cm x 50cm, 4-wheel drive, DC model car (Figure 1). The computer can run a program from its floppy drive or hard drive. However, due to the robot's very harsh working environment, we decided to store our program in a Solid State Disk (SSD) to avoid any mass-storage malfunction during competition. The robots can be trained and the programs can be upgraded by attaching a floppy drive, a hard drive or by using the on-board networking facility.

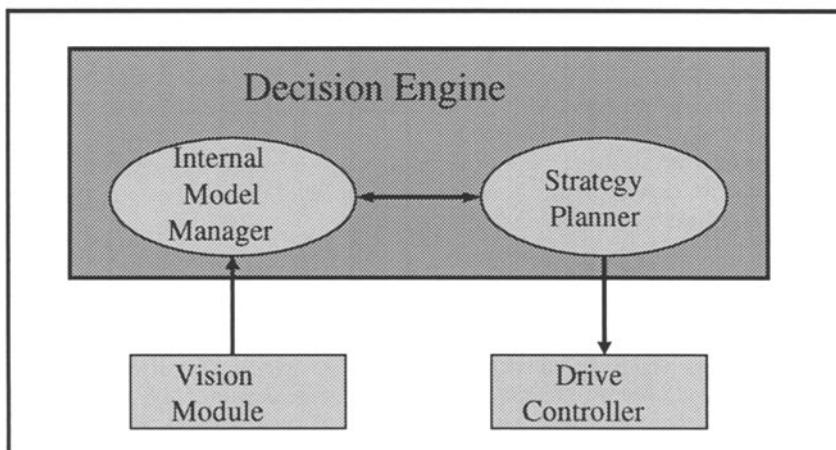


Figure 2. Diagram of the general architecture

The inputs from the on-board camera are collected through a parallel port. Every agent uses the input from the sensors as well as an internal model of the environment and itself to decide its next action. After determining the proper action the robot is steered by commands sent through the I/O ports to the motor controller. Two DC motors, one on each side, provide full motion by quickly turning left or right, and by moving forward or in reverse. Each robot is designed in a modular way, so that we can easily add new software or hardware to extend its working capabilities.

The three main software components of a robot agent are the vision module, decision engine, and drive controller. A diagram showing the interactions between these different modules are shown in Figure 2. The vision module outputs a vector for every frame taken by the agent's on-board camera. Each vector contains the positions of the objects in the frame, such as the ball, players and the goal. This information is then processed by the decision engine. The decision engine is composed of two processing units - the internal model manager and the strategy planner. The model manager takes the vision module's output vectors and maintains an internal representation of the key objects in the soccer field. The strategy planner combines the information that is contained in the internal model with its own strategy knowledge, in order to decide the robot's next action. Once the action has been decided, the command is sent to the drive controller that is in charge of properly executing the command. In the following sections we explain each component in further detail.

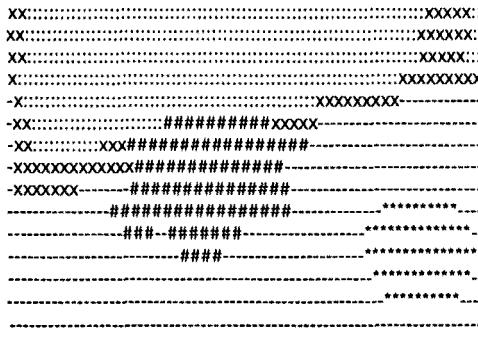


Figure 3. Internal representation of the visual input from Figure 1
(*Numeric color values are represented here as ASCII characters for a better understanding of the picture.*)

3. Vision Module

Just as eyesight is essential to a human player, the visual input to a robot is critical for decision making in a soccer game. A robot relies on its visual input to determine its current position on the field, the positions of other players, the ball, the goals, and the field lines (sidelines, end of field, and penalty area). Each soccer agent receives visual input from an on-board camera. Due to its simplicity, robustness, and software availability, we have chosen Connectix Color PC QuickCam [8] as the on-board visual input hardware. This is a CCD-based camera that outputs 658x496 RGB pixels through the parallel port of the agent's computer. The camera also contains a micro-controller and other electronics to control operation of the camera and pixel data transmission.

To be able to navigate and perform a game plan, our agents depend on visual cues like relative position of the goals, the ball, and other agents. Thus, the agents need to recognize these objects through their visual systems. The output from the QuickCam consists of a series of 3-byte RGB pixels, with each byte representing the intensity of each corresponding color (0 = color absent, 255 = maximum color intensity). The vision software checks each pixel color to see if it may belong to one of the key objects on the field. The pixels are then organized as color clusters to be mapped to identifiable objects. The size and position of each color cluster are used to calculate direction and distance from the actual object on the field. Direction is calculated as an offset of the perceived object with respect to the center of the screen, and distance results from comparing the perceived object size and position within the frame with the known real object size. An example of such an internal representation of objects is shown in Figure 3.

The limitations of the on-board computing power, and the need for a fast, real-time reactive behavior, cooperate to constrain the level of feasible complexity in the algorithms to be run by the agents. Based on the current hardware configuration, visual frames are acquired at a minimum rate of 10 per second. If the processing of a given frame takes more than 0.1 second, the latest frame is processed next, and any intermediate buffered frames are discarded. In this way agents overcome possible lack of synchronization due to variable time needed to process a given frame. It is important to point out that before the agents can correctly recognize objects on the field, diverse lighting conditions and opponent colors mandate an adjustment of the comparison intensity values for each colored object.

2.1. A Vision Calibration Tool

We have developed a calibration tool that can automatically adjust the parameters of the vision system to account for the changes in the robot's surroundings. This vision tool gives the robot the capability to adapt to the new environment. Our adaptive color tracking system uses a supervised learning system based on the inductive learning.

In the first step we take several pictures of the different objects to be recognized in the new environment. The system is attempting to learn the correspondence between the colors and the objects. We repeat the procedure for a given object with a different angle and shadow. In the second step the adaptive color tracking system finds a set of rules to recognize each object, distinguishing which set or class of RGB pixels are associated with a specific object. As the number of colors increase by entering different objects on a game field, the learned rules become more complicated.

RED	GREEN	BLUE	CLASS
240	115	78	Ball
195	95	62	Ball
63	185	90	Field
58	193	115	Field
128	104	213	Goal

Figure 4. Object classification based on RGB

The input for each rule is a combination of Red, Green and Blue intensity for a given pixel in a frame. The output of a rule (the right hand side) would be a class which could be Ball, Wall, Opponent-player, Team-mate, Own-goal and Opponent-goal. As each of these items has a different color in the field. Figure 4 shows an example of input data and a simple rule to differentiate between Red (Ball), Green (Field) and Blue (Goal).

One of the common methods in knowledge discovery and intelligent data analysis is induction. Tree induction methods produce decision trees, concerning the data set as a result of their classification process. To find a valid interpretation in our large frame database we can employ any decision tree algorithm to find the needed association rules between the frame color patterns and the class of objects. In this approach we applied C4.5 [9] as the main induction method. C4.5 package generates decision trees, which provides a classification for every object within the database. The package allows the decision tree to be simplified, using a pruning technique which reduces the size of the tree according to a user-defined confidence level.

Once the vision system has been calibrated to the new environment, it can then process the necessary information about the objects that are perceived in the camera frames. This information (the size and position of the object in the frame) is then sent in the form of object or frame vectors to the decision engine, where it is used to decide the actions of the robot.

3. Decision Engine

This component makes decisions about the actions of an agent. It receives input from the sensor modules and sends move commands to the drive controller. The decision engine bases its decisions on a combination of the received sensor input, the agent's internal model of its environment, and knowledge about the agent's strategies and goals. The agent's internal model and strategies are influenced by the role the agent plays on the soccer field. There are three types of agent roles or playing positions: goalkeeper, defender, and forward. The team strategy is broken down into pieces and instilled in the role strategies of each of the agents. Depending on the role type, an agent can be more concerned about a particular area or object on the soccer field, e.g. a goalkeeper is more concerned about its own goal, while the forward is interested in the opponent's goal. These differences are encoded into the two modules that deal with the internal model and the agent's strategies.

Together, the internal model manager and the strategy planner, form the decision engine. These sub components of the decision engine communicate with each other to formulate the best decision for the agent's next action. The model manager converts the vision module's frame vectors into a map of the agent's current environment, as well as generating a set of object movement predictions. It calculates the salient features and then communicates them to the strategy planner. To calculate the best action, the strategy planner uses both the information from the model manager and the strategy knowledge that it has about the agent's role on the field. It then sends this information to the drive controller and back to the model manager, so that the internal model can be properly updated.

3.1. Model Manager

Because the soccer agents need to know about their environment and themselves before taking an appropriate action, an internal model is used to address this problem. The model manager is responsible for building and maintaining the internal model that provides this information. The internal model contains a map of the soccer field and location vectors for nearby objects. The object vectors from the vision module are used to create the location vectors for the internal model.

A location vector consists of four calculations or elements; distance and direction to the object and the change in distance and direction for the object. The change in distance and direction is used for predicting the object's new location. The model manager continuously receives frame vectors from the vision module and updates the location vectors accordingly.

The model manager keeps track of information based on the role of the robot on the soccer field. For a goalkeeper the information will include the location vectors for the goal, ball and opponent's forwards. This is necessary for not overloading the strategy planner with extra information.

An internal model is necessary for several reasons. Since the visual information provided by the vision module is incomplete, the robot can see only the objects that are within its current visual frame. More information about the environment can be deduced by using a map of the soccer field and the historical data. For example, if the robot can see and calculate the distance to the left wall of the field, it can find out the distance to the right wall even when the right wall is not visible. Similarly, the approximate location of an opponent that was previously visible, but currently not in view can be calculated using the historical data from

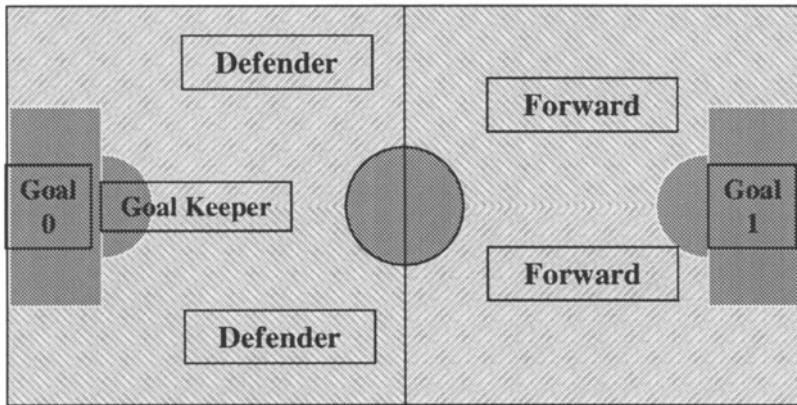


Figure 5. Playing positions of agents on the Soccer field

the previous frames. This also provides greater robustness for the robot. If the camera fails for a few cycles (e.g. due to a hit or being blocked etc.), the robot can still operate using its internal model of the environment.

An internal model is also necessary for predicting the environment. This is important for the strategy planner. For example, to intercept the ball the robot needs the current location of the ball and also a prediction of the current heading of the ball, so that it can calculate an intercept course to the ball. The internal model is also used for providing feedback to the strategy planner to enhance and correct its actions. The strategy planner tells the model manager what actions will be taken and the model manager updates the internal model using this information. It then receives information from the vision module and compares it with the expectations contained in the model. Any discrepancies are reported to the strategy planner, so that it can then use this information to fine tune its operations.

3.2. Strategy Planner

In order to play a successfully soccer game, each robot must react appropriately to different situations in the field. This is accomplished by the strategy planner that resides in the decision engine on each robot. Since a strategy (or a policy) is a mapping from situations to actions, let us first define situations and actions for our robots. As we mentioned in the description of model manager, objects in the soccer field are represented by their relative positions to the observer. Internally, an object is represented by a location vector of four elements: the distance between the object and the observer, the direction of the object, and change in distance and

direction. The values of the first two elements are qualitative: the distance can have values near, medium, far, and ? (unknown). The direction can have values left, center, right, and ? (unknown). To increase the response time of our decision process, not all objects in the field are constantly tracked, instead only those that are currently in the visual field of the observer. (Those objects that are currently not in view have ? as their values in the location vector.)

Based on this terminology, we define a situation as a set of observed location vectors in the field. For example, if a forward player is facing the opponent's goal and between the goal and the ball (see Figure 5), it is represented as the following vector:

$$\left. \begin{array}{l} \text{Ball: } <?, ?, ?, ?>, \\ \text{Goal0: } <5, 11, ?, ?>, \\ \dots \end{array} \right\}$$

The basic actions of robots are the five commands to the motors; they are move-forward, move-backward, stop, turn-left and turn-right. Based on these basic actions, a set of compound actions or behaviors is then defined. The set of compound actions includes kick, line-up, intercept, homing, and detour. Some brief descriptions of these actions are shown in Figure 6.

KICK:	kick the ball
LINE-UP:	move to line up the ball and the goal.
INTERCEPT:	calculate intercept path to the ball
HOMING:	go back to its "home" position
DETOUR:	go around the ball

Figure 6. Examples of compound action

Each action has a termination condition and a time threshold. For example, the termination condition of line-up is when the robot is behind the ball and the goal, the ball, and itself are on the same line. In case actions are not successfully terminated, they will time-out as soon as the duration of the action passes the time threshold.

Based on the defined situations and actions, the task of strategy planner is to select the right action for a given situation. This decision process is captured by a Policy Table shown in Figure 7. As we can see from this table, each line is a rule that maps a situation to an action. For example, if the position of ball is unknown in the current situation, the action is to turn left for 30 degree in search of the ball.

Situation	Action				
Ball	Goal0	Goal1	Wall	Player	X
<?,?>	<.,.>	<.,.>	<.,.>	<.,.>	Turn Left(30)
<x, y>	<.,.>	<.,.>	<.,.>	<.,.>	LINE-UP
...

Figure 7. A Policy Table

There are five positions a robot can play on our soccer team: left-forward, right-forward, left-defender, right-defender, and goalkeeper. Shown in Figure 5, each player has its own territory and home position. For example, the left-forward has the territory of the left-forward quarter of the field, and its home position is near the center line and roughly 1.5 meter from the left board line. (The territory of the home position for the right-forward is symmetric to that of the left-forward). Similarly, the left-defender is in charge of the left-back quarter of the field and its home position is at the left front of the base goal. Given this configuration, we now describe the policy tables for each of the three types of robots: goalkeeper, forward and defender.

Goalkeeper

For the goalkeeper, the two most important objects in the field are the ball and its own goal. Its home position is in front of the goal, and its policy is to keep itself in the line of the ball and the goal. Since most of its actions are parallel to the base line, the goalkeeper's camera is mounted on the side (for all other robots, the camera is mounted in the front), so that it can move sideways while keeping an eye on the ball. Its policy table is as follows:

Situation	→	Action
Ball = <near, left>, Goal0 = <near, center>		Move-left
Ball = <near, right>, Goal0 = <near, center>		Move-right
Ball = <far, _>		Homing
Ball = <?, _>, Goal0 = <near, left>		Turn-right
Ball = <?, _>, Goal0 = <near, right>		Turn-left
.....	

Figure 8. The *goalkeeper* policy table

As we can see, the first two actions are to prevent the ball from entering the goal, the third action is to position itself, and the last two actions are to look for the ball. Note that although policy tables are easy to describe conceptually, their implementation in the real system require much engineering to make them correct and robust.

Forward

The task of forward is to put the ball into opponent's goal whenever possible. Like the goalkeeper, it must look for the ball when it is not in sight, and head back to its home position when the ball is out of its territory. The main difference is that the forward's main interest is to kick the ball towards the opponent's goal (Goal1 in our current example) whenever it can. So its policy table, shown in Figure 9, must reflect that.

Note that as long as the ball and Goal1 are both in sight, the forward will kick the ball. The policy also tells the robot to search for the ball whenever it is not in sight (e.g., seeing only the wall). It returns to its home position if it sees the opponent goal but not the ball. Whenever it sees the ball and its own goal (Goal0), it must make a detour, so that it can kick the ball in the correct direction.

Situation	→	Action
Ball = <near, center>, Goal1 = <., ~?>		Kick
Ball = <?, _>, Goal1 = <., ~?>		Homing
Ball = <?, _>, Goal1 = <_, left>		Turn-right
Ball = <?, _>, Goal1 = <_, right>		Turn-left
Ball = <?, _>, Wall = <near, ?>		Turn-left
Ball = <near, _>, Goal0 = <far, ?>		Detour
.....	

Figure 9. The *forward* policy table

Defender

The defender's policy is very similar to that of the forward, except that the distance to Goal1 is further away compared to the position of the forward. Similar to the goalkeeper, it also tries to position itself between the ball and its own goal (Goal0).

4. Drive Controller

As mentioned before, each robot consists of a single 586 based computer on a 30cm x 50cm, 4-wheel drive, DC model car (Figure 1). The inputs from the on-board camera are collected through parallel and serial ports. The drive controller takes commands from the decision engine, and steers the robots through its I/O ports. Four I/O ports are connected to the DC motor drivers and steer the robot by quickly turning left or right, and by moving forward or in reverse. The drive controller stops the motion by disconnecting the move order. The car also has the capability to spin in place by turning the two sides of wheel into opposite direction. In order to make movements that are as precise as possible, the effects of actions are controlled by the amount of time of a command. To deal with much of the uncertainty associated with the system, closed-loop control in software is used whenever possible.

5. Related Work

The current research follows the original work from a prediction-based architecture called LIVE [5] for integrating learning, planning and action in autonomous systems [4], and a behavior-based robot control system [1,2]. Compared to similar architectures in recent agent literature (for example [7]), our approach uses the closed loop control theory in many aspects of robot building, and uses the internal model to help detect and recover from errors. Moreover, our robot agent can also be configured quickly into different soccer playing roles and that has greatly increased the flexibility of the entire robot team.

Our philosophy on multi-agent collaboration [6] is that if each agent has a sufficient understanding of other agents' action, then collaboration can be accomplished without any explicit communication. In our team, there is an implicit form of collaboration in that each agent has a particular role in which it knows its function and behaviors, as well as those of its teammates. Indeed, some limited collaboration behavior has been observed during the competition, and additional evidence of this hypothesis can also be found in the description of the champion team in the simulated league in this volume.

6. Conclusion and Future Work

Conventionally, agents in the real world are controlled and guided by an external supervisor when performing complex tasks. In this study, we propose an autonomous system model for soccer robots that do not use any external computation resources, information, or guidance. Our soccer agents are designed with simple structures, but demonstrate that with an acceptable strategy and system modularity they can achieve the goal of autonomous behavior in a context of teamwork. These autonomous soccer robots use on-board cameras as their own sensors, and decide appropriate actions based on-board computers. Three main features seem to contribute the most to this success. First, the model manager is responsible for translating and interpreting the vision information in order to obtain the knowledge necessary for the strategy planner. Second, the closed-looped control mechanism deals with many intrinsic uncertainties in the soccer domain. Third, the modularity of hardware and software design of the system has made it possible for a single architecture to play multiple roles.

In our future work, we would like to improve the quality of autonomous sensing and acting, as well as extend the collaborations between robots. In particular, we would like to add more and heterogeneous sensors, such as sonar and others to speed up an agent's decision making and increase their efficiency and accuracy. We would also like to extend our system to use explicit communication for agent collaboration.

7. Acknowledgements

We would like to thank USC/ISI, especially people in the Division of Intelligent System, for their generous moral and financial support for the Dreamteam. We are also grateful for the international travel support provided by Japan Airline, and an Omnibook 600 laptop computer provided by Hewlett Packard for earlier experiments on the vision and control systems. Finally, thanks to Patrick Reynolds from the University of Virginia for the free software package *cqcam* for using QuickCam.

References

1. Arbib, M. Perceptual Structures and Distributed Motor Control. In *Handbook of Physiology -- The Nervous System*, II, ed. V. B. Brooks. American Physiological Society, 1981.
2. Arkin, R.C. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robotics Research* (1987) 92-112.
3. Brooks, R. A. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation* 2(1), 1986.
4. Shen, W.M. *Autonomous Learning From Environment*. Computer Science Press, W.H. Freeman, New York. 1994.
5. Shen, W. M. LIVE: An Architecture for Autonomous Learning from the Environment *ACM SIGART Bulletin (Special Issue on Integrated Cognitive Architectures)* 2(4), 151-155. 1991.
6. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.. Robocup: The Robot World Cup Initiative. In *Proceeding of the first International Conference on Autonomous Agents*. Marina del Rey, CA, 1997.
7. Garcia-Alegre, M.C., Recio, F.. Basic Agents for Visual/Motor Coordination of a Mobile Robot. In *Proceeding of the first International Conference on Autonomous Agents*. Marina del Rey, CA, 1997.
8. Connectix Corporation, Connectix Color QuickCam, May 1996.
9. Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

Vision-Based Robot Learning Towards RoboCup: Osaka University “Trackies”

S. Suzuki¹, Y. Takahashi², E. Uchibe², M. Nakamura²,
C. Mishima¹, H. Ishizuka², T. Kato², and M. Asada¹

¹ Dept. of Adaptive Machine Systems, Osaka University,
Yamadaoka 2-1, Suita, Osaka 5650871, Japan.

² Dept. of Computer-Controlled Machinery, Osaka University,
Yamadaoka 2-1, Suita, Osaka 5650871, Japan.

Abstract. The authors have applied reinforcement learning methods to real robot tasks in several aspects. We selected a skill of soccer as a task for a vision-based mobile robot. In this paper, we explain two of our method; (1)learning a shooting behavior, and (2)learning a shooting with avoiding an opponent. These behaviors were obtained by a robot in simulation and tested in a real environment in RoboCup-97. We discuss current limitations and future work along with the results of RoboCup-97.

1 Introduction

Building robots that learn to perform a task in a real world has been acknowledged as one of the major challenges facing AI and Robotics. Reinforcement learning has recently been receiving increased attention as a method for robot learning with little or no *a priori* knowledge and higher capability of reactive and adaptive behaviors [3]. In the reinforcement learning scheme, a robot and an environment are modeled by two synchronized finite state automata interacting in discrete time cyclical processes. The robot senses the current state of the environment and selects an action. Based on the state and the action, the environment makes a transition to a new state and generates a reward that is passed back to the robot. Through these interactions, the robot learns a purposive behavior to achieve a given goal.

As a testbed to apply the reinforcement learning method for real robot tasks, we have selected soccer playing robots [1]. We have been doing various kinds of research topics as follows;

1. learning a shooting behavior in a simple environment [11]
2. learning a coordinated behavior of shooting and avoiding an opponent [12][15]
3. self construction of a state space [13]
4. learning of a real robot in a real environment [14]
5. modeling other agents [16]

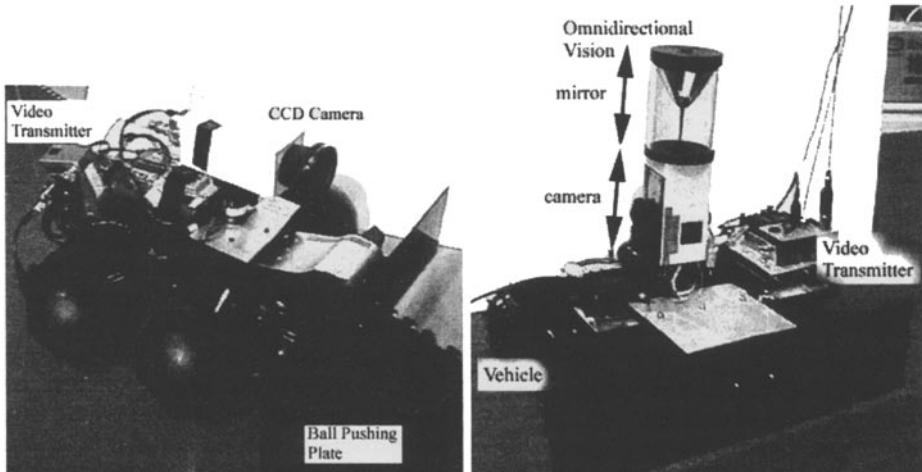
Two methods ([11] and [15]) are tested in RoboCup-97 in which robots take actions based on the learned policy that has not include cooperation between teammate robots yet this year.

In this paper, we summarize our research issues involved in realizing a robot team for RoboCup-97. This article is structured as follows: In section 2, we explain the configuration of our robot system. In section 3, we give a brief overview of Q-learning. In section 4, we explain acquisition of shooting behavior. In section 5, we explain acquisition of a coordinated behavior combined shooting and avoiding an opponent. In section 6, we describe the result. Finally, we give a conclusion.

2 The Configuration of the Robot System

We have decided to use a radio-controlled model car as a robot body and to control it based on the remote brain approach [9]. This makes us implement and monitor the system activities easy.

In RoboCup-97, we participated with five robots consisting four attackers and one goalie (see Figure 1). In this section, we explain the hardware and the control architecture for our robots.



(a) The attacker robot

(b) The goalie robot

Fig. 1. Our Robots

2.1 Hardware of the Robots

We use radio-controlled model cars with a PWS (Power Wheeled Steering) locomotion system. Four of them are called “Black Beast” produced by Nikko as an attacker robot (see Figure 1(a)), and one called “Blizzard” produced by Kyosho as a goalie (see Figure 1(b)). A plate is attached to push the ball on the field. The attacker has the plate in front of the robot and the goalie has on its side. The robots are controlled by signal generated on the remote computer through the radio link.

Each robot has a single color CCD camera for sensing the environment and a video transmitter. The attacker robot has a SONY CCD camera with a wide lens while the goalie has an omnidirectional vision system [10] so that it can see the goal and the ball coming in any direction at the same time. The image taken by the camera is transmitted to the remote computer and processed on it.

For power supply, three Tamiya 1400NP batteries are mounted on the robot. Two drive two motors for locomotion, and the remaining one supplies 12V through a DC-DC converter to drive the camera and the transmitter. The life of the battery is about 20 minutes for locomotion and 60 minutes for the camera and the transmitter.

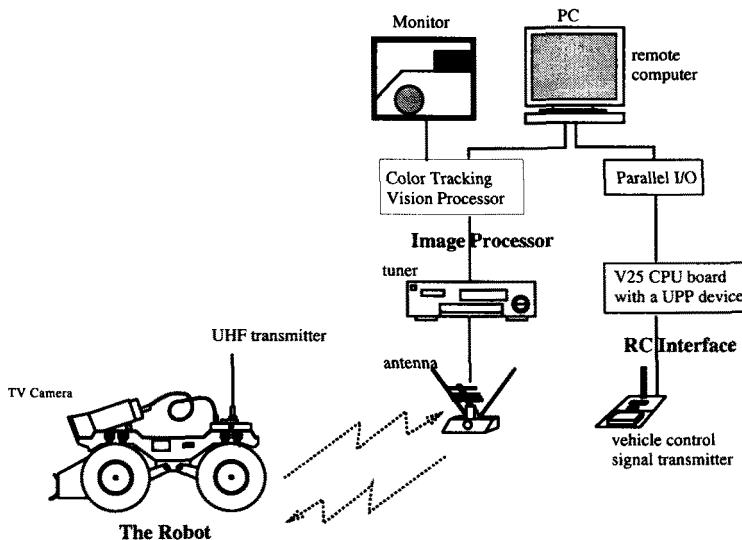


Fig. 2. Configuration of robot controller

2.2 The Control Architecture

The controller of each robot consists of three parts; a remote computer, an image processor, and a radio-control interface (RC interface). Figure 2 shows a

configuration of the controller in which PC is used as the remote computer.

The action of the robot is controlled by the following steps:

1. the robot transmits the image from its camera,
2. the image processor receives the image through UHF and processes it,
3. the remote computer decides the robot's action based on the result of image processing,
4. the RC interface generates a signal corresponding to the decided action, and
5. the robot receives the signal and drives its motors.

We use a color tracking vision board produced by Fujitsu for the image processing, and a UPP device to generate the control signal. Objects in the environment (a ball, a goal, and an opponent) are detected as colored regions in the image according to RoboCup regulations.

3 Q-learning for Robot Learning

In the reinforcement learning scheme, the robot senses the current state of the environment and selects an action. Based on the state and the action, the environment makes a transition to a new state and generates a reward that is passed back to the robot. Through these interactions, the robot learns a purposive behavior to perform a given task (see Figure 3). As a method for reinforcement learning, we adapted Q-learning that is one of most widely used reinforcement learning method. In this section, we give a brief overview of Q-learning and problems when we apply it to real robot tasks.



Fig. 3. Interaction between the robot and the environment

3.1 Basics of Q-learning

We assume that the robot can discriminate the set S of distinct environment states, and can take the set A of actions on the environment. The environment is modeled as a Markov process, making stochastic transitions based on its current state and the action taken by the robot. Let $T(s, a, s')$ be the probability of transition to the state s' from the current state-action pair (s, a) . For each state-action pair (s, a) , the *reward* $r(s, a)$ is defined.

Given the definitions of the transition probabilities and the reward distribution, we can solve for the optimal policy(a policy f is a mapping from S to A), using methods from dynamic programming [2]. A more interesting case occurs when we wish to simultaneously learn the dynamics of the environment and construct the policy. Watkin's Q-learning algorithm gives us an elegant method for doing this [6].

Let $Q^*(s, a)$ be the expected *action-value function* for taking action a in a situation s and continuing thereafter with the optimal policy. It can be recursively defined as:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a' \in A} Q^*(s', a'). \quad (1)$$

Because we do not know T and r initially, we construct incremental estimates of the Q -values on-line. Starting with $Q(s, a)$ equal to an arbitrary value (usually 0), every time an action is taken, the Q -value is updated as follows:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r(s, a) + \gamma \max_{a' \in A} Q(s', a')). \quad (2)$$

where r is the actual reward value received for taking action a in a situation s , s' is the next state, and α is a learning rate (between 0 and 1).

3.2 Problems in Applying Q-learning to Real Robot Tasks

To apply Q-learning, we must cope with several problems which occur in real environments. Two major problems are construction of state and action sets, and reduction of learning time [11].

Construction of State and Action Sets In the environment where the robot exist, everything changes asynchronously. Thus traditional notions of state in the existing applications of the reinforcement learning algorithms dose not fit nicely [5]. The following principles should be considered for the construction of state and action spaces.

- Natural segmentation of the state and action spaces: The state (action) space should reflect the corresponding physical space in which a state (an action) can be perceived (taken).

- Real-time vision system: Physical phenomena happen continuously in the real environment. Therefore, the sensor system should monitor the changes of the environment in real time. This means that the visual information should be processed in video frame rate (33ms).

The state and action spaces are not discrete but continuous in the real environment, therefore it is difficult to construct the state and action spaces in which one action always corresponds to one state transition. We call this “**state-action deviation problem**” as a kind of the so-called “perceptual aliasing problems” [7] (i.e., a problem caused by multiple projections of different actual situations into one observed state). The perceptual aliasing problem makes it very difficult for a robot to take an optimal action. The state and action spaces should be defined considering this state-action deviation problem.

Reduction of Learning Time This is the famous *delayed reinforcement* problem due to no explicit teacher signal that indicates the correct output at each time step. To avoid this difficulty, we construct the learning schedule such that the robot can learn in easy situations at the early stages and later on learn in more difficult situations. We call this *Learning from Easy Missions* (or LEM).

4 Learning a Shooting Behavior

For the first stage, we set up a simple task for a robot [11], to shoot a ball into a goal as shown in Figure 4. We assume that the environment consists of a ball and a goal. The ball is painted in red and the goal in blue so that the robot can detect them easily. In this section, we describe a method for learning the shooting behavior with consideration of the problem mentioned in section 3. Here we focus on the method implemented on the attacker robot in RoboCup-97 (see [11] for more detail).

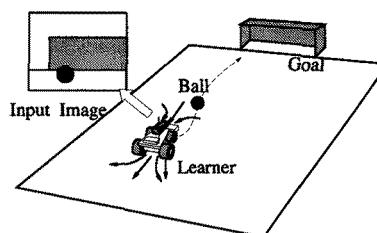


Fig. 4. The task is to shoot a ball into a goal

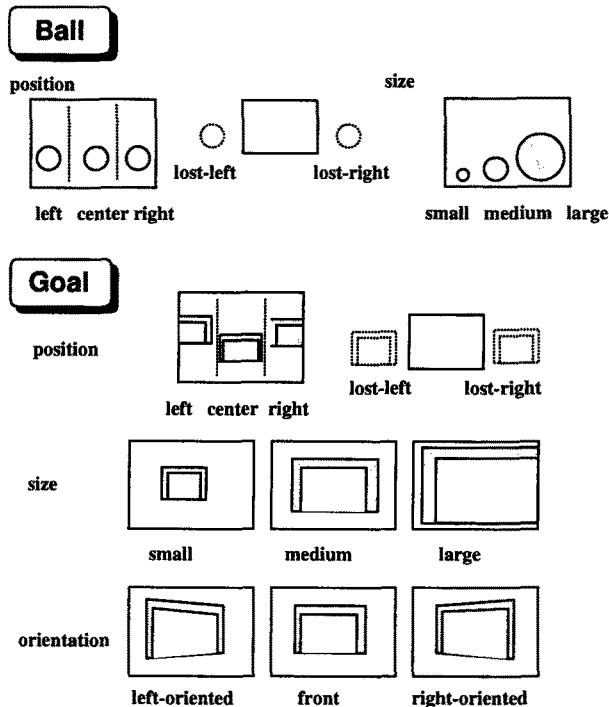


Fig. 5. The ball substates and the goal substates

4.1 Construction of Each Space

(a) a state set S : The ball image is classified into 9 substates, combinations of three classifications of positions (left, center, or right) and three types of sizes (large (near), middle, or small (far)). In addition to the size and the positions, the goal image has 27 substates considering the orientation which is also classified into three categories (see Figure 5). Each substate corresponds to one posture of the robot towards the goal, that is, the position and the orientation of the robot in the field.

In addition, we define states for the cases in which the ball or the goal is not captured in the image: three states (ball-unseen, ball-lost-into-right, and ball-lost-into-left) for the ball, and three more states (goal-unseen, goal-lost-into-right and goal-lost-into-left) for the goal. In all, we define 12 (9 + 3) states for the ball and 30 (27 + 3) states for the goal, and therefore the set of states S is defined with 360 (12 × 30) states.

(b) an action set A : The robot can select an action to be taken in the current state of the environment. The robot moves around using a PWS (Power Wheeled

Steering) system with two independent motors. Since we can send the motor control command ω_l and ω_r to each of the two motors separately, each of which has forward, stop, and back, we have nine action primitives all together.

We define the action set A as follows to avoid the state-action deviation problem. The robot continues to take one action primitive at a time until the current state changes. This sequence of the action primitives is called an action.

(c) a reward and a discounting factor γ : We assign the reward value to be 1 when the ball is kicked into the goal and 0 otherwise. This makes the learning very time-consuming. Although adopting a reward function in terms of distance to the goal state makes the learning time much shorter in this case, it seems difficult to avoid the local maxima of the action-value function Q .

A discounting factor γ is used to control to what degree rewards in the distant future affect the total value of a policy. In our case, we set the value at slightly less than 1 ($\gamma = 0.8$).

4.2 Simulation

We performed the computer simulation. Figure 6 shows some kinds of behaviors obtained by our method. In (a), the robot started at a position from where it could not view a ball and a goal, then found the ball by turning, dribbled it towards the goal, and finally shot the ball into the goal. This is just a result of learning. We did not decompose the whole task into these three tasks. The difference in the character of robot player due to the discounting factor γ is shown in (b) and (c) in which the robot started from the same position. In the former, the robot takes many steps in order to ensure the success of shooting because of a small discount, while in the latter the robot tries to shoot a ball immediately because of a large discount. In the following experiments, we used the average value of γ 0.8 as an appropriate discount.

We applied the LEM algorithm to the task in which S_i ($i=1,2$, and 3) correspond to the state sets of “the goal is large”, “medium”, and “small”, respectively, regardless of the orientation and the position of the goal, and the size and position of the ball. Figure 7 shows the changes of the summations of Q -values with and without LEM, and ΔQ . The axis of time step is scaled by M (10^6), which corresponds to about 9 hours in the real environment since one time step is 33ms. The solid and broken lines indicate the summations of the maximum value of Q in terms of action in states $\in S_1 + S_2 + S_3$ with and without LEM, respectively. The Q-learning without LEM was implemented by setting initial positions of the robot at completely arbitrary ones. Evidently, the Q-learning with LEM is much better than that without LEM.

The broken line with “ \times ” indicates the change of $\Delta Q(S_1 + S_2 + S_3, a)$. Two arrows indicate the time steps (around 1.5M and 4.7M) when a set of the initial states changed from S_1 to S_2 and from S_2 to S_3 , respectively. Just after these steps, ΔQ drastically increased, which means the Q -values in the inexperienced states are updated. The coarsely and finely dotted lines expanding from the time

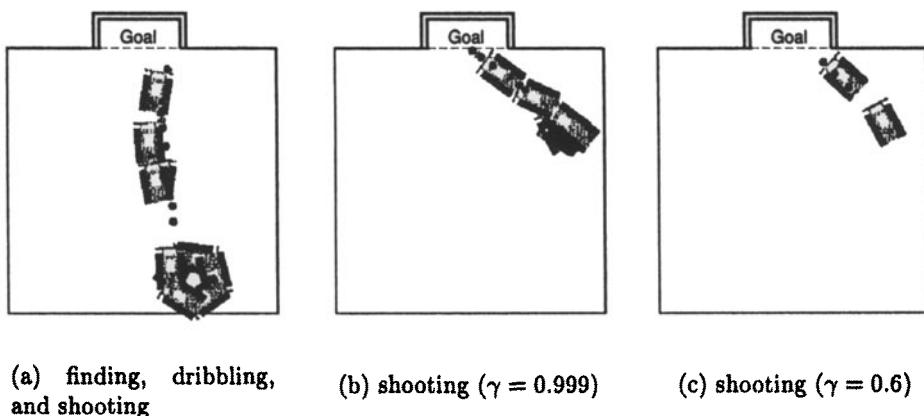


Fig. 6. Some kinds of behaviors obtained by the method

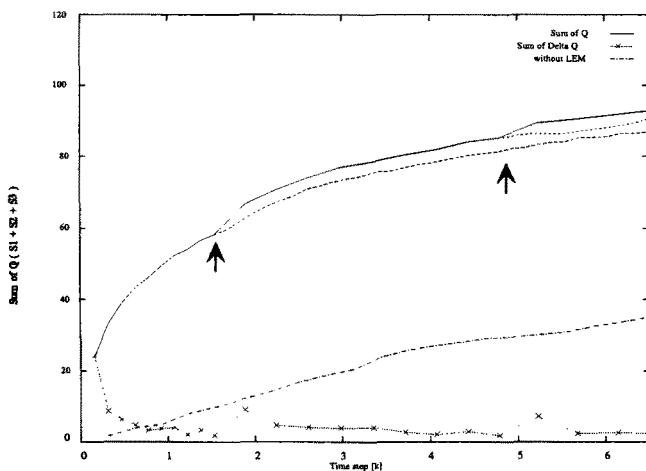


Fig. 7. Change of the sum of Q-values with LEM in terms of goal size

steps indicated by the two arrows show the curves when the initial positions were not changed from S_1 to S_2 , nor from S_2 to S_3 , respectively. This simulates the LEM with partial knowledge. If we know only the easy situations (S_1), and nothing more, the learning curve follows the finely dotted line in Figure 7. The summation of Q-values is slightly less than that of the LEM with more knowledge, but much better than that without LEM.

5 Shooting a Ball with Avoiding an Opponent

In the second stage, we set up an opponent just before the goal and make the robot learn to shoot a ball into a goal avoiding the opponent (see Figure 8). This task can be considered as a combination of two subtasks; a shooting behavior and an avoiding behavior of an opponent. The basic idea is first to obtain the desired behavior for each subtask, and then to coordinate two learned behaviors. In this section we focus on the coordination method implemented on the attacker robot in RoboCup-97, see [12] and [15] for more detail.

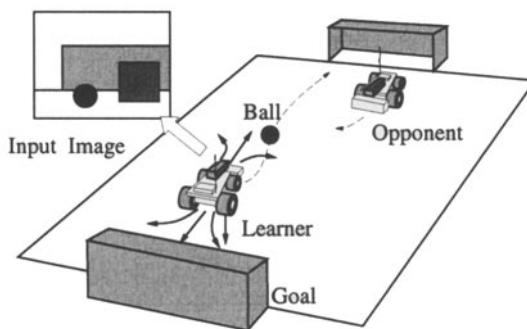


Fig. 8. The task is to shoot a ball into the goal avoiding an opponent.

5.1 Learning a Task from Previously Learned Subtasks

The time needed to acquire an optimal policy mainly depends on the size of state space. If we apply the monolithic Q learning into a complex task, the expected learning time is exponential in the size of state space [8]. One technique to reduce learning time is to divide the task into some subtasks and to coordinate behaviors which is independently acquired. The simple coordination method is summation or switching of the previously learned action value functions.

However, these methods cannot cope with local maxima and/or hidden states caused by direct product of individual state spaces corresponding to the subtasks. Consequently, an action suitable for these situations has never been learned. To cope with these new situations, the robot needs to learn a new behavior by using the previously learned behaviors [12]. The method is as follows:

1. Construct a new state space S :
 - (a) construct the directly combined state space from subtasks' state s_1 and s_2
 - (b) find such states that are inconsistent with s_1 or s_2
 - (c) resolve the inconsistent states by adding new substates $s_{sub} \in S$.

2. Learn a new behavior in the new state space S :
- calculate the value of the action value function Q_{ss} by simple summation of the action value functions of each subtasks.

$$Q_{ss} = \max_{a \in A} (Q_1((s_1, *), a) + Q_2((*, s_2), a)) \quad (3)$$

where $Q_1((s_1, *), a)$ and $Q_2((*, s_2), a)$ denote the extended action value functions. $*$ means any states, therefore each of these functions considers only the original states and ignores the states of other behaviors.

- initialize the value of the action value function Q for the normal states s and the new substates s_{sub} with Q_{ss} . That is,

$$\begin{aligned} Q(s, a) &= Q_{ss}(s, a) \\ Q(s_{sub}, a) &= \text{original value of } Q_{ss}(s, a) \end{aligned} \quad (4)$$

- control the strategy for the action selection in such a way that a conservative strategy is used around the normal states s and a high random strategy around the new substates s_{sub} in order to reduce the learning time.

For the first subtask (shooting behavior), we have already obtained the policy by using the state space shown in Figure 5. For the second subtask (avoiding behavior), we defined the substates for the opponent in the same manner to the substate of the ball in Figure 5. That is, a combination of the position (left, center, and right) and the size (small, medium, and large) is used.

A typical example of inconsistent states is the case where the ball and the opponent are located at the same area and the ball is occluded by the opponent from the viewpoint of the robot. In this case, the robot cannot observe the ball, and therefore the corresponding state for shooting behavior might be the state of “ball-lost,” but it is not correct. Of course, if both the ball and the opponent can be observed, this situation can be considered consistent. This problem is resolved by adding new substates $s_{sub} \in S$. In the above example, a new situation “occluded” is found by estimating the current state from the previous state, and the corresponding new substates are generated (see [12] for more detail).

5.2 Simulation

Based on the LEM algorithm, we limit the opponent’s behavior when the robot learns. If the opponent has learned the professional techniques to keep the goal, the robot might not be able to learn how to shoot the ball into the goal anymore because of almost no goals. From this viewpoint, the opponent’s behavior is scheduled so that the shooting robot has its confidence to shoot a ball into the goal.

In the simulation the robot has succeeded to acquire a behavior for a shooting the ball into the goal (see Figure 9). In the figure, the black is the learner and the white is the opponent. In (a), the robot watches the ball and the opponent. In (b),(c), and (d), the robot avoids the opponent and moves toward the ball. In (e) and (f), the robot shoots the ball into the goal.

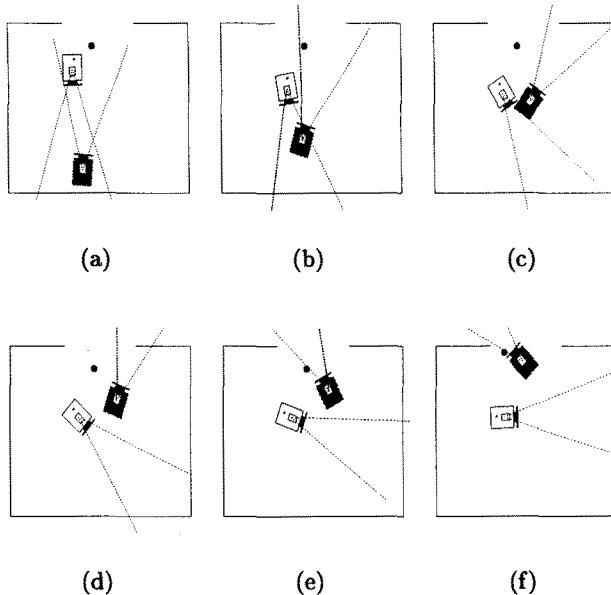


Fig. 9. The robot succeeded in shooting a ball into the goal

6 Experimental Result in RoboCup-97

We participated the middle size robot league of RoboCup-97 with five robots: four attackers and one goalie. For the goalie, we defined a set of rules and implemented on it as a goal keeping behavior. For the attackers, we implemented the behavior obtained by the simulation described in section 4.2 and 5.2.

Our team had five matches in total; two preliminary, two exhibition matches and the final. The result is shown in Table 1. Figure 10 and Figure 11(a) show a scene of a match, in which an attacker shoots the ball and the goalie keeps the goal respectively. Figure 11(b) is the view of the goalie in the situation of Figure 11(a). Our robot could get two goals in total, because four of two goals were own goals by the opponent team (USC).

7 Conclusions

In this paper, we have explained two of our reinforcement learning methods applied for real robot tasks tested in RoboCup-97. Our robots had learned a shooting behavior and a shooting behavior with avoiding an opponent, and played five matches there. They got two goals during more than 50 minutes of total playing time (time of one match was 10 minutes).

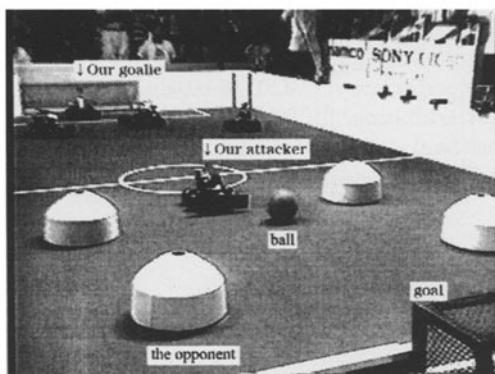
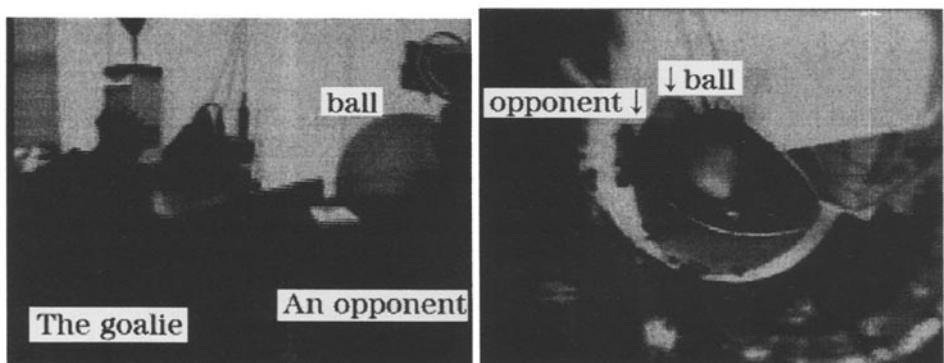


Fig. 10. One attacker shoots the ball



(a) The goalie and an opponent

(b) The view of the goalie

Fig. 11. A behavior of the goalie

We are difficult to say that the robot performed the task well. However, getting two goals means that the robot could performed the task when it met a certain situation. This fact shows a potential ability of reinforcement learning methods to make the robot adapt to the real environment.

There are some reasons why the performance was not good enough. We had a trouble with color recognition because of noise on image transmission and uneven lighting condition on the field. Especially there were a plenty of noise sources around the field and the image became black and white so often. Though these problems are beyond the scope of our research issue, treatment of these

date	match	opponent team	score		result
25 August	preliminary	RMIT Raiders	0-1	us	win
26 August	preliminary	USC Dreamteam	2-2	us	draw
27 August	exhibition	UTTORI United	0-1	us	win
28 August	final	USC Dreamteam	0-0	us	draw
28 August	exhibition	The Spirit of Bolivia	1-0	us	lose

Table 1. The Result of matches

problems will improve the performance of the task.

A problem of our methods was construction of the state space. We ignored the case when the robot watches several robots in its view at a time, though nearly 10 robots existed on the field in every matches. In our future work, we need to focus state construction in a multi robot environment. Some topics have been already started, such as self construction of states by the robot [13],[14] and estimation and prediction of an opponent's behavior [16].

References

1. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: RoboCup: A Challenge Problem of AI. *AI Magazine* **18** (1997) 73–85
2. Bellman, R.: *Dynamic Programming*. Princeton University Press (1957)
3. Connel, J. H., Mahadevan, S.: *Robot Learning*. Kluwer Academic Publishers (1993)
4. Kaelbling, L. P.: Learning to Achieve Goals. *Proc. of IJCAI-93* (1993) 1094–1098
5. Mataric, M.: Reward Functions for Accelerated Learning. In *Proc. of Conf. on Machine Learning-1994* (1994) 181–189
6. Watkins, C. J. C. H., Dayan, P.: Technical note: Q-learning. *Machine Learning* **8** (1992) 279–292
7. Whitehead, S. D., Ballard, D. H.: Active Perception and Reinforcement Learning. In *Proc. of Workshop on Machine Learning-1990* (1990) 179–188
8. Whitehead, S. D.: Complexity and Coordination in Q-Learning, In *Proc. of the 8th International Workshop on Machine Learning* (1991) 363–367
9. Inaba, M.: Remote-Brained Robotics: Interfacing AI with Real World Behaviors. In *Preprints of ISRR'93* (1993)
10. Yagi, Y., Kawato, S.: Panoramic Scene Analysis with Conic Projection. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems* (1990)
11. Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. *Machine Learning* **23** (1996) 279–303
12. Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., Hosoda, K.: Coordination Of Multiple Behaviors Acquired By A Vision-Based Reinforcement Learning. *Proc. of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems* (1994) 917–924

13. Asada, M., Noda, S., Tawarayamida, S., Hosoda, K.: Vision-Based Reinforcement Learning for Purposive Behavior Acquisition. Proc. of the IEEE Int. Conf. on Robotics and Automation (1995) 146–153
14. Takahashi, Y., Asada, M., Noda, S., Hosoda, K.: Sensor Space Segmentation for Mobile Robot Learning. Proceedings of ICMAS'96 Workshop on Learning, Interaction and Organizations in Multiagent Environment (1996)
15. Uchibe, E., Asada, M., Hosoda, K.: Behavior Coordination for a Mobile Robot Using Modular Reinforcement Learning. Proc. of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems (1996) 1329 – 1336
16. Uchibe, E., Asada, M., Hosoda, K.: Vision Based State Space Construction for Learning Mobile Robots in Multi Agent Environments. Proc. of Sixth European Workshop on Learning Robots(EWLR-6) (1997) 33–41

RoboCup97: An Omnidirectional Perspective

Content Areas: Omnidirectional Motion, Lightweight Robots

Andrew Price, Andrew Jennings, John Kneen
andrew.price@cse.rmit.edu.au

Department of Computer Systems Engineering
Royal Melbourne Institute of Technology
Melbourne Victoria Australia.

Abstract

RoboCup-97 proved to be a major learning curve. At RMIT University we took up the challenge of trying to build the most suitable robot platform for playing robot soccer. We investigated existing platforms, examined their strengths and weaknesses, and related each design to the needs of actual soccer players. We determined a set of criteria that we believe defines the needs of soccer playing robots of the future. Armed with this knowledge we set out to design a robot chassis that fulfilled, or at least, had the potential to fulfil as many of the desirable attributes as we could. Our approach is a long term one. It is very difficult to innovate in all areas of robotics at once. Unlike virtually all other teams we took a ground up approach, developing a unique, ideally suited robot platform first, giving a strong foundation to develop more sophisticated vision and control systems.

The development of omnidirectional sphere based technology at RMIT has produced a very lightweight practical omnidirectional drive for applications requiring rapid response, robust construction and isotropic¹ maneuverability in an adversarial environment. The mechanism competed in the first world cup for Robot Soccer in Nagoya and won the inaugural Engineering Challenge award for innovation in design.

1 Introduction

A number of omnidirectional mechanisms have been proposed and demonstrated (Subir Kumar Saha 1993; Francois G. Pin. 1994; Asama 1995; Mark West 1995) and it is evident that there are a significant number of approaches that are possible when designing an omnidirectional mechanism. Wheel based platforms, using both conventional and non-conventional wheels have been developed for a number of applications (Borenstein 1985) and considerable research has gone into the development of the control of such mechanisms (Alain Betourne 1993; Igor E. Paromtchik 1994). Wheel based robots can be roughly divided into conventional wheel systems and non-conventional wheel systems. Conventional

¹ **Isotropic:** Having uniform physical properties in all directions.

wheel systems are typically in the form of differential steered, front/rear wheel steered or four wheel steered. Each of these variations has limited degrees of freedom. Non-conventional wheeled systems (Jonsson 1985; Subir Kumar Saha 1993; Asama 1995) use wheels fitted with rollers in order to allow lateral motion. Such roller mechanisms, though having the advantage of being positively driven via a central shaft are often mechanically complex, and require significant control in order to achieve any form of omnidirectional ability. To date, each of these mechanisms has been produced in metal, requiring significant machining time and skilled labor to produce. A significant weight penalty is also inherent in the construction method.

Spheres have long been used as truly omnidirectional *transfer* mechanisms. In industry, ball transfer units form omnidirectional conveyor systems and trolleys. In the standard computer mouse, a sphere translates omnidirectional motion into x and y vectors. Such mechanisms are capable of isotropic motion, however they are not easily translated into practical and simple omnidirectional sphere based *drive* mechanisms.

Existing sphere based drive mechanisms (Mark West 1995) are complex and heavy. Although precise they have not been designed to operate in an unstructured, adversarial, or even hostile environment under autonomous control. The mechanism presented in this paper has been designed specifically to operate for relatively long periods in adversarial environments. Issues such as dead reckoning, directional stability and simplicity are considered. Robot soccer(Hiroaki Kitano 1995) is used as an example of an adversarial environment in order to test the design. The working design was presented for the first time at RoboCup: The First World Cup for Robotic

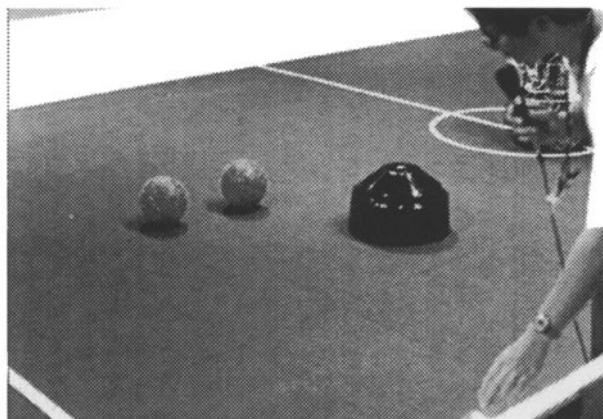


Figure 1 The RMIT Omnidirectional platform in action at RoboCup-97.

Soccer in Nagoya Japan (Andrew Price 1997) and was widely acclaimed, winning the Engineering Challenge award for innovative design.

2 An Adversarial Environment: Robot Soccer

In order to test the practicality of our design we decided to place the drive mechanism in environments that were challenging and not contrived. Adversarial environments provide a more realistic model for designing in the real world because it is very difficult to predict what an independent adversary will do in any situation. As such they represent a more realistic challenge for an autonomous vehicle test bed. Robot soccer, specifically the RoboCup-97, World Robotic Soccer Championship was selected as a manageable sized problem space for our drive mechanism. However selecting such an environment to begin with influenced the design itself. (Minoru Asada 1997) gives an overview of the challenges faced by designers of soccer playing robots. For developers of drive technology there are even more fundamental problems to be considered. One of the most significant is dead reckoning. In stable artificial environments, and environments under continuous human control dead reckoning is a manageable problem (Mark West 1995). But in adversarial environments it is quite feasible that one robot will collide (touch or bump) another robot. This is acceptable and should be expected for such an environment, particularly when the emphasis is placed on the higher level tasks and not on collision avoidance. In any case, one can only do so much to avoid a collision. In the autonomous environment of RoboCup it is not possible to keep aligning the robot manually. Though slippage can be minimized, lateral shifts of the entire mechanism render internal feedback reckoning systems ineffective. The RoboCup challenge presents substantial problems. Our robots were designed to compete in the middle sized league for real robots. We produced a team of five robots to compete against other teams of five robots. In all, up to ten robots were on the field at any time. In addition to the number of robots, the surface of the field was constructed from synthetic carpet tiles. This surface was found to be firm in places and spongy in other places. As a result, robots tended to perform differently in different parts of the field. A comparison of battery life performed on the surface prior to the competition showed that batteries consumed approximately twice the energy on the carpet compared with linoleum. Instead of developing a robot for a well managed laboratory environment, RoboCup requires consideration of unknown surface conditions, battery life, weight, speed agility, vision and a host of other real world considerations

3 Omni directional Motion

We began the design of our robot hardware with the requirement that it be the most suitable design for the environment that we could produce, in an attempt to simplify as many software problems as possible. This led to many additional problems and a five month development program. Initially we considered cost as a driving factor and examined model cars because they were economical. However we felt that although showing significant straight line speed they were not as maneuverable as we wanted, and showed a significant shortfall, if for example, the ball was sitting beside the robot, it would have to reorient itself in order to attack the ball. This would incur a time delay as well as the need to reprocess a certain amount of information when the robot reaches its attack position. This assumes that the ball is where the robot left it, which is not guaranteed.

We chose to avoid this problem by developing an omni-directional robot.

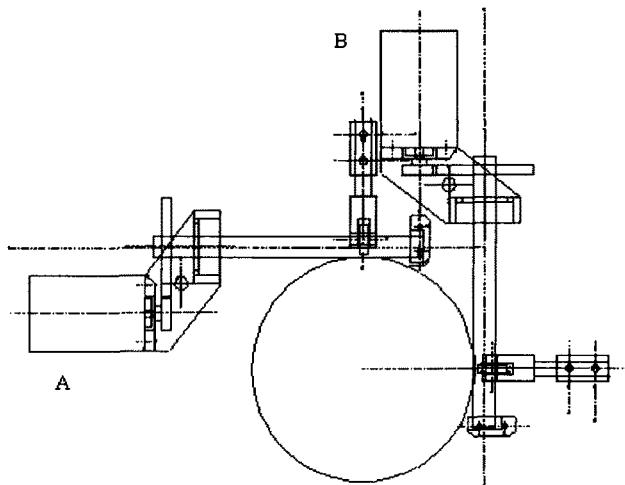


Figure 2 Omni Directional Drive mechanism

The drive mechanism for our robot is depicted in Figure 2. The principle of operation may be considered as a typical computer mouse, operating in reverse. A standard remote control car motor and gear mechanism rotates a shaft that is pressed against the drive sphere by means of a spring loaded roller. Two such shafts oriented at 90 degrees to each other drive each sphere. Thus when motor 'A' turns, the sphere is driven in the Y direction and when motor B turns, the sphere is driven in the X direction. When both motors turn, the sphere is driven at 45 degrees to the X and Y directions. By varying the speed and direction of each motor motion in 360 degrees is achieved. The spheres are suspended and are free to rotate inside a cage of bearings. The bearings, motors, and fastening screws are the only metal components in the entire design.

The design is simple and effective however its drawback is that each robot requires two drive mechanisms and therefore four motors. If we were to manufacture the drive mechanism out of say aluminium it would be prohibitively heavy. A second failing is that, we are building a team of six robots (five for the contest plus one reserve) and to produce enough parts, with comparable tolerance, in time and at reasonable cost was not possible. Hence we chose to develop our robots using plastic. Each of the components was custom made as a pattern, then cast repeatedly in lightweight high strength plastic in a very simple process. Given that four standard model car motors drive our robots and the weight for the entire mechanism including on-board processor and batteries is 5Kg , its acceleration is considerable.

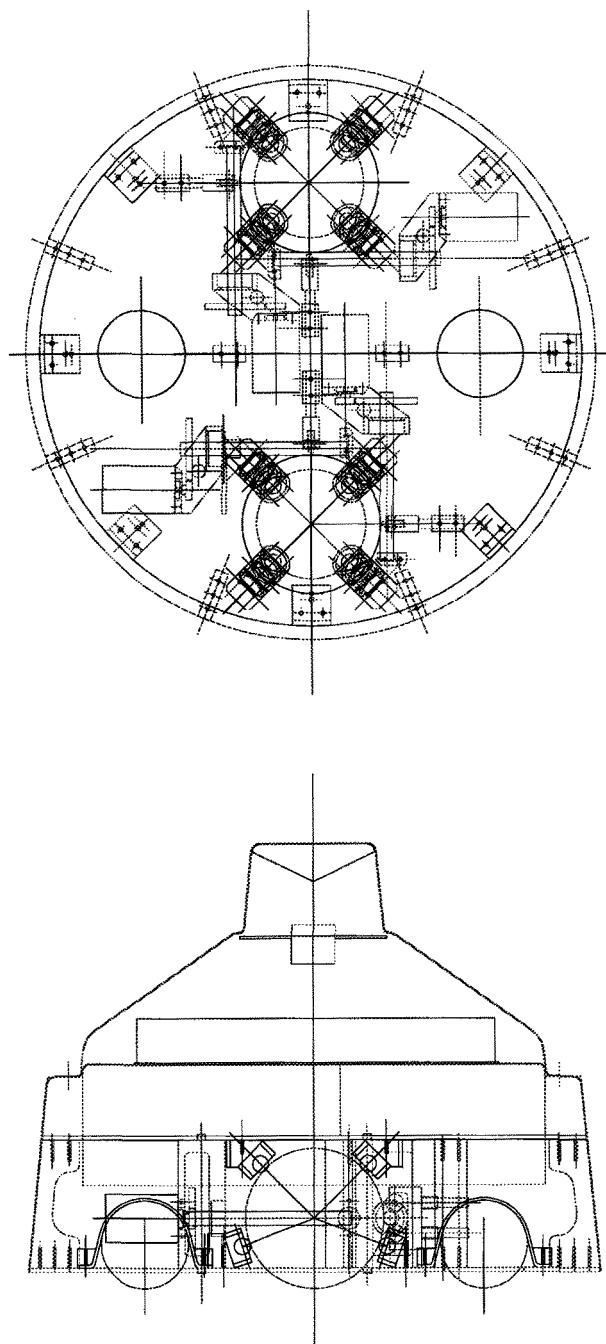


Figure 3 Omnidirectional Robot: The Raider

There are several key issues in this hardware design:

1. The drive shaft must press against the ball with enough pressure to transfer rotational energy with minimal slipping.
2. The bearings must support the weight of the robot without significantly compressing the sphere and providing unnecessary drag.
3. The sphere must grip to the arena carpet and provide suitable traction.

The exact composition of the drive sphere became a very significant factor in the success of our design. After a considerable amount of experimentation we determined that an Elastomer rubber compound with a Shore hardness of 50 was a suitable compromise for all three interfaces described above. The construction of the robot showing both drive mechanisms is shown in Figure 3. Control of the drive system is performed by an onboard HC11F1 microprocessor system using H-bridge bi-directional motor drive circuits (Jones 1993). Since the speed relationship between two motors driving the sphere at 45 degrees is slightly higher than one motor driving the ball in say, the Y direction, the robot can move faster at angles.

In our original design we planned to use optical shaft encoders to perform the task of directional and speed control. Shaft encoders can reliably detect the rotation of the sphere with respect to the drive shaft. However the drive mechanism relies on friction between the sphere and the surface of the playing field. In addition, any form of collision caused a translation or rotation of the entire mechanism that could not be reliably measured by internal sensors mounted on, or part of the drive. Field testing confirmed this problem which was exasperated by differences in the surface density, cleanliness and texture of the field. Any variation in alignment through differing motor speeds, slippage between the drive shaft and ball (a minute quantity), slippage between the ball and the field surface (a substantial quantity on initial acceleration) caused a cumulative error in directional control. Sensing slippage sufficiently accurately throughout the system would add considerable cost and complexity to the drive.

The solution to directional stability in an adversarial environment in our design relies on external feedback. For this purpose we have employed a digital compass mounted centrally as shown in Figure 4. The digital compass provides absolute directional awareness without relying on wheel sensors or slip detectors. Given that the drive mechanism contains multiple friction drive points, all of which are imperfect transfer mechanisms to some degree, the compass provides a simple means of determining direction to within one degree (+/- 0.25%).

An internal reference is stored, arbitrarily chosen as 0 degrees North. This is used to determine the error angle when the robot moves in an arbitrary direction.

Differential Error minimization is applied in two dimensions by resolving a vector from the origin of the compass at the reference angle into its components at the error angle and perpendicular to the error angle. The resulting components are then applied as moderating factors to the speed of each motor such that the error is minimized.

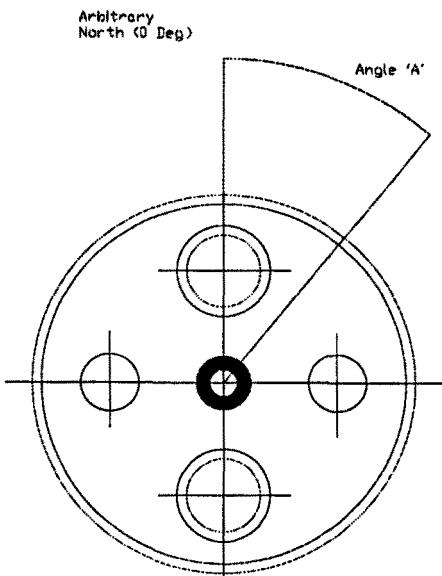


Figure 4. Compass Mechanism for Directional Stability.

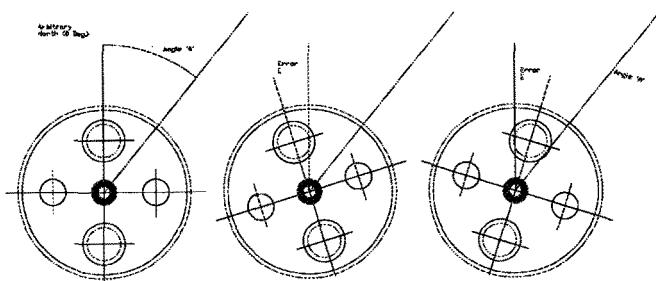


Figure 5. Error angle produced by different speeds of each sphere.

As with differential steering on a conventional wheeled robot, the correction is applied to each sphere to minimize response time and overshoot. Figure 6. shows a unit vector in the desired plane resolved into two components, one parallel to the True Plane (at the error angle) and one perpendicular to the True plane. In the first case, sphere 1 is travelling slower than sphere 2. E is assigned a positive magnitude. In the second case, sphere two is travelling faster and E is assigned a negative magnitude. Given the vectors A, B, C and D the following algorithm demonstrates how the internal reference direction may be used to correct the speed difference:

If E is positive:

Increase speed of One perpendicular to True Plane to minimize vector A (Send to 0)

Increase speed of One parallel to True Plane to maximize vector B (Send to 1)

Increase speed of Two parallel to True Plane to maximize vector C (Send to 1)

Decrease speed of Two perpendicular to True Plane to minimize vector D (Send to 0)

If E is negative:

Decrease speed of One perpendicular to True Plane to minimize vector A (Send to 0)

Increase speed of One parallel to True Plane to maximize vector B (Send to 1)

Increase speed of Two parallel to True Plane to maximize vector C (Send to 1)

Increase speed of Two perpendicular to True Plane to minimize vector D (Send to 0)

By applying correction across all motors a more rapid response, with less overshoot is achieved.

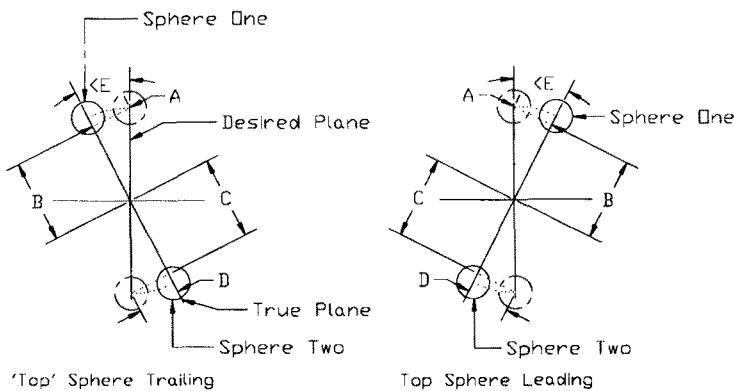


Figure 6. Correcting the speed difference based on the error angle.

Absolute direction alone cannot provide sufficient information to tell the robot where it is, only provide the necessary references to maintain orientation. Absolute position in an adversarial environment relies on external cues, or global positioning. At RoboCup-97 we used an overhead camera system. This proved effective in locating the robot on the entire field to within 25mm but was susceptible to varying light and detracted from the idea of fully autonomous and self contained robots. Our preferred solution relies on vision and other senses.

4 Vision and other Senses

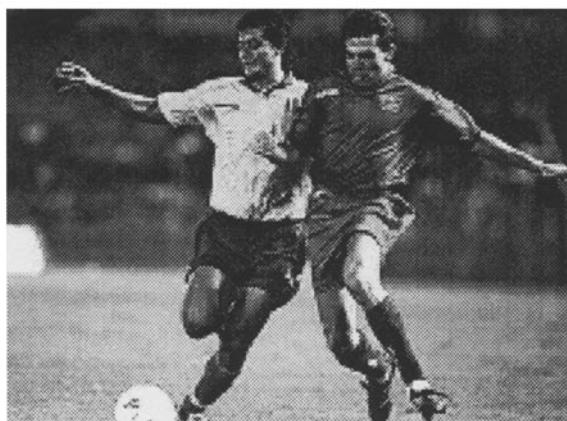


Figure 7. Soccer is a game requiring more than simple 'vision'

The two players in Figure 7. are using more than vision to determine their actions. Both players are looking at the ball, however they are in physical contact and are in a position to hear their opponent. In addition they have an awareness of what their opponent is trying to achieve. Vision is not the only sense available to a human soccer and should not, in our opinion be the only sense available to a robot soccer player. As we have already stated, robot soccer is an adversarial and dynamic environment. It is expected that robots will touch, and also that one or more robots will collide in some fashion with the ball, hopefully driving it to their own advantage. A global positioning system gives a substantial advantage by mapping the entire environment in one stroke. Positions of the ball, the robots, goals and walls are easily extracted from an image recorded centrally over the field. One of the aims of the RoboCup Challenge is to eventually use fully autonomous and self-contained robots. To this aim we have developed an omnidirectional vision system to work in conjunction with the omnidirectional drive.

In order to achieve 360 degree view and a one meter radius image surrounding a robot would have required the camera to be mounted approximately two meters above the robot. This was unrealistic. However by means of a custom designed mirror, as shown in Figure 8. and by placing the camera facing upward into the mirror we were able to achieve the required 360 degree vision with over one meter of image radius. This has several significant advantages over vision systems used at RoboCup-97.

Cameras mounted facing in one direction away from the robot (out the front on a conventional design) are susceptible to interference caused by the presence of a crowd. This occurs if the lens is sufficient to capture spectators as well as the important information on the field. In most systems where this camera arrangement is employed the offending section of the image is simply cropped. The problem with cropping is that depending on the orientation of the camera a significant amount of the image may be wasted. Since capture time and processing ability are very limited on existing robots, any loss is to be avoided.

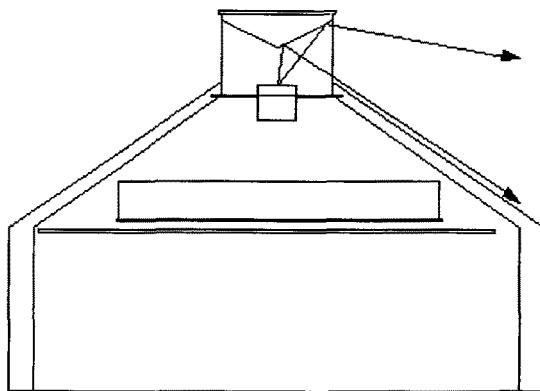


Figure 8 Omni Vision Camera System

The RMIT mirror has been designed such that the field of view is from just above the surface of the robot, to approximately 1 meter distant in all directions. The image produced is an annular ring that contains a 360 degree view of the surrounding area. By shaping the mirror only the area of interest is viewed and minimal image content is wasted. The mirror is supported on a transparent polycarbonate plastic tube that provides an unobstructed view while serving to protect the camera within and support the mirror. We anticipate this tube being damaged during the course of a soccer game, should the ball be lifted into it, and so have made this section easily replaceable. The need to view the area directly surrounding the robot dictated the shape of the robot body. The camera is capable of seeing the ball directly touching any edge of the robot and can detect the proximity of other robots or walls. Instead of supplying an overall image of the field the image produces shows the area local to the one robot. As a result the performance of the team playing soccer becomes dependent on greater communication and collaboration. Each robot locates itself within the current state of the world and acts accordingly, independent of the overall status of the world. As a result, absolute position is not as critical to our omnidirectional platform design. An awareness of direction, and the possibility of movement in any direction, preferably to the robots own advantage is crucial.

5 The Goal Keeper Problem and the Next State.

Conventional design of a soccer playing goal keeper has generally produced a mechanism that, unlike its striker and defender team mates, is capable of significant sideways motion, and generally minimal forward and backward motion. One of the most noticeable results from RoboCup-97 was the fact that goal keeper robots, though capable of stopping a ball, defending, and deflecting, were unable to directly change the state of play from defending to attacking. As a result, attacking robots that followed the ball in were often able to deflect the ball around the goal keeper. The omnidirectional platform that we have designed is capable of overcoming this problem. As a goal keeper, the robot may best serve by moving side to side. But once the ball is in the robots presence, it would be most advantageous to drive the ball away from the goal, preferably in

the direction of a team mate, or to the opposite goal. Another robot can drop back and take the place of the original goal keeper, allowing the keeper who has acquired the ball to become a striker, thus minimizing the risk of losing possession. For this reason the RMIT robots were built identical without a designated or specialized goal keeper.

A second observation from the RoboCup-97 was the need for the Control system to sense a certain condition, such as "Robot-inline with ball-inline with Goal" before it would act. In a classic subsumption architecture problem (Brooks 1986), it is necessary for robots to be able to generate the conditions necessary to achieve the next state (i.e. move from searching for the ball to positioning the ball, then striking the ball at the goal).

An advantage of our omnidirectional platform during RoboCup-97 was its extreme mobility and rapid acceleration. While applying a purely defensive strategy the team of robots managed to successfully drive the opposition back to the far end of the field. This demonstrated that agility plays as significant a role in robot soccer as it does in human soccer.

6 Logistics, Scale, and the Future of Robot Soccer

One of the greatest challenges facing future RoboCup competitors is without doubt logistics. It was amusing to note that the entire RMIT Raiders team complete with support equipment weighed less than 50kg. This was relatively easy to distribute between team members on the long flights to Japan. However it should be noted that on the return journey we chose to freight our team home. The overall cost was \$900 Australian dollars. In addition, the boxes were somewhat cumbersome; requiring that each person could only transport two at a time. Fortunately we were aware of the immense difficulties of transportation for our chosen team size. We designed solid packing containers that were themselves light. We packed our team such that there were only six boxes, to be shared between three persons. Fortune smiled upon us at the Nagoya airport where a taxi truck was available. However, it is important not to underestimate the bulk of equipment needed to support a mid size league team. We only used one PC for the entire team. If we had used one PC per robot our weight would have tripled and the number of boxes doubled. As complexity and sophistication increases the amount of support equipment is likely to increase and the cost of transportation is likely to skyrocket. We see this as being the greatest threat to the future success of RoboCup.

Without doubt our chassis system is a complex piece of hardware, both to design and construct. It is also prone to gathering dust. One of the main reasons why we chose the mid-sized league to compete was the fact that components that we simply could not manufacture had to be purchased, or adapted from some existing piece of hardware. The scale of the robots made the task considerably easier to find bearings, motors and various fittings for the mid-sized league.

Though it is now possible due to improved manufacturing and a greater range of available components and materials, primarily achieved through publicity of our success in Japan, to construct small league robots, the future of RoboCup, we believe, is in the development of bipedal and quadrupedal walking robots. These

platforms offer significant research challenges, in the production of the mechanical hardware, their control, and their coordination to play a game of soccer. For now, the small and middle sized leagues offer significant enough research challenges.

7 Conclusions

The design and construction of omnidirectional chassis is problem specific. Our platform was specifically designed for use over extended periods in adversarial environments where we could not guarantee collision avoidance. An electronic compass mechanism is used to provide absolute direction and correction for cumulative steering errors. The entire mechanism was designed to be as light and easily manufactured as possible. The complete robot weighs 5 kg. Despite being manufactured almost entirely from lightweight rigid plastic, the robustness of the design was tested with favorable outcomes at the World Robotic Soccer Championships 1997.

8 References

- Alain Betourne, A. F. (1993). "Kinematics, Dynamics and control of a Conventional Wheeled Omnidirectional Mobile Robot." Conf Proc. 1993 Intern. conf on Systems man and Cybernetics. 2: 276-281.
- Andrew Price, A. J., John Kneen, E. Kendall (1997). "Learning, Deciding, Predicting: The Soccer Playing Mind." The First International Workshop on RoboCup, IJCAI-97: 15-22.
- Asama, H., Sato, M., Bogoni, L., Kaetsu, H., Matsumoto, A. Endo, I. (1995). "Development of an Omni-Directional Mobile Robot with 3 dOS Decoupling drive Mechanism." Proc. of IEEE Int. conf. on Robotics and Automation; 1925-1930.
- Borenstein, J., Koren, Y. (1985). "A mobile platform for nursing robots," IEEE Trans. on Industrial electronis, 32(2): 158-165.
- Brooks, R. A. (1986). "A Robust Layered Control System for a Mobile Robot." IEEE Journal of Robotics and Automation, RA-2, Number 1, March 1986.
- Francois G. Pin., S. M. K. (1994). "A New Family of Omnidirectional and Holonomic Wheeled Platforms for Mobile Robots." IEEE Transactions on robotics and automation 10(4): 480-489.
- Hiroaki Kitano, M. A., Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa (1995). "Robocup: The Robot World Cup Initiative." IJCAI-95 Workshop on Entertainment and AI/Alife: 19-24.

Igor E. Paromtchik, U. R. (1994). "A Practical Approach to Motion generation and Control for an Omnidirectional Mobile Robot." Proc. IEEE International conf. on Robotics and Automation 4: 2790-5.

Jones, J. F., A.M. (1993). "Mobile Robots: Inspiration to Implementation". Wellesley, Massachusetts, 1993.

Jonsson, S. (1985). "New AGV with revolutionary movement." Proc. of the 3rd Int. Conf on Automated Guided Vehicles, Stockholm: 135-144.

Mark West, H. A. (1995). "Design and Control of Ball Wheel Omnidirectional Vehicles." IEEE International converence on Robotics and Automation: 1931-38.

Minoru Asada, Y. K., Alexis Drogoul, Hajime Asama, Maja Mataric, dominique Duhaut, Peter Stone, Hiroaki Kitano (1997). "The RoboCup Physical Agen Challenge: Phase-I (Draft)." The First International Workshop on RoboCup IJCAI 97: 51-56.

Subir Kumar Saha, J. A., John Darcovich (1993). "the Kinematic Design of a 3-DOF Isotropic Mobile Robot." Proc. IEEE Int conf on Robotics and Automation. 1: 283-288.

Omni-directional Autonomous Robots Cooperating for Team Play

K. Yokota¹, K. Ozaki¹, A. Matsumoto², K. Kawabata³,
H. Kaetsu³ and H. Asama³

¹ Department of Mechanical Systems Engineering,
Faculty of Engineering, Utsunomiya University,
2753 Ishii-machi, Utsunomiya-shi, Tochigi 321-8585, JAPAN.

² Department of Mechanical Engineering,
Faculty of Engineering, Toyo University,
2100 Kujirai-Nakanodai, Kawagoe-shi, Saitama 350-8585, JAPAN.

³ The Institute of Physical and Chemical Research (RIKEN),
Hirosawa, Wako-shi, Saitama 351-01, JAPAN.

Abstract. In order for multiple robots to accomplish a required task together, they need to communicate, organize themselves and cooperate. We have discussed these issues for the distribute autonomous robot system. The developed technologies are applied to the football game, in which the multiple mobile robots will maneuver and handle the ball towards the goal. The robots we use are holonomic, omni-directional mobile robots with vision and various sensors. They also have wireless LAN devices for communication. They can perform cooperation based on negotiation.

1 Introduction

The distributed autonomous robotic system (DARS) is a flexible and robust robotic system in which multiple robots and other agents act in a cooperative and coordinated manner to achieve tasks which are not easy for a single robot [1]. Communication, self-organization and cooperation are essential technologies for DARS [2].

We regard the football game as a collection of cooperative actions of autonomous robots and consider that the above technologies are also essential for us to construct various *plays* in the football game. In order for a team to score a goal, its players must cooperate in bringing the ball forward, maneuvering between fore players. One robot may assume a particular role in a situation, but changes the role in another, depending on the overall organization of the team suitable for the given condition.

To this end we assembled a team of five omni-directional, autonomous robots which are capable of communicating and sharing information between them. The omni-directional mechanism facilitates rapid and dextrous movement. It gives greater freedom to a robot in designing a path in the game, and the robot can implement rapid and reactive motion.

2 Framework for Cooperation

There are two approaches to realize cooperation among robots in the multi-robot system: one is based on negotiation between the robots, the other is based on sensing.

In the negotiation-based cooperation, communication among robots is important. The robots explicitly exchange various information, such as requests, offer and intention, via communication media in order to negotiate with each other. Negotiation is used to organize robots, allocate resources and resolve conflicts [3][4]. We designed a communication system for the multi-robot system and the protocol for negotiation [5][6]. The negotiation-based cooperation is more synthetic than the sensing-based cooperation.

In the sensing-based cooperation, individual robots decide their next action according to sensory information. Their collective actions constitutes cooperation. There is not explicit exchange of information or intention among the robots. Therefore, if the sensing and reasoning ability of the robot is limited, we may as well say that this type of cooperation is just the result of selfish actions of the robots and they are not aware that they are participating in the cooperation. The sensing-based cooperation can be made more reliable if the robot is capable enough to recognize the other robots, understand the given situation and reason its next action guessing the next action of nearby robots,

While we can expect that robots and the overall system will behave in predictable manners in the negotiation-based cooperation, thus, it is considered more appropriate for implementing a specific cooperative task in a reliable way, the sensing-based cooperation will be suitable when the robots need to react quickly to the situation and there is not sufficient time to enter into negotiation with the others.

3 Omni-directional Autonomous Mobile Robot

3.1 System Overview

Fig. 1 illustrates a football playing robot team of five omni-directional autonomous mobile robots and a human interface system.

The robots are autonomous and self-contained. The control system of each robot is composed of a processor board, an network interface board, a D/A converter and a digital I/O board on the ISA bus. The CPU is an i486DX4 processor running at 100 MHz. Programs run on the real-time, multi-task operating system VxWorks.

Each robot is equipped with a CCD camera connected to a frame memory board whose resolution is 512 dots by 512 dots. Infra-red and ultrasonic sensors are also provided[7]. Although configuration of the driving mechanism and the main processing system on each robot are approximately equivalent, specification of sensing systems are different. Therefore, the overall system is a heterogeneous multi-robot system.

The human interface system is used only for instructing the robots to start or stop the game. It is not used to control the robot.

The robots are equipped with a wireless communication device. The human interface system is connected to wired communication network. In order for the robots and the human interface system to communicate, we have provided two bridges between the wireless and wired networks. The specification of the communication device is shown in Table 1.

But for the goal keeper, all the robots currently do not have an actuator for propelling the ball. They give the ball the initial speed simply by colliding with it.

Table 1. Specification of the Wireless LAN Device

Frequency	2471–2497 MHz
Modulation	Spectrum diffusion modulation
Speed	2 Mbps
Distance	up to 60 m
Transmission method	CSMA/CD

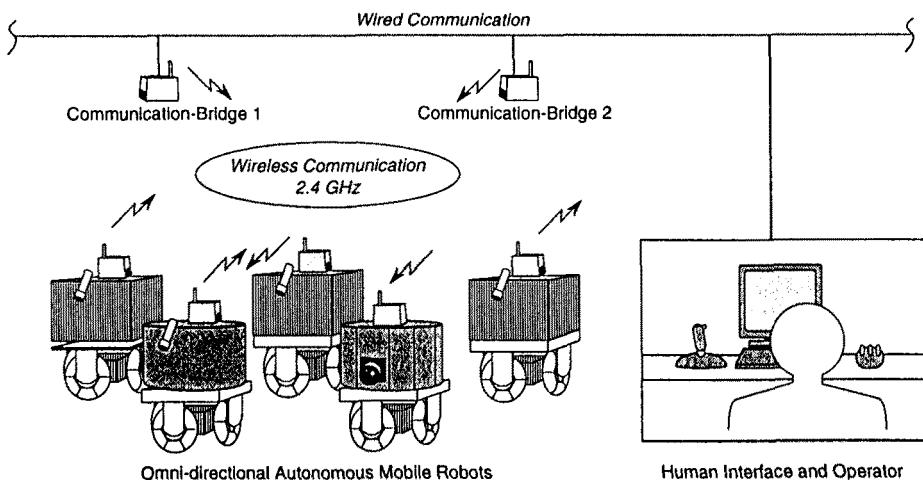


Fig. 1. System Overview

3.2 Omni-Directional Driving Mechanism

The omni-directional driving mechanism of the mobile robots used at the RoboCup is the improved design of the prototype[8].

The mobile robot which operates in the two dimensional space, namely on a floor, has three DoF (degrees of freedom), of which two are for translational motion and one is for rotational motion. A holonomic omni-directional mobile robot should be able to be controlled in any combination of the three DoF in any condition.

In order to realize stable translational motion, we decided to utilize four special wheels. The structure of the special wheel is shown in Fig. 2, in which free barrel-shaped rollers with small diameter and large diameter are arranged alternatively in the single raw along the circumference. The friction of the free rollers is small enough so that the driving force of the wheel is applied to the ground only in the circumferential direction, and the external force exerted in the axle direction is released by the free rollers. Therefore, the active wheel motion by driving the axle can be effective to the forwarding motion, but the wheel is passive to the axle direction.

Fig. 3 shows the arrangement of four wheels. The translational motion along X_r is generated by driving wheel 2 and 4 in the same direction, when wheel 1 and 3 do not interfere the motion. In the same way, the translational motion along Y_r is generated by driving wheel 1 and 3, when wheel 2 and 4 do not interfere the motion. Translational motion along any diagonal direction can be produced by composition of the translational motion along X_r and Y_r . The rotational motion about Z_r is produced by rotating all four axles about their axes to the same wise (clockwise or counter clockwise). Thus, the holonomic omni-directional motion can be obtained with these four special wheels. This mechanism makes the translational motion stable because the motion is realized by driving two parallel wheels.

If four actuators are provided for each wheel, the system becomes redundant because the system has four DoF which are larger than three, the DoF to be controlled. In order to eliminate redundancy, we designed a 3 DoF decoupling drive transmission mechanism which enables driving four wheels by three actuators. Fig. 4 illustrates the transmission mechanism of the omni-directional mobile robot. (The actual robot coordinate system is located at the center of the robot.) With this mechanism, three DoF motion, namely two DoF translational motion along X_r and Y_r axis and one DoF rotational motion about Z_r axis, are mechanically decoupled and driven by each correspondent actuators without redundancy.

The driving force of actuator 1 is transmitted to the shaft connected to the spur gears 2 and 4, and revolves the wheel 2 and 4 via differential gear 2 and 4 to translate the robot to the X_r direction. In the same manner, the driving force of actuator 2 contributes the translational motion to the Y_r direction. The driving force of actuator 3 revolves four shafts connected to the other input of differential gears through the bevel gear, which rotates four wheels about their axes to the same wise (clockwise or counter clockwise). When the robot translates with

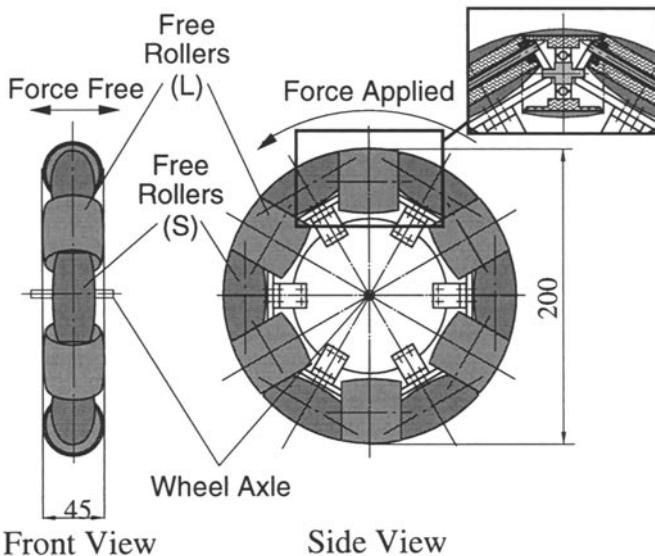


Fig. 2. Structure of a wheel

rotation, the differential gears produce the required velocity difference between two opposed wheels along each translational direction.

By solving the equations in terms of gear transmission, the following kinematics is obtained:

$$\bar{\dot{\mathbf{p}}} = \begin{pmatrix} \bar{\dot{x}} \\ \bar{\dot{y}} \\ \bar{\dot{\theta}} \end{pmatrix} = \begin{pmatrix} 2k_1 k_3 r & 0 & 0 \\ 0 & 2k_1 k_3 r & 0 \\ 0 & 0 & k_2 k_3 \frac{r}{R} \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix} \equiv \bar{\mathbf{J}} \dot{\mathbf{q}} \quad (1)$$

where k_1 , k_2 , k_3 denote reduction ratios of the spur gears, the bevel gears, and the harmonic drives respectively, and r and R denote radius of a wheel and half tread (the distance from the center of robot to the center of the wheel) respectively. $\bar{\dot{\mathbf{p}}}$, $\bar{\mathbf{J}}$, and $\dot{\mathbf{q}}$ denote the velocity of the robot relative to $\Sigma_{\bar{\tau}}$, the Jacobi matrix with respect to $\Sigma_{\bar{\tau}}$, and the velocity of actuators. Since $\bar{\mathbf{J}}$ turns out to be a diagonal matrix, it is proved that translational motion along each directions or rotational motion of the robot with respect to $\Sigma_{\bar{\tau}}$ is decoupled by this transmission mechanism and can be driven by each correspondent actuators.

As other mechanical characteristics, suspension mechanisms are equipped for each wheel so that each wheel can react to the uneven floor and all the wheels keep contact with it. In addition, a harmonic reduction gear is incorporated in each wheel so that the large backlash in the transmission mechanism can be reduced at the wheel.

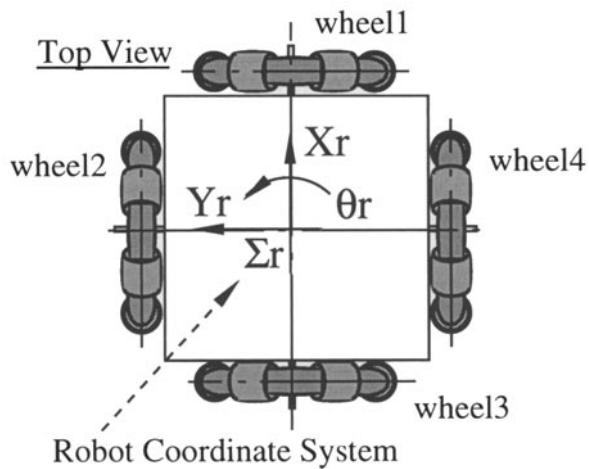


Fig. 3. Arrangement of wheels

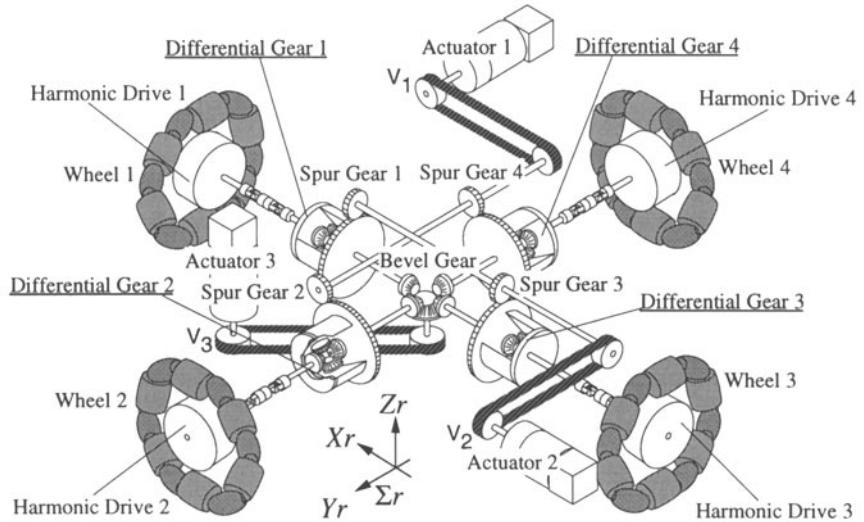


Fig. 4. Transmission mechanism of the omni-directional mobile robot

This holonomic omni-directional mobile mechanism with 3 DoF decoupling drive transmission fulfills two requirements: stable translational motion and non-redundancy. The 3 DoF decoupling drive transmission mechanism has another advantage that computation of the output to the actuators is simple because each actuator contributes only to the correspondent DoF motion.

The omni-directional mechanism facilitates dexterous movement. It enables us to implement rapid and reactive motion of the robot.

3.3 Communication System

In order to achieve cooperation or team play by multiple robots, it is necessary for the robots to exchange various messages. We defined two classes of communication procedures for this purpose.

1. Announcement
2. Negotiation

Announcement is to share information. *Negotiation* is for planning. We further classify the negotiation into three categories which correspond to planning and behavior for team play, and defined three types of message exchange for following means: *collective decision*, *request* and *order*.

Fig. 5 shows the time charts of *announcement*, *collective decision*, *request* and *order* respectively. A message is formatted as shown in Table 2.

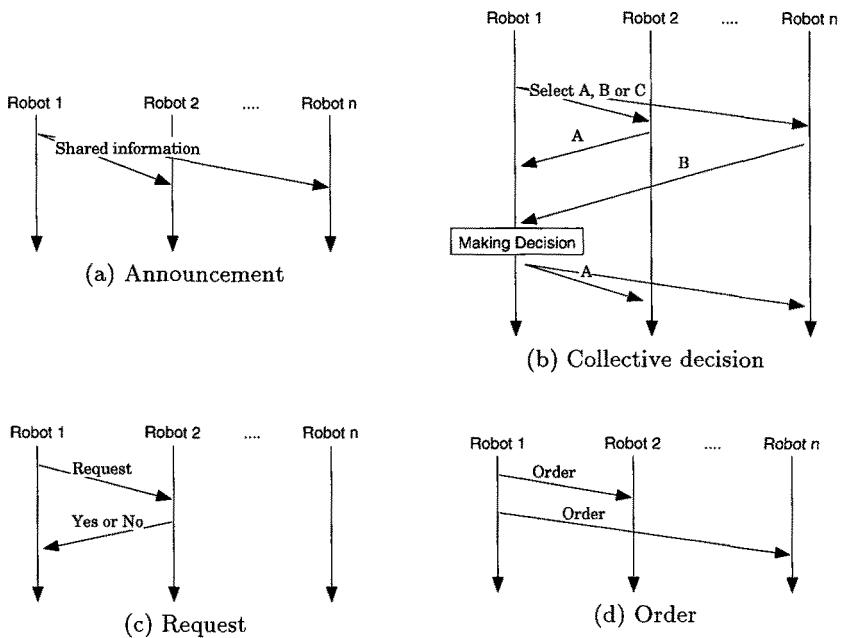
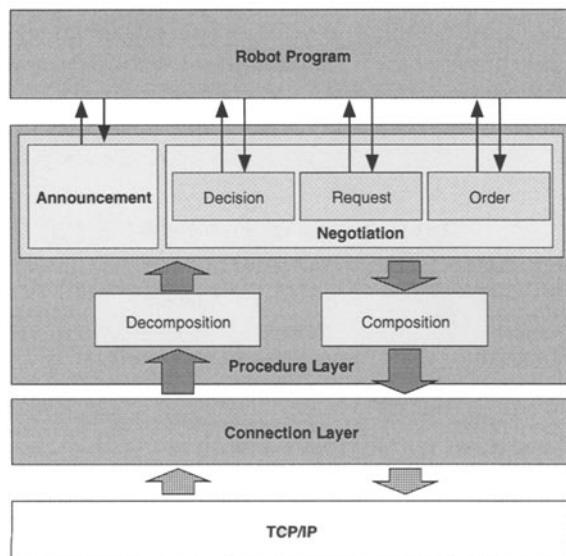
The communication system is implemented as shown in Fig. 6. Modules are layered on top of TCP/IP protocol. The *Connection Layer* establishes and manages connection between robots and the human interface system. The *Procedure Layer* transmits and receives messages through the *Connection Layer*, and executes procedures described above. The upper layer robot programs do not need to care about details of communication procedures.

Table 2. Message Framework

Field	Contents
To:	<i>Receiver's Address</i>
From:	<i>Sender's Address</i>
Control:	<i>Communication Control</i>
Type:	<i>Procedure Announcement or Negotiation</i>
Message:	<i>Message Body</i>
Reply:	<i>Results or Status</i>

3.4 Sensing System

The robot is equipped with a CCD camera. It is the primary means for the robot to recognize the environment. Under the rules of RoboCup, objects in the

**Fig. 5.** Message Exchange**Fig. 6.** Software Architecture in Communication System

field are assigned different color schemes, thus, the vision system is optimized to extract colors quickly, rather than detecting shapes of the objects.

Consideration should also be given to the facts that lighting condition of the field cannot be accurately predicted beforehand, and that the field may not be lighted evenly. With up to ten robots in the field, we should expect there will be many shadows casted by the robots. For this reason, the captured camera image in RGB values is first converted to the CIE representation, then, colors in the image is extracted.

In order to process the image quickly, the resolution of the image is reduced to 64 dots by 64 dots from 512 dots by 512 dots, and integer arithmetic is used wherever possible. The maximum image processing speed is approximately 7Hz. Given the fact that a single CPU is used for both actuator control and image processing, this is sufficiently fast.

4 Cooperation for Team Play

4.1 Team Level and Individual Level Task Modeling

When programming robots to cooperate with each other in order to achieve a common task, it is important to model and represent the actions of not only individual robots but also the team of the robots as a whole. Otherwise, it would be extremely difficult to verify that the algorithm is correct and the system is feasible.

We model the football game as a set of state transition graphs of which graphs at the higher level correspond to the states and actions of the team and graphs at the lower level represent the states and actions of individual robots. This way, team level behaviors and individual level behaviors are expressed in an integrated and uniform manner. Fig. 7 shows the top level state transition graph which represents the states of the team. The graph starts with kick-off and ends when a goal is scored, time is up or the judge suspends the game. The state of the team moves between *offense* and *defense* states.

The offense states are illustrated in Fig. 8. Offense starts when one of friendly robots acquires the control of the ball and ends when a goal is scored, half-time is up, the judge suspends the game or the opponent takes control of the ball.

The states of the individual robot are represented in lower levels. Fig. 9 shows the individual level graph for the *infiltration* state. A robot is in one of five states according to the current situation. If the robot is in control of the ball, it may choose to dribble the ball to the enemy field, or pass the ball to another robot. If it currently doesn't have the ball (but another friendly robot does), it may choose to receive the passed ball if so requested, run into the enemy field, or stay put. Whichever state in Fig. 9 the robot is in, it understands that his team is in the *infiltration* state.

4.2 State Recognition

The current state of the team and transition of between states are recognized by the robot by referring to the current environment information. As the infor-

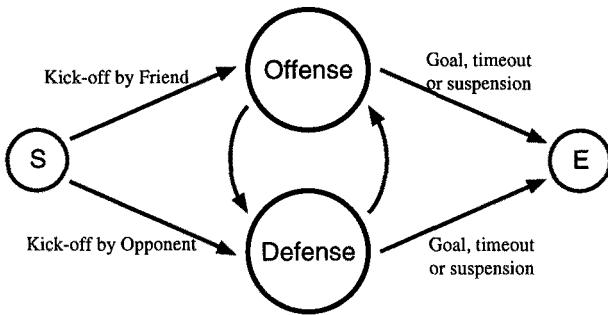


Fig. 7. State Transition Graph – Team Level

mation is updated asynchronously and we do not guarantee that information held by one robot exactly matches the others', as described in 4.5, there is good possibility that the state recognized by a robot is not the same as the state recognized by another robot.

For example, if the team has lost the sight of the ball and in the *ball search* state and one robot finds the ball and tries to tell the others the location of the found ball, the others continue to think they are still in the *search* state until they receive the message. There can be significant delay before the last robot receives the message, depending on the condition of communication traffic.

4.3 Planning Actions

The state transition graphs depict possible states and transition between them. However, they do not directly show how the robot will plan for next actions. Action planning concerns two issues:

- The path along which the ball is brought forward to the goal.
- How the ball is handled (dribbled or passed) along the path by which robot.
- Who plans the path and when.

The first issue is more strategic than the second and closely related with the formation of robots in the field. The second issue is tactical and should be decided according to the current situation.

The path is planned in terms of areas defined in 4.4, because it is unreasonable to expect that the robot can bring the ball exactly along a line. A path may be planned from scratch or chosen from pre-defined paths in a database. The paths are indexed by its start position (area) in the database so that they are looked up with the current position of the ball as a key.

Formation of robots is another possible areas of strategy for defense and offense [10].

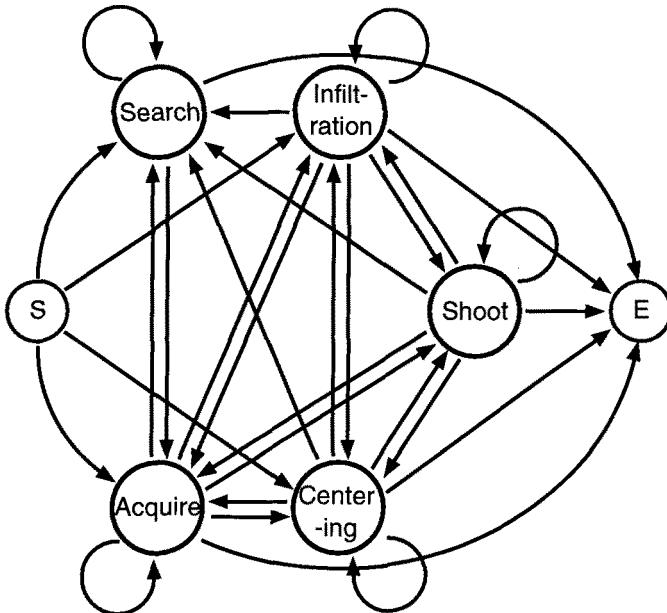


Fig. 8. Team Level Offense States

4.4 Environment Information

Information on the working environment can be divided into two categories: static and dynamic. The static environment information in the robotic football includes the field map. The position of the robot measured by itself during the game bounds to be inaccurate, whether it is calculated by dead reckoning or obtained by vision or active sensors. Thus, the representation of the field map should offer a means to approximate positions in the field. The field is covered with hexagonal cells (Fig. 10). Four adjacent cells constitute an area and areas are classified into friend, neutral and enemy zones.

The robot recognizes the positions of the other robots and the ball by means of cells and areas. When it needs to communicate position to another robot, a cell or area number will be transmitted rather than numerical coordinate values. This map representation is also useful for the robot when planning next actions, because exact positions are difficult to handle when planning paths of the robot and the ball. When planning the path, it is more logical to plan in terms of approximated positions, such as “near the enemy goal” and “the opposite side”.

As there are only a limited number of robots in the field, there are not many pieces of dynamic information.

- information about the ball
 - position

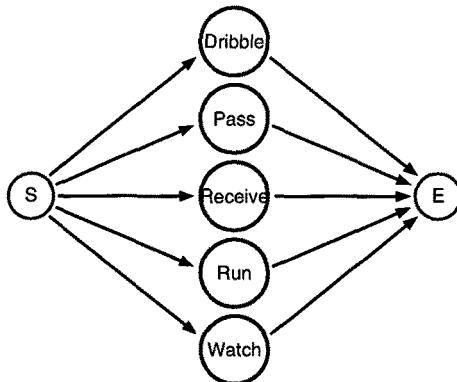


Fig. 9. Individual Offense States

- state
 - * the ball is controlled by a friendly robot
 - * the ball is controlled by an opponent robot
 - * the ball is free
 - * unknown
- information about friendly robots
 - name (ID)
 - position
- information about opponent robots
 - name (ID)
 - position

The positions are expressed in terms of cells or areas. Friendly robots are distinguished by IDs. There currently is no way to distinguish individual opponent robots, the team cannot track the behavior of a particular opponent. It may also loses sight of opponents, thus, not all positions of the opponent robots are known during the game.

Of the above pieces of information, the position and ownership of the ball are most important as the robot will decide whether it should offend or defend, based largely on these facts.

The number of items is small but these pieces of information tend to change frequently during the game, thus, the information which is not updated for certain duration should not be relied upon.

4.5 Information Sharing

Dynamic environment information is shared among the friendly robots and needs to be routinely updated. The information can be managed either at the central agent or by each robot. It is easier to maintain centrally managed information

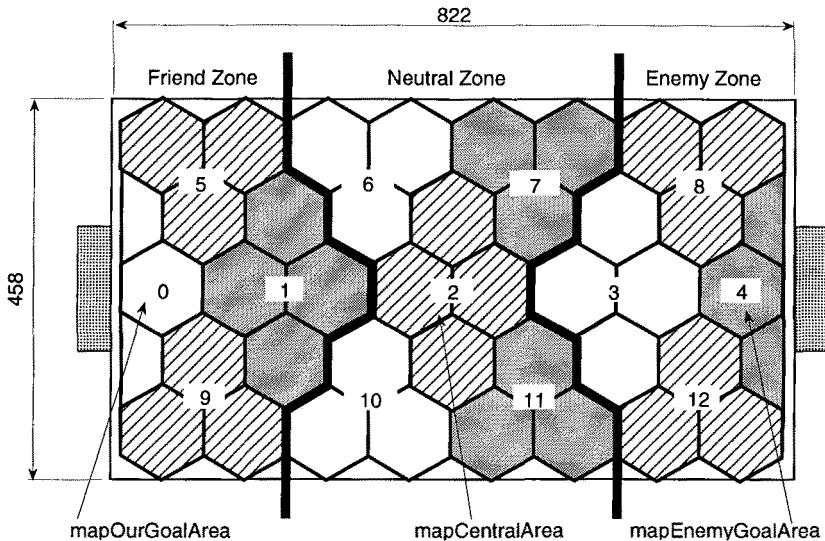


Fig. 10. Static Environment Information – Field

and guarantee its consistency than to do same to distributed information. Data exchange is less intensive for the distributed information than for the centrally managed information [3].

It is important to consider robustness and reliability of information itself and the procedure to share and update the information. We cannot discount the possibility that communication between robots may be disturbed by noise or failure of communication devices. It is one thing to devise error correction and recovery procedures. However, in the quickly changing environment such as in football game, cost and time spent for such procedures need to be minimal.

As discussed in the previous section, the dynamic environment information in the robotic football tends to change quickly. Each robot has to frequently refer to the information in order to make decisions on the next actions. For this reason, we adapt the distributed model.

There is no guarantee that the information held by a robot is correct and up to date. In other words, it may not be the same as the information maintained by the other robots at any given time frame in the game. This means that robots may make different judgment on the given situation and their chosen actions may not be optimal, although each robot thinks its decision is correct and appropriate for the situation. It is far more important for the team to let each robot make decisions and act in a responsive manner than to leave the robots spending a long time trying to find the optimal actions.

5 Analysis

5.1 Performance at RoboCup-97

The hardware of the robots of our team, UTTORI United, is robust and was surprisingly free of major mechanical troubles at RoboCup-97 in Nagoya, apart from one CCD camera which did not work sufficiently under the strong light in the field. There were failures of small parts. Fortunately they did not lead to failure of the entire system.

However, the team did not perform very well in matches mainly because of software problems described in the following sections.

5.2 Software Structure

The team consists of five robots from three separate institutions. They are used for various different projects at respective laboratories. They share the same overall structure and mechanisms as described in Section 3. However, due to differences in I/O subsystems, sensor systems and experimental hardware, software modules of each robot are patchwork of common and unique routines.

Because of this, the robots were not at the even level and there were rough edges in different part of software when the team was assembled for RoboCup-97. In one sense the team is a very good example the heterogeneous multi-agent system of robots of varying capabilities. In another sense, it was not a easy task to coordinate software development and ensure the desired operation of the robots.

5.3 Communication Problems

It appears that the venue of RoboCup-97 suffered from major radio disturbance. Our team was vulnerable to the radio disturbance in three areas. First, software of each robot needed to be downloaded from the workstation via wireless LAN. When radio condition was bad, it took a long while to complete this process. This is why the team needed a long time before starting a match.

Second, when the robot is placed in the field, it will wait for the start command from the operator. The command is again sent via radio. After this command, the robot acts autonomously and the human operator can do nothing but stop the robot by sending another command. During the matches in Nagoya, it happened that some robots received the start command and started moving, but the others did not seem to receive the command and stayed motionless.

Third, communication is used to share and update information among the robots, as described in the previous sections. If no message comes from outside, the robot will act independently based on its own sensory information. However, the robot may be left in the same state and action for a prolonged time. For example, if a robot is not able to find the ball by itself but another robot does, and if the report that the second robot has detected the ball does not reach the first robot, it will continue searching for the ball. This possibility was foreseen as discussed in 4.5, but the delay was longer in practice than anticipated.

6 Conclusion

In the multi-robot system, communication, self-organization and cooperation are essential technologies for the system to achieve tasks in robust and flexible manners. The football game consists of a series of plays which multiple players participate in and cooperate for. The robot with the omni-directional driving mechanism can make dexterous movement and react quickly to the given situation detected by sensors. Individual robots can perform only relatively simple actions such as kick and catch. With communication and self-organization, they can exercise more complicated maneuvers such as passing the ball.

References

1. Asama, H., Fukuda, T., Arai, T. and Endo, I. (Eds). *Distributed Autonomous Robotic Systems*. Springer-Verlag, (1994).
2. Asama, H., Matsumoto, A. and Ishida, Y. *Design of an Autonomous and Distributed Robot System: ACTRESS*. Proc. of IEEE/RSJ Int. Workshop on Intelligent Robots and Systems, (1989) 283–290.
3. Asama, H., Ozaki, K., Ishida, Y., Habib, M. K., Matsumoto, A. and Endo, I. *Negotiation between Multiple Mobile Robots and an Environment Manager*. Proc. of 5th Int. Conf. on Advanced Robotics, (1991) 533–538.
4. Asama, H., Ozaki, K., Itakura, H., Matsumoto, A., Ishida, Y. and Endo, I. *Collision Avoidance among Multiple Mobile Robots Based on Rules and Communication*. Proc. of IEEE/RSJ Int. Workshop on Intelligent Robots and Systems, (1991) 1215–1220.
5. Matsumoto, A., Asama, H., Ishida, Y., Ozaki, K. and Endo, I. *Communication in the Autonomous and Decentralized Robot System ACTRESS*. Proc. of IEEE/RSJ Int. Workshop on Intelligent Robots and Systems, (1990) 835–840.
6. Asama, H., Ishida, Y., Ozaki, K., Habib, M. K., Matsumoto, A., Kaetsu, H. and Endo, I. *A Communication System between Multiple Robotic Agents*. Proc. of Japan/USA Symp. on Flexible Automation, (1992) 647–654.
7. Suzuki, S., Asama, H., Uegaki, A., Kotosaka, S., Fujita, T., Matsumoto, A., Kaetsu, H. and Endo, I. *An Infra-Red Sensory System with Local Communication for Co-operative Multiple Mobile Robots*. Proc. of IEEE/RSJ Int. Workshop on Intelligent Robots and Systems, (1995) 220–225.
8. Asama, H., Sato, M., Bogoni, L., Kaetsu, H., Matsumoto, A. and Endo, I. *Development of an Omni-Directional Mobile Robot with 3 DoF Decoupling Drive Mechanism*. Proc. of IEEE Int. Conf. on Robotics and Automation, (1995) 1925–1930.
9. Ozaki, K., Asama, H., Ishida, Y., Yokota, K., Matsumoto, A., Kaetsu and H., Endo, I. *Negotiation Method for Collaborating Team Organization among Multiple Robots*. in *Distributed Autonomous Robotic Systems*. Springer-Verlag, (1994) 199–210.
10. Matsumoto, A. and Nagai, H. *An Approach to the Evaluation of Decentralized Decision Making in Multi-agent Robotic Soccer*. Proc. of the 3rd France-Japan Congress on Mechatronics, (1996) 411–415.

The Spirit of Bolivia: Complex Behavior Through Minimal Control

Barry Brian Werger¹ with team members Pablo Funes¹,
Miguel Schneider-Fontán¹, Randy Sargent², Carl Witty², and Tim Witty²

¹ Ullanta Performance Robotics
CSCI - SAL 300, 941 West 37th Place (USC), Los Angeles, CA 90089-0781

<http://www-robotics.usc.edu/~barry/ullanta>

² Newton Research Laboratories
14813 NE 13th St., Bellevue, WA 98007 - <http://www.newtonlabs.com>

Abstract. The “Spirit of Bolivia” is a robotic soccer team which demonstrates *minimally comprehensive team behavior*. By this we mean that each member of the team makes progress towards team goals, and obstructs progress of the opponent, by interacting *constructively* with teammates and in a *sportsmanlike* manner with opposing players. This complex behavior is achieved with simple on-board processors running very small behavior-based control programs; team behaviors are achieved without explicit communication. *Externalization* - the use of the environment as its own best model - and *tolerance* - a bias towards reducing the need for accurate information rather than attempting to recognize or correct noisy information - are the keys to robustness and sophistication of team behavior.

1 Introduction

While it is often tempting to look to more powerful hardware, more complicated programs, more precise sensors, and more communication in the attempt to generate more sophisticated robotic behavior, we feel that the key to success in such an endeavor lies rather in the philosophy of trying to fully exploit minimal systems ([7]). Not only are we far from getting the most out of even the simplest robots; experience has shown that in many cases more precise, “better” robots are more difficult to control ([6]). We present here the “Spirit of Bolivia,” a team of RoboCup mid-size soccer-playing robots that exhibit a high degree of sophistication with minimal control and tolerance for - rather than attempts to reduce - imprecision of information. The team is able to keep the ball consistently in play, engaging in constructive offensive and defensive group and individual behavior without explicit communication or modeling of the environment. [1], [3], [2], and [8] have pointed out in various ways the benefits of replacing models of the environment and precise calculation with *externalization* - the use of the world not only as its own best model, but as a convenient medium for implicit communication.

Though the “Spirit of Bolivia” is primarily a software effort, we next present a brief description of the physical hardware for context. We then describe the

individual and group behavior of the system, and discuss its performance in the RoboCup '97 competition.

2 The Robots

The "Spirit of Bolivia" consists of five robots: four fielders (all but one borrowed), Pioneer I robots from ActiveMedia/RWI; and one goalie, built by Newton Labs with an RWI B14 base. A description of the behavior of the goalie appears only here since it is distinct from, and simpler than, that of the fielders.

The financial necessity of borrowing robots for the competition had two major implications: on one hand, the robots were not specialized for RoboCup competition, as were all competing teams; on the other hand, the human team members were able to spend all of their development time on team behavior without uncertainty of hardware reliability or functionality. In hindsight we feel that the tradeoff was highly beneficial.

Robot Vision All of the robots used the Cognachrome [5] vision system from Newton Laboratorie. Based on a 16-MHz 68332 microcontroller, the system extracts color-blob information from a video source at a frame rate of 60 Hz. It can be trained to recognize three distinct colors at a time, and outputs the size, visual field location, bounding box, and other data for multiple blobs of each color. It also provides various types of line-tracking data. The robots used wide angle (60-degree) cameras which were not actuated.

The Pioneers - Llaqwa, Sillp'ancho, Salteña, and Ullanta the Robot
Levin The Pioneer I robots, as pictured in Figure 1, feature an 18"x14" differentially steered base, five forward- and two side-facing sonars, and grippers (made into large bumpers to comply with RoboCup rules) with contact sensors on each finger. Llaqwa and Ullanta are equipped with the MARS/L system from I.S. Robotics, an on-board 68332 programmed in a Common LISP extended for behavior-based control; Salteña and Sillp'ancho are run with offboard processors over a radio modem link, programmed in C using the PAI libraries from ActivMedia (Salteña) and P-LIGO (Sillp'ancho).

The only modification made to the Pioneers for RoboCup is the raising of their sonar pingers to a level (two inches higher than normal) where they do not perceive the ball. The pitch-angle of the forward facing camera is set so that a wall at maximum field distance reaches to the top of the field of view, preventing perception of anything outside of the field.

The Goalie - Papa Wayk'u The goalie is an RWI B14 synchronous-steered base with two Cognachrome vision systems for sensing and control. Its simple task is to stay between the ball and the goal while moving along the goal line.

The camera connected to one of the vision systems points out into the field, in order to locate the ball and report its position to the other vision system. To locate the ball, the vision system finds the largest red object below the horizon line; if this is larger than a preset threshold, it is assumed to be the ball. This vision system relays the ball location to the second vision system 60 times per second.

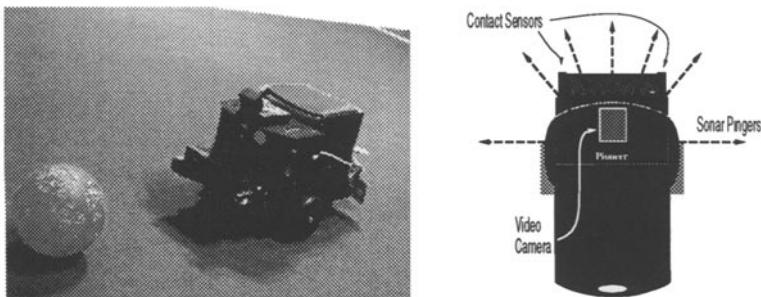


Fig. 1. The Pioneers are differentially steered bases with seven sonar pingers, grippers with contact sensors on the tips, and a Cognachrome vision system.

Since the camera has less than a 180 degree field of view, it can't see the entire field. We point the camera at a 45-degree angle so the goalie looks "ahead and left" (see Figure 2). Whenever the robot doesn't see the ball, it assumes the ball is to its right, and moves to intercept it. By maintaining a "guard" position at the right end of the goal, Papa Wayk'u is able to see the ball whenever it approaches the goal, and quickly zip across to block it or "kick" it out of the way.

The second vision system uses a line-following algorithm to follow along the goal line. It moves left and right based on information received from the first vision system using a simple proportional control loop.

3 The Behavioral System

Figure 3 shows the overall structure and information flow of the fielders' behavioral system. There are two main behavioral groupings: the ball-manipulation behaviors centered around the *Object Tracker*, and the safety-related behaviors *Avoidance Generator* and *Velocity Generator*. Team behaviors "emerge" out of the interactions between these two groups.

3.1 Ball Manipulation

The fielders manipulate the ball in three ways: pushing it forward, kicking it forward, and batting it to the side. These three forms of manipulation are implemented through a set of *basis behaviors* [4]; this is a minimal set of behaviors that is sufficient for generating the complex trajectories necessary for soccer play. The set consists of *Orbit-CW*, *Orbit-CCW*, and *Kick-Ahead*, which are illustrated in Figure 4.

These three behaviors are mutually inhibitory in such a way that only one is active at a time. Changes of active status happen at a potential rate of 10 Hz,

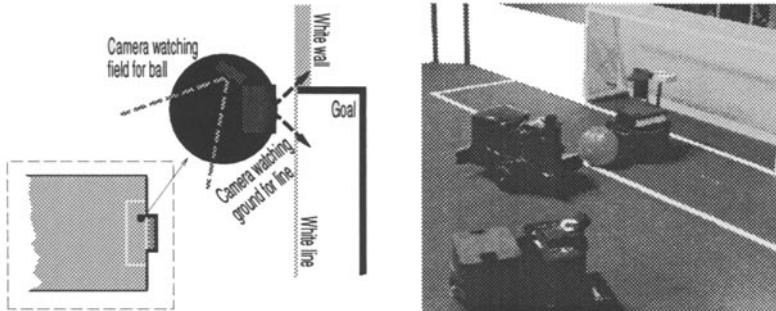


Fig. 2. The Goalie is a synchronous-steered base with two Cognachrome vision systems: one which tracks the line across the goal entrance, and one which watches for the ball.

which allows generation of smooth trajectories such as can be seen in Figure 5, and quick enough reaction times for proper handling of a moving ball.

Interpolation The two *Orbit* basis behaviors suffice for interpolation of a fielder between the ball and its own goal, which is essential to both offense and defense. In the process of doing so, we want the robot to avoid ever getting between the ball and the opponent's goal (which may block a teammate's actions, and increases the likelihood of a self-goal). We also want the robot to be facing towards the opponent's goal once interpolated. The switching of activation to achieve this interpolation and move the ball forward once in a good position is one of the functions of the *Object Tracker*, which makes a determination as follows:

Given **B** is the ball, **OG** is the opponent's goal, **MG** is the player's goal, and "north" is the direction towards **OG**,

```

If I see B
  If I see OG
    If B overlaps OG,          activate Kick-Ahead
    Else if B is left of OG,   activate Orbit-CW
    Else if B is right of OG,  activate Orbit-CCW
  Else if I see MG
    If B is left of MG,        activate Orbit-CCW
    Else if B is right of MG,  activate Orbit-CW
    Else if I'm facing north,  activate Kick-Ahead
    Else if I'm facing east,   activate Orbit-CCW
    Else if I'm facing west,   activate Orbit-CW
  Else I don't see B,         activate Patrol

```

Thus, an *Orbit* behavior is maintained around the ball until the robot's perception indicates that it is in position to advance towards the opponent's goal. Directional determinations (north, east, or west) are made by dead-reckoning. The reckoning is accurate only for short periods, but the combination of very rough distinctions (forty

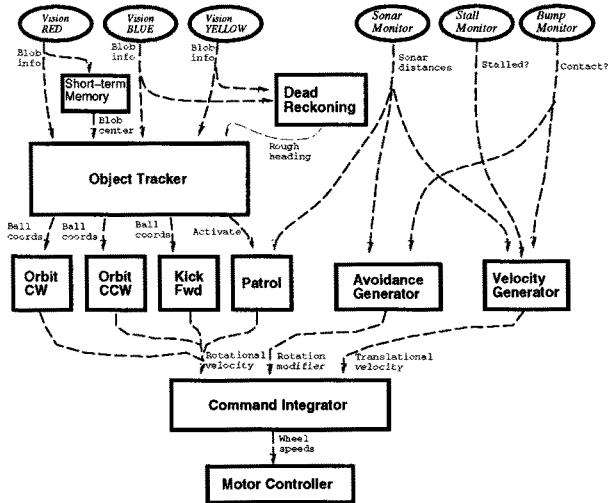


Fig. 3. The control system of the fielders. Boxes are individual behaviors; ovals are perceptual devices; and links are labeled with the type of information they carry.

degrees of “north”, 180-degree distinction of east and west) and a rough calibration of the dead-reckoning whenever either of the goals is seen with some degree of confidence results in robust orientation.

Behavior Tables and Short-Term Memory The basis behaviors are implemented as simple tables which divide the field of view into a 5x5 grid; each section of the field of view has an associated rotational velocity (Figure 4). When the ball leaves the field of view, the short-term memory (Figure 3) feeds the most-recently-viewed ball position into the *Object Tracker* for a period of a few seconds. With this in mind, we have built “recovery behaviors” into the edges of the behavior tables. When the ball

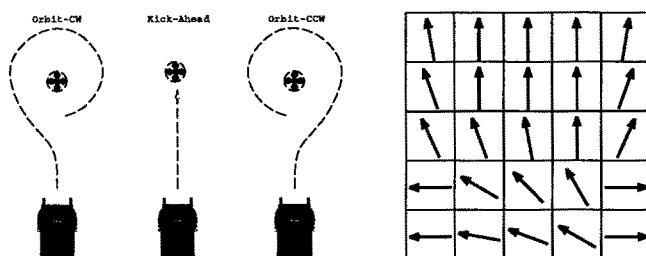


Fig. 4. The three basis behaviors sufficient to generate ball-manipulation for soccer play. Each is implemented as a table that assigns rotational velocities to segments of the visual field, such as the one shown for *Orbit-CW*.

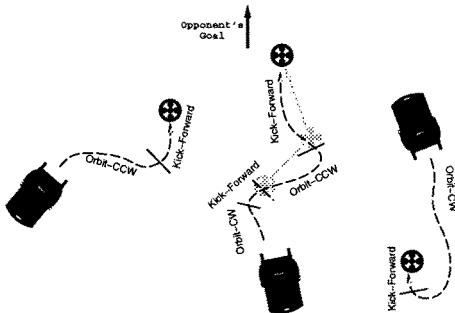


Fig. 5. Some typical trajectories generated by combination of the ball-manipulation basis behaviors.

does leave the robot's field of view, the robot takes an action which is very likely to cause it to regain direct sight of the ball. This allows the robots to react to an unseen ball without explicitly calculating motion of the ball within some model of the environment, though in practice it *appears* that the robot is doing just that. The same effect is what allows the *Orbits* to function. As can best be understood by imagining the ball at various positions in the table in (Figure 4), the robot actually "tacks" around the ball as it loses and regains sight of it. This tacking behavior also causes it to continuously change its angle so that it is very likely to line up for a switch to *Kick-Ahead* behavior during the interpolation process.

Pushing, Batting, and Kicking Pushing is the simplest of the manipulations; when the ball is directly in front of the robot, the *Kick-Ahead* behavior moves the robot forward as fast as possible, pushing the ball and making small corrections until such time that the perceptual situation and *Object Tracker* determine that major (*Orbit-based*) corrections are necessary. Batting occurs when the ball is stuck against a wall or being advanced by an opponent; the combination of tacking behavior inherent in interpolation and reduced speed due to the *Velocity Generator*'s perception of the wall or opponent causes the robot to swing back and forth more forcefully as it gets closer to the ball, always remaining on the proper side of the ball for interpolation. This results in the ball being batted (by the side of the gripper) towards the opponent's goal. Kicking results from the observation that in certain configurations, there is no way that the robot can push or bat the ball productively without backing up (which would involve complex, sequenced behavior). This occurs when the ball is right next to the front of the robot (at the lower corners of the visual field) where forward motion would at best knock it to the side (Figure 6). When the ball is in such a position, the robot rotates at high speed away from it, loses sight of it, and thus continues rotating (due to short term memory) for a full rotation. Since the robot is oblong, its rear-end often "kicks" the ball smartly forward.

3.2 Safety Behaviors and Motor Command Integration

The *Velocity Generator* outputs a speed inversely proportional to the distance to the closest object detected by the sonar; the *Avoidance Generator* outputs rotational ve-



Fig. 6. The proper configuration for a rear-end kick.

locities that cause the robot to turns away from objects to its sides. The *Command Integrator* combines the rotational velocities from the ball-manipulation behaviors and the translational velocity from the *Velocity Generator* to form final wheel-speeds. The *Avoidance* rotations override the manipulation rotations only when necessary. The *Velocity Generator* monitors stalls and gripper contact points to detect situations where the sonars do not properly register collisions.

4 Team Behavior

Team behavior arises from the interaction of obstacle avoidance, ball manipulation, and short-term memory. In a manner akin to the Dispersion basis behavior used in [4], fielders move away from objects close to their sides. This dispersion leads to effective offensive and defensive formations of some sub-set of the team when combined with ball manipulation behaviors. When advancing towards the opponent's goal, team members fall into the configuration seen in Figure 7: one robot leads with the ball, while others stay to the side and behind, ready to recover the ball should it be fumbled or stolen. The robots on the sides stay slightly behind because they slow down as they draw abreast of the ball and turn more sharply towards it, perceiving the leader with sonar. It is not uncommon that possession of the ball changes between members of the advancing group as it moves. The number of robots participating in the group is limited by perception; after some critical mass (three, in practice), robots on the edge of the group tend to lose perception of the ball for longer than the short-term memory duration and take up a Patrol behavior.



Fig. 7. Offensive (left) and defensive (right) group formations.

In a defensive situation the ball is not advancing, so the robots fall into a semi-circular arrangement around it. This effectively prevents the opponent from dodging the defense, and places players in a good position to gain possession of the ball. The offensive or defensive response is triggered solely by the behavior of other agents and the ball, with no need for symbolic distinction of teammates from opponents.

Given the lack of information about opponent strategies, we felt that the benefits of homogenizing the fielders outweighed those of specializing them for field positions. To prevent clustering, however, it proves effective to vary the direction of rotation of the Patrol behavior, which is implemented quite simply as a fixed rotational velocity. Robots with different directions of rotation tend to wander separate "territories," due to effects of safety behaviors during interactions.

5 Summary of Control System

The "Spirit of Bolivia" demonstrates an outstanding level of behavioral sophistication in the RoboCup mid-size field, including polite "sportsmanlike" behavior towards opponents, constructive interaction with teammates, and ability to continuously manipulate the ball both offensively and defensively. This is all done with almost no internal state (only the dead-reckoning position used when neither goal can be perceived, and the short-term memory), no explicit communication, and with extremely simple code: the most complex behavior is the *Object Tracker* presented in pseudocode in Section 3.1, and the entire system occupies fewer than four pages of LISP code. The system has redundancies (e.g., dead-reckoning), seamless recovery procedures (e.g., edges of basis behavior grids), and a minimal need for accurate information that all lead to extremely robust, purposeful, and lively behavior.

6 Performance at RoboCup '97

Though not one of the co-champions, the "Spirit of Bolivia" was the only team at RoboCup whose defense proved impenetrable (no points were scored against the team). The team did not have a chance to play against either of the championship teams during the competition, but in an unofficial game shortly after the finals, the "Spirit of Bolivia" won against one of them, ending up as the only team to have won against either of the co-champions, who consistently tied against each other. Given that our team was the only one attempting obstacle avoidance, which had a major impact on the robots' speed and ability to maneuver, we consider the "Spirit of Bolivia" to be a great success.

Acknowledgements

We would like to thank all the people who were so generous in lending us their precious robots, including everyone at ActivMedia, RWI, and Newton Labs, the organizers of RoboCup for an excellent - and difficult! - job, and all those intrepid souls who dealt with the daunting task of getting the robots to Japan and back, especially Merri Naka, Jon Reese, and Anne Wright for the slick airline cargo idea...

References

1. Beckers, R., O. Holland, and J. Deneubourg. From Local Actions to Global Tasks: Stigmergy and Collective Robotics. Proceedings of the 4th International Workshop on the Synthesis and Simulation of Living Systems, 1994.
2. Brooks, R. Intelligence Without Reason. Computers and Thought - IJCAI 91.
3. Donald, B., J. Jennings and D. Rus: Information Invariants for Distributed Manipulation. International Journal of Robotics Research, **16**(5), 1995.
4. Mataric, M.: Issues and Approaches in the Design of Collective Autonomous Agents. Robotics and Autonomous Systems **16**, 1995.
5. Sargent, R., B. Bailey, C. Witty, and A. Wright: Dynamic Object Capture Using Fast Vision Tracking. AI Magazine **18**(1), 1997.
6. Smithers, T.: On Why Better Robots Make it Harder. Proceedings of SAB-94.
7. Werger, B.: Principles of Minimal Control for Comprehensive Team Behavior. Ulanta Performance Robotics Technical Report #97-02
8. Werger, B. and M. Mataric. Robotic Food Chains: Externalization of State and Program for Minimal-Agent Foraging. Proceedings of the 4th International Conference on Simulation of Adaptive Behavior (SAB '96).

AT Humboldt — Development, Practice and Theory

Hans-Dieter Burkhard, Markus Hannebauer, Jan Wendler*

Institute for Computer Science
Humboldt-University of Berlin
10099 Berlin, Germany

Abstract. This article covers three basics of our virtual soccer team AT Humboldt: We describe our development process in the frame of a practical exercise for students. The resulting efficient agent-oriented realization is explained, and we give a theoretical embedding of our planning component based on BDI.

1 Introduction

One of the recent fields of Artificial Intelligence is *Agent-Oriented Programming* (AOP, cf. e.g. [WOOLDRIDGE/JENNINGS, 1994], [SHOHAM, 1993]). AOP is proposed especially for the programming of autonomous components (“agents”) in open heterogeneous systems. The agents (players) in Artificial Soccer [KITANO ET AL., 1997] must plan and execute actions individually and efficiently resulting in a successful cooperative team behavior.

This article covers three basics of our virtual soccer team AT Humboldt: We present a description of our development process in the frame of a practical exercise for students at the Humboldt-University of Berlin. We show how this has lead to an efficient agent-oriented realization. We also give a theoretical embedding of our architecture and our planning component.

We hope that our article can be useful for three types of readers: The practitioner may find some new ideas to enhance his own team, the teacher may get an impression of our work with students, and the theoretician may be interested in an applied *Belief-Desire-Intention* (BDI) architecture.

The article starts with the motivation of our interests in Artificial Soccer. The development process is figured out in Section 3. The main part (Section 4) describes the components and implementation of our agents and Section 5 discusses the theoretical background. Finally, future developments and conclusions are discussed.

* Many ideas and first of all hard implementation work came from Pascal Müller-Gugenberger, Amin Coja-Oghlan, Adrianna Foremniak, Derrick Hepp, Heike Müller and Kay Schröter in a practical exercise. We wish to thank them for their great efforts.

2 Our Interests in Artificial Soccer and First Experiences

The main interests in our group concern Distributed AI, Multi Agent Systems (MAS), Agent-Oriented Techniques (AOT), and Case Based Reasoning (CBR). All of these fields have direct relations to Artificial Soccer and we have fixed the following scientific goals for our participation in RoboCup:

1. *Agent architectures* — We are interested in experiments and evaluations for different approaches like the subsumption architecture ([BROOKS, 1990]) and the BDI architecture ([BRATMAN, 1987], [RAO/GEORGEFF, 1995]). This concerns the efficient realization of the “belief-to-action”-cycle and the optimal relationship between deliberative and reactive behavior.
2. *Cooperation in MAS* — We are interested in emerging cooperation and improvements by communication, negotiation and (explicit) joint plans. Further interests concern the formation of a team out of different roles and “characters”.
3. *Agent learning* — We are interested in the usage of CBR for the training of capabilities and decision procedures (off-line learning) and for the adaptation to opponents’ behavior (on-line learning), respectively. Especially the treatment of time in these two cases is challenging.
4. *Decision making using vague information* — We have developed a special technique for vague matching which we use in CBR applications (“Case Retrieval Nets”, [LENZ/BURKHARD, 1996], [BURKHARD, 1997]). We want to test it as control structure for the mind of agents (e.g. for the choice and adaptation of precompiled plans using similarity of sensor information/belief conditions).

The implementation of our program until August 1997 has covered mainly point 1 and partially point 2. We have discussed different approaches and finally we have implemented an architecture which is best comparable with the BDI-approach (cf. section 5). The *Advanced Skills* (cf. section 4.4) could also be considered as a layer in a subsumption architecture. Because the higher levels of deliberation have continuous control, it is a more goal oriented architecture in our understanding. Under this view, the *Advanced Skills* are considered as predefined plans.

Emerging cooperation has already given very exciting results: players act according to their expectations concerning the behavior of their team-mates. Different roles result in an efficient usage of the whole playground. But we have also identified several situations where communications would improve the behavior (but this was not implemented up to now).

The realization of Point 3 is supported in the architecture by the history mechanism within the world model (cf. section 4.2), but it was not used for learning up to now. There was some misunderstanding in the announcements: We have declared CBR as our research goal for RoboCup (hoping to get ready with it until the competition in Nagoya). But in fact we did not use it in RoboCup97. We have observed in our studies that learning may lead to suboptimal (or even

worse) behavior if it is applied to insufficiently analyzed/developed underlying skills. To give an example: A directed kick needs some preparation (stop the ball, place the ball for kicking, final kick). Hence the players have to learn a related *sequence* of parameterized actions. A careful analysis can specify a skeleton for a successful skill, while parameter settings according to a given situation are due to training (as for humans). Another example concerns the decision component: You can learn proper deliberation only if you can rely on the proper execution of the skills.

The Case Retrieval Nets (Point 4) seem to be suited for the learning of deliberation processes, but we have no experience up to now.

3 The Development

The overall development strategy was to keep in mind all our interests from the very beginning (e.g. histories for learning, desires and intentions for deliberation). That means to develop a structure which is open for further refinements and extensions. We also tried to obtain best results with least effort (e.g. cooperation without communication as far as possible).

We have used protocols (log-files) of internal and external information flow and decisions. The study of those files gave us hints to inexact implementations of our ideas (but it also occurred that "wrong" executions were caused by net overload).

We started with basic work on a class library for UDP/IP, sensors and commands. First ideas were developed by prototyping (extended "simple client"). The development of soccer agents was the topic of a practical exercise for students in connection with a lecture on MAS/AOP/CBR during summer semester (April – July 1997). Besides the discussion of concepts some modules and routines were implemented in C++ and JAVA.

Starting in the second half of July a team of three students finished the work on the implementation of the modules (only in C++ for reasons of computational speed). The modules were combined and tested. The first results showed approximately equivalence to the power of the "Ogalets". After fixing some bugs, tuning and further refinements we could improve the scoring to 29:1 before leaving for Nagoya.

Some further changes during the breaks of the competition in Nagoya concerned especially positioning (fuzzy positioning, positions for corner kick and goal kick, changes in the home positions).

4 The Ideas of the Realization

This section gives an overview and some details of the different components of our soccer playing agent team. Additionally it may help the reader to get through our published code²: We will mention the particular files which implement the

² (see http://www.ki.informatik.hu-berlin.de/RoboCup97/index_e.html)

described parts. The whole project was implemented in C++ under Solaris and Linux, respectively.

4.1 Clear Architecture

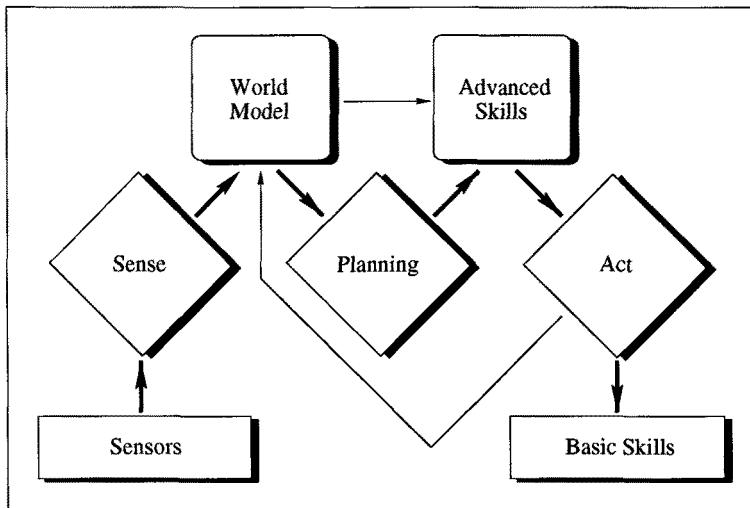


Fig. 1. General Architecture of a Soccer Agent.

The general architecture is identical for all agents, the player identifies its role (goal-keeper, defender, attacker ...) by the player-number given from the server. Special behaviors according to the roles are due to different parameter values. Figure 1 shows the agent's abstract architecture. Thick arrows indicate the main information flow. Rhombs symbolize concurrently running components. An agent (agent.*) consists of all these components which can be seen as "organs".

The component *Sensors* (*sensors.**) parses and transforms the string coded sensor information which the player receives from the SoccerServer. The component *Basic Skills* (*baseskills.**) provides methods for sending basic commands to the server.

4.2 Stable World Modelling

In a dynamic and uncertain domain like Artificial Soccer a consistent modelling of the environment is necessary. Short-term false information has to be corrected, un-precise information must be evaluated and inferences are necessary for missing information. This leads to a certain stability of the agent's belief. It is realized by the complex component *World Model* (*weltmod.**, *fussball.**, *positions.**, *sensori.**, *spielfe.**), which provides e.g. basic classes for

linear algebra. Every object on the field is described by a special class. Inheritance is strongly used, and additional features like timed objects and encapsulated environments make synchronization easier.

The agent's absolute position on the field is calculated using the visual information concerning lines, flags and goals. The triangulation considers all possible cases (actually several hundred). The agent's velocity is estimated from the commands sent to the server. Additionally, the player records its stamina. Absolute positions of all other seen objects can be computed because the own absolute position is known. A special algorithm matches new information on unnamed objects (e.g. a player with a missing number) to known objects. The world model can also close the information gaps for unobservable objects by simulation.

Simulation is also used to predict future situations by using the knowledge about positions and velocities. This ability is extensively used by *Planning* and *Advanced Skills* to estimate consequences of possible commands. For example, *Advanced Skills* can instantiate a new ball object, simulate it for some time steps and look at the position and speed. Additional features like wind could be easily taken into account.

World Model logs an adjustable number of environments to keep track of the player's history. This ability was implemented to support on-line learning, but it was not exploited in RoboCup97.

4.3 Bounded Rationality

The component *Planning* (*entscheidung*.**) embeds the planning and reasoning process which leads successively from coarse plans to concrete command sequences. The theoretical background of this procedure is described in section 5. Here we show how the internal process works.

A new planning process is initiated each time a new sensor information has arrived. The situation is classified: If the ball is under control, the agent is able to pass the ball or to dribble. If the player has no control of the ball, it can decide whether to intercept it, to watch the surrounding or to run to a certain position. This goal (target) finding is done by a usual decision tree which selects a goal out of a fix goal library. Some of the decisions are trivial ("Is the ball in the kick range or not?") but some are really tricky ("Shall I run to the ball or shall my team-mate do it?"). The latter decision is done by a reachability simulation: If the agent supposes to be the first of its team to reach the ball, it will run. If not, it relies on its team-mates and runs back to its home position. Because we have not used communication yet, the stamina of team-mates has not been regarded and may have caused wrong expectations.

After the goal selection the player has to find the best way to achieve its goal. This phase of the planning process produces a coarse long-term plan with some parameters (*partial plan*). The plans must be based on the skills of the agents. This means that plans are in a close correspondence to the *advanced skills*, which are themselves building blocks for the execution of the long-term plans. There are two major cases to cope with: The agent is out of the kick range or it can control the ball (i. e. has ball possession).

In the first case the player calculates an optimal interception position if it has decided to get the ball. For several reasons (e.g. to regard the wind if necessary) we decided to utilize the simulation capability of the world model. The agent tries mentally to reach the ball in one step, in two steps and so on until it finds a certain number of steps in which it can reach the ball. This procedure also provides a distance measure because it can be applied to every player and ball instance. Figure 2 shows this process graphically and gives a commented example.

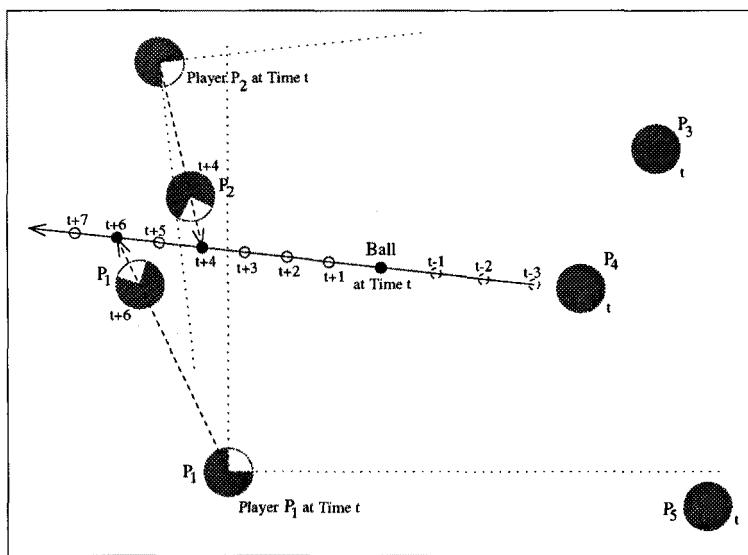


Fig. 2. Decision Finding for Ball Interception.

The solid line shows the ball movement with the ball positions at the time steps $t \pm i$. The dotted lines represent the view sectors of the different players P_1 and P_2 at time t . The dashed lines show the movements of the players.

P_1 does not see P_2 and P_5 . It calculates distance 26 for P_4 , 29 for P_3 and 6 for itself. The agent decides to intercept the ball. P_2 whose view sector includes P_1 calculates 6 for P_1 , 30 for P_3 , 26 for P_4 , 39 for P_5 and 4 for itself. It also decides to go for the ball. Hence both players of the same team go for the ball.

P_1 calculates interception position at ball position $t+6$. Likewise does P_2 for ball position $t+4$. At time $t+3$ a new sensor information comes in. P_1 cannot see the ball. It will keep its plan according to the implementation of our planning component for such situations. P_2 sees the ball and also continues intercepting the ball.

If the player has decided not to intercept the ball, it returns to its home position, or (if it is already there) collects information by turning and waiting. Further development will lead to more sophisticated behavior (e.g. double passes and explicit team strategies). The architecture is prepared for such extensions

(we have implemented special behavior for corner kicks or goal kicks just in the breaks of the competition in Nagoya).

If the player controls the ball, it has to decide whether to pass the ball or to dribble. Furthermore it has to decide in which direction to kick or to dribble, respectively. It should prefer a direction with best chances to score or to pass the ball to a team-mate. At the same time it should prefer directions promoting an offensive play style. A fixed direction d is evaluated by the following formula:

$$\Delta(d) = \omega_b(\beta_t(d) - \beta_o(d)) + \omega_m(\mu_t(d) - \mu_o(d)) - \gamma_o(d) + \gamma_t(d)$$

The indexes t are used for terms indicating values of the own team, indexes o for the values of the opponents, respectively. ω_b and ω_m are role dependent weight factors with $\omega_b + \omega_m = 1$.

β is the minimum of the distances δ to the ball for all players of a team (indexes t or o). The distances δ are calculated by simulation as described before, if the ball is kicked in the given direction d with a certain velocity. It is computed for the own team by:

$$\beta_t(d) = \min \left\{ \sqrt{\delta(d, i) \cdot \lambda(d, i)} \mid i \in S_t \right\}$$

S is the set of seen players. The additional factor λ is the length of the perpendicular from player i to the ball line (this value provides an influence of stamina loss).

μ provides the mean distance of a team to the ball line. The unseen players are approximated using a worst distance δ_w (P is the set of all team players):

$$\mu_t(d) = \frac{\sum_{i \in S_t} \left(\sqrt{\delta(d, i) \cdot \lambda(d, i)} \right) + |P_t \setminus S_t| \cdot \delta_w}{|P_t|}$$

γ is a goal hitting bonus, where δ_k is the “typical” kick distance and δ_g is the distance to the goal, respectively:

$$\gamma_t(d) = \begin{cases} \max \left(0, \delta_w - \frac{\delta_w \cdot \delta_g}{\delta_k} \right), & d \text{ hits our goal} \\ 0 & , \text{ otherwise} \end{cases}$$

The values for ω_b , ω_m , δ_w and δ_k have to be tuned (it is intended to use learning). β_o , μ_o and γ_o are defined analogously.

Having the formula to evaluate directions d by their values $\Delta(d)$ we can look for an optimal direction. Due to the definition (convex combination by weight factors, symmetrical values for opponents’ players and team-mates) the evaluation function is normalized to zero. This means that negative values indicate a “good” direction and positive values indicate a “bad” direction. In our recent implementation several discrete directions are evaluated and then the best direction is taken. Figure 3 exemplifies such an evaluation process.

Values around zero are neutral. Especially these values around zero are difficult to judge. For this purpose a randomized function was implemented which decides afterwards whether to kick or to dribble.

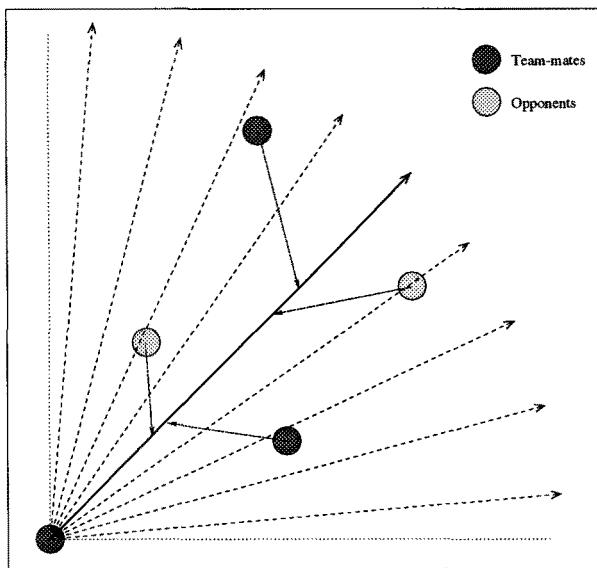


Fig. 3. Evaluation of Discrete Kick Directions.

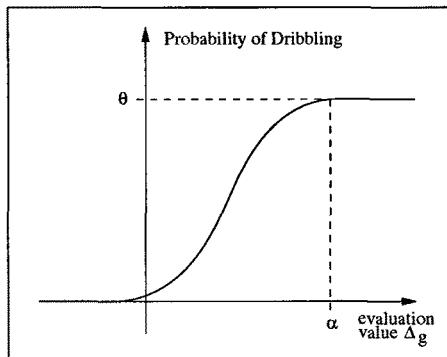


Fig. 4. Probability function for Dribbling.

The function is shown in figure 4. It is on the left half a Gaussian function with the following definition:

$$P(\text{Dribble}) = \begin{cases} \theta \cdot e^{-\frac{(\Delta_g - \alpha)^2}{3.38}}, & \Delta_g \leq \alpha \\ \theta, & \Delta_g > \alpha \end{cases}$$

θ is a role dependent dribble factor (e.g. the goal-keeper has $\theta = 0.1$, the attacker $\theta = 0.5$). Δ_g is the minimal distance over all evaluated directions. α is an acceptance constant for kicking which is also role dependent.

The main problem of the evaluation strategy described above is that players move in unpredictable ways. Therefore the positions of unseen players are

uncertain. This is why we decided to evaluate only directions in the view area (we used a view angle of 90 degrees and high quality). To prevent too many backward shots, every area on the field has a defined preferred direction. Our first idea was the following procedure: Turn to preferred direction and evaluate the kick directions. If there is no “good” kick direction, determine a turn direction and turn, evaluate this sector and so on. This was a safe way to find the global best direction, but was actually to slow because the agent had to wait 300 ms to get the next information. So we implemented the following behavior: If the agent looks approximately into the preferred direction (the definition of “approximately” is role dependent), it will do the evaluation process. If not, it will do an “emergency kick” directed to the opponents goal. With additional information via communication this simple behavior could be omitted in favor of full 360 degree evaluation in the future.

Cooperation between team partners emerges by relying on the behavior of team-mates, that means team-mate modelling. The player relies on the fact that the team-mate with the shortest distance to the ball will try to intercept it when it is passed in its direction.

As soon as the agent has decided for a partial plan, it is given to the component *Advanced Skills* for execution.

4.4 Efficient Execution

The component *Advanced Skills* (`advancedskills.*`) is a library of skills which facilitate efficient ball handling and optimal movement. The technical task of this component is to split the long-term partial plans into short-term plans, which means concrete command sequences with full parameters. These short-term plans are not longer than the interval between consecutive sensor information (with our preferences these are actually three basic commands). This way the long-term plans are executed by iterated calls of advanced skills after each sensor information.

It was one of the major decisions during development to use this strategy of plan execution. The more common strategy is to fix a long term plan which may be adapted during execution if necessary. Such a long term plan can start with some initial actions to achieve a well defined situation (e.g. a suitable ball position for dribbling). Afterwards the actions are performed in the fixed sequence according to the plan, such that each action relies on the successful execution of its predecessors.

In our strategy, a strictly new deliberation process can start for each new sensor information (actually each 300 msec), and in this case we have a new long term plan started just at this time point. If we would need always certain (new) initial actions for preparation, then we might never come to the continuation of a plan (cf. Section 5 for further discussions). To overcome this problem, the advanced skills are designed to deal immediately with any situation which might appear at the beginning or during the execution of a long term plan (e.g. to continue dribbling in any situation). As a side effect, the advanced skills are

able to realize the fastest way for goal achievement in a very flexible way from arbitrary start situations.

Advanced Skills uses *Basic Skills* to send basic commands to the server. We modified the basic *Turn* command because the original command which is directly supported by the server is influenced by the player's speed. We enhanced the *Turn* command to compensate this influence by increasing the turn angle or — in worst case — stopping first and then turning. The other basic commands are sent without modification to the server.

The main advanced skills of the player agent are: *Directed Kick*, *Go to Position* and *Dribble*. *Go to Position* enables the agent to reach every absolute position on the field. It produces one *Turn* (if needed) and/or up to two/three *Dashes*. If demanded, this procedure avoids obstacles like other players. *Dribble* moves the ball into a certain direction without loosing contact to it. This includes the production of several *Kick*, *Turn* and *Dash* combinations.

The *Directed Kick* skill was one of our competitive advantages and therefore it will be described in detail in the following. This capability allows the players to kick the ball into any direction with a demanded power (as far as possible). It handles difficult situations like high velocities and situations where the player itself is an obstacle for the desired direction. If the desired direction with the desired speed cannot be achieved, the skill tries to meet the demands as good as possible.

First of all the skill tries to determine the kick angle and power which is necessary to transform the current movement vector into the demanded movement vector (Part A of figure 5). If the length of the necessary kick vector (the power) is physically impossible, the skill tries to keep at least the right direction.

A complication occurs for kick vectors which are possible but hit the player itself. In this case an intermediate target is calculated which is at the side of the player (Part B). The first kick leads to this point and further kicks are calculated from there (Part C). In some cases the ball can be kicked once more (Part D).

This leads to the efficient kicks which were observable in the matches of our team. All this can be done with three basic kicks: The implementation of the SoccerServer for Nagoya permits addition of velocities by consecutive kicks following the laws of vector algebra. According to the Nagoya settings, the players can start with approximately 10 m/sec and the ball with 13 m/sec. By additional dashes, the players can get a speed of approx. 16 m/sec. Two consecutive kicks in the same direction give the ball a speed of approximately 25 m/sec. which is less than two times faster than the players' speed. It is reasonable that it takes two actions (i.e. more time and more precise action settings) to make a stronger kick: The two kicks can be considered as a compound action (just like turn + dash). It can even be considered as a triplet together with the need to place the ball at a certain position before a strong kick. This setting leads to an interesting challenge for the learning of compound actions.

Problems arise for the interception of a fast running ball: Especially increasing velocity for fast moving balls by likewise small additional kicks of an intercepting player looks problematic. Furthermore, we found it difficult (but not

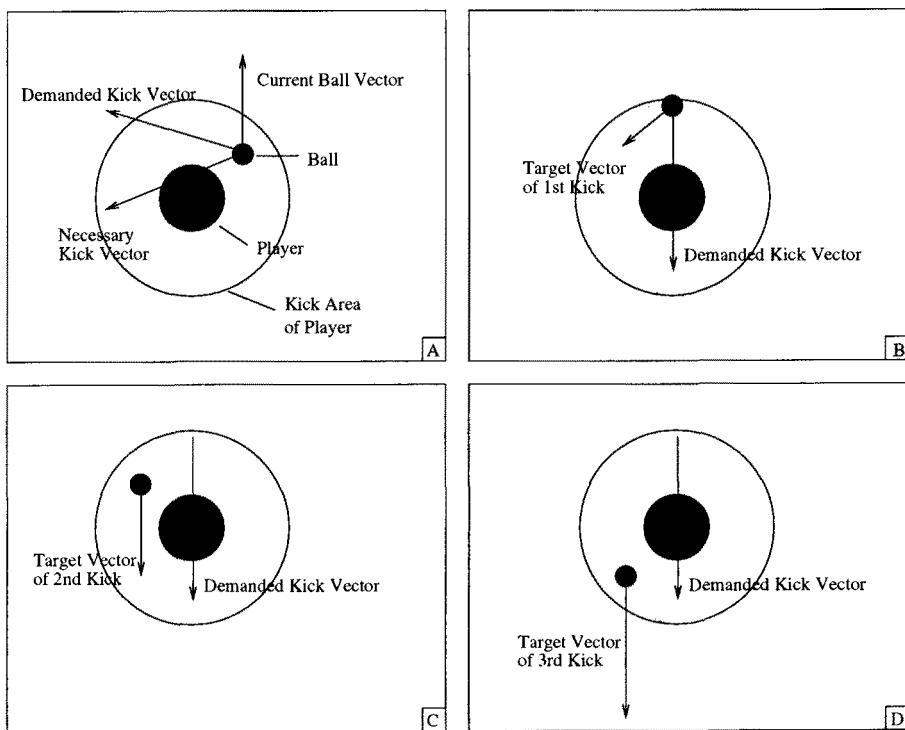


Fig. 5. Several Steps of the *Directed Kick* Skill.

impossible) to stop a ball by defending players.

Alternatively the SoccerServer could simulate kicks as new settings of the ball velocity instead of additions. In this case, the following problem arises: Setting of a fast arriving ball with a moderate single kick to a new slow velocity (defense could be too simple with such settings)³.

4.5 Precise Synchronization (system programming)

In the real time environment of Artificial Soccer it is essential to keep synchronized with the server. For this purpose some components have to run in parallel. In our implementation the components *Sense* (*sensorik.**) and *Act* (*sender.**) run concurrently with the main routine to allow an undisturbed reasoning-process and to keep track of the server.

There are different ways to provide pseudo concurrency in a UNIX environment. Processes raise difficulties with communication and storage organization. Threads are not widely supported (Solaris has native support, Linux and others have only library support like pThreads) and have a lot of administration overhead. In addition they are difficult to time. On the other hand UNIX system

³ This point is still under discussion while writing this article.

signals are efficient, well timed and easy to use. In fact one signal handler is enough to provide all the concurrency the agent needs.

The signal is adjusted to 50 ms. The signal handler looks at top of the incoming message system queue and parses new information if necessary. Accordingly information flags for the planning process are set. In addition the handler peeks every 100 ms on top of the outgoing command queue, sends one command if necessary and initiates a simulation cycle of *World Model*. If the time information of an incoming sensor information differs from the agent's internal time, additional simulation cycles are done until synchronization is achieved. This is essential because the estimation of the agent's own speed relies only on sent commands and passed time. If the calculation of the agent's own speed is wrong, then all other speed calculations will be wrong because they are known only relatively to the values for the agent.

5 The Theory

The consideration of programs as agents focuses at first on the aspect of autonomy: Programs have to act in an appropriate way to changes in the environment. Therefore they need some input or sensor facilities and some output or actoric components. The mapping from input to output can be done in very simple ways (e.g. strictly reactive) or in more sophisticated ways up to models which are inspired by human decision processes. We found that mental notions like capabilities/skills, belief, goals/desires and intentions/plans are very useful pictures to make agent programming transparent. The aspect of rationality forces agents to deal efficiently with their resources, especially with time.

The so-called BDI model fits best to our concept of soccer agents. BDI stands for belief – desire – intention, and the approach is based on the philosophical work of BRATMAN [BRATMAN, 1987], and the theoretical and practical work by RAO/GEOERGEFF [RAO/GEOERGEFF, 1995] and others (cf. e.g. [WOOLDRIDGE/JENNINGS, 1994], [BURKHARD, 1996]). Agent-Oriented Programming is a fast developing field of research, and Artificial Soccer is a very suitable field for experiments. Rao and Georgeff write about typical characteristics of problem domains which can be successfully solved by BDI. These characteristics fit well to the soccer domain:

- The SoccerServer and the opponents create a non-deterministic environment.
- The agent itself reacts non-deterministically because parts of the planning process are randomized.
- The player can have different goals at the same time, e.g. to reach the ball while covering an opponent.
- The success of the player's own commands depends strongly on the simulated environment and the opponents.
- The whole information is local and different for every player.
- The environment pushes *bounded rationality* because too deep reasoning is without pay-off in a dynamic surrounding.

In the BDI-approach, agents maintain a model of their world which is called *belief* (because it might not be true knowledge). The way from belief to actions is guided by the *desires* of the agent. BRATMAN has argued that *intentions* are neither desires nor beliefs, but an additional independent mental category. Intentions are considered as (partial) plans for the achievement of goals by appropriate actions. Commitment to intentions is needed which has impact on the rational usage of resources: The (relative) *stability of committed intentions* prevents overload in deliberation and useless plan changes, and it serves trustworthiness in cooperation.

All components of a BDI-architecture can be identified in our planning process. *Belief* equals the component *World Model* (cf. section 4.2 in our realization. The strict relativity of the server statements leads to an individual image of the world in every agent. Additionally the agent cannot rely on the accuracy of the received and interpolated data, therefore it is belief not knowledge. The update routines (including the simulation for unseen objects), the simulation of expected future situations and the history of situations are also considered to be parts of the belief.

In our implementation *Desires* are goals which are selected out of a fixed goal library. The list of possible goals is still small, but the set will be extended e.g. to allow joint goals (like double passes) in the future. In the present realization different (even opposite) goals may be achievable but the agent selects only one of them⁴. To do this the environment is classified by a decision tree. The selection function and the execution of the chosen goal must be fast because it must not be interrupted by new information which could be decision relevant. In spite of that, reasoning must be accurate enough to fulfill its task optimally. This is the same trade-off as described by Rao and Georgeff. The risk of full execution of long-term plans is that the agent cannot adapt correctly to unforeseen events. On the other hand permanent evaluation and control of every planning step is too expensive in the sense of computing resources.

This implies *Intentions* which are divided into two stages of planning in our system. At first the best possibility to reach the chosen goal is computed and fixed as an intention (cf. section 4.3). This corresponds to the long-term plan with some parameters which can be also seen as a partial plan. Its calculated end is the fulfillment of the selected goal. The execution of the intention is split into smaller pieces which are implemented as short-term plans in the component *Advanced Skills* (Section 4.4). As mentioned above *Advanced Skills* provides precompiled plan skeletons of a size that fits between two time points of sensor information. They have their own calculation capability which is used to compute the short time optimal command sequence. Looking at the mentioned trade-off these short-term plans are atomic and cannot be interfered by sensor information. But in composition they build a long-term plan that is complex enough to fulfill higher

⁴ In the future we may deal with the commitment to concurrent goals. In such a case we will have to regard the “scope of admissibility” (BRATMAN) set by previous intentions. For example, an existing intention to preserve the off-side position of an opponent may restrict the commitment to later goals for reaching special positions.

goals. There is a certain overlapping with the decision procedures for desires: This is necessary, since the decision process has to look for achievable desires. The realization of the intention relies on the capabilities of the agent, which are implemented by the advanced skills.

The consideration of our agents as BDI-constructs is appropriate since we have for each new sensor information a complete deliberation process with update of belief, choice of a desire, commitment to an intention and execution of a plan part.

A problem arises from the fact that commitment of intentions is mostly performed independently from the previous intentions: This might contradict the mentioned principle of stability of committed intentions, which is a central point in BRATMAN's theory. The "canonical" deliberation process has to maintain an old intention as long as there are no serious counter indications.

Because a new deliberation process might be initiated every time a new sensor information comes in and then new plans are created, the planning strategy has to ensure stability of the long-term plans to avoid constantly changing goals or intentions. The stability of our goal tracking relies on the fact that even in the case of a new initialization of the whole planning process very often the same steps are chosen because the situation has not radically changed. That means that the same goal and intention are created, too. The player must prevent that missing knowledge or only slightly better other intentions destroy the former behavior. Indeed, a simple implementation of our strategy would have serious drawbacks. As already mentioned in Section 4.4, simple plans could result in only repeated initial actions. To avoid this the agent for example lets out minor turns on the way to a certain position which would cost too much time. This could be called implicit persistence.

The explicit persistence of goals and intentions in our implementation can be exemplified by the realization of the goal "Go to home position". To decide whether to intercept the ball or to go home the agent has calculated the minimal distance of all team-mates and opponents to the ball (cf. section 4.3) and has stored these values. If the decision is to go home, the agent will use these values to determine a time interval in which it must not care for the ball because no other player will be able to change the ball's known movement. The decision tree usually strongly relies on sight of the ball but in this case the agent won't turn for the ball in the calculated "don't care"-interval on its way to its assigned position. This results in a straight run to the designated position until the "don't care"-time will be over. In general that means that the old goal and intention is kept as long as the calculated interval lasts.

We found this to be a very interesting implementation of the stability principle for committed intentions without explicitly using the old intention. Our agents are able to adapt a plan to new situations if necessary (e.g. a turn-command with a greater change would not be dropped). It might be the case that future implementations would need an explicit treatment of previous intentions (e.g. if there was a commitment given to team-mates in explicit cooperation).

6 Future Work

The next steps to improve our players concern various forms of training and learning as well as other methods of cooperative play:

- Some decisions and skills use individual parameters which values were found by testing evaluation. These parameters shall be tuned by the agent itself.
- Methods from Case Based Reasoning can help to reuse “good” decisions in equal situations.
- Learning means training of behavior with off-line learning as well as adaption to the opponents’ behavior in the match (on-line learning).
- The team play shall be improved by a special behavior in well-known situations, explicit cooperative skills and use of the provided communication on the field.

7 Conclusion

The following lessons have been learned during our discussions and implementations:

- An architecture which makes the agent processing transparent is important for development of concepts, implementation and fast changes.
- An efficient implementation needs the integration of methods from different fields, especially Mathematics, Software Engineering and Artificial Intelligence.
- Methods from Artificial Intelligence are efficient if applied to well performing basic behavior.

The last experience was very important to us. It showed us that “pure” Artificial Intelligence may be insufficient. In our development phase sometimes a better solution from AI view has led to less performance. The reason were basic mistakes which were increased by the sensitive AI techniques. As a result we think that learning can only be based on a strong and correct conventional foundation.

We have published our code to make transparent our ideas (we have to apologize for some “unorthodox” program parts, we hope to present a better version in the future).

We want to express our thanks to all people which are engaged in RoboCup for giving us so much fun. Special thanks are due the Japanese colleagues for organizing RoboCup 97!

References

[BRATMAN, 1987]

M. E. Bratman: Intentions, Plans and Practical Reason.
Harvard University Press, 1987.

- [BROOKS, 1990] R. A. Brooks: Elephants don't play chess. In P. Maes (ed.): Designing Autonomous Agents. MIT press, 1990.
- [BURKHARD, 1996] H. D. Burkhard: Abstract goals in multi-agent systems. In W. Wahlster (ed.): 12th European Conf. on Artificial Intelligence (ECAI96). 524 – 528. John Wiley & Sons, 1996.
- [BURKHARD, 1997] H. D. Burkhard. Cases, Information, and Agents. In P. Kandzia, M. Klusch (eds.): Cooperative Information Agents. Proc. First Int. Workshop CIA'97. 64–79. LNAI 1202, Springer, 1997.
- [KITANO ET AL., 1997] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, Minoru Asada. The RoboCup Synthetic Agent Challenge 97. In M. E. Pollack (ed.): Proc. IJCAI-97. 24–29. Morgan Kaufmann, 1997.
- [LENZ/BURKHARD, 1996] M. Lenz and H. D. Burkhard. Lazy propagation in case retrieval nets. In W. Wahlster (ed.): 12th European Conf. on Artificial Intelligence (ECAI96). 127–131. John Wiley & Sons, 1996.
- [RAO/GEORGEFF, 1995] A. S. Rao and M. P. Georgeff: BDI agents: From theory to practice. In V. Lesser (ed.): Proc. of the First Int. Conf. on Multi-Agent Systems (ICMAS-95). 312–319. MIT Press, 1995.
- [SHOHAM, 1993] Y. Shoham: Agent oriented programming. 60:51–92. Artificial Intelligence, 1993.
- [WOOLDRIDGE/JENNINGS, 1994] N. R. Jennings and M. Wooldridge: Proceedings of the ECAI-94-Workshop on Agent Theories, Architectures, and Languages. LNAI 890, Springer, 1994.

Refinement of Soccer Agents' Positions Using Reinforcement Learning

Tomohito Andou

Department of Mathematical and Computing Sciences,
Tokyo Institute of Technology
`andou@is.titech.ac.jp`

Abstract. This paper describes the structure of the RoboCup team, Andhill, which won the second prize in the RoboCup97 tournament, and the results of the reinforcement learning in which an agent receives a reward when it kicks the ball. In multi-agent reinforcement learning, the trade-off between exploration and exploitation is a serious problem. This research uses observational reinforcement to ease the exploration problem.

1 Introduction

Conventional reinforcement learning research assumes that the environment is based on Markov decision processes (MDPs) and that the problem is small enough for the entire state space to be enumerated and stored in memory. From the practical viewpoint, however, reinforcement learning should be able to approximate to the optimal policy in complex problems or large problems with high frequency, even if it is not guaranteed for convergence to optimality. Actually, recent researches often tackle complex problems and/or large problems [1].

RoboCup (The World Cup Robotic Soccer) [3], of which the first tournament was opened in August 1997, is a suitable problem to test reinforcement learning methods. In the RoboCup simulator league, software agents play a game in a virtual field provided by Soccer-server [5]. An agent receives perceptual information and executes actions through Soccer-server. Generally, this problem includes the following difficulties:

- **Multi-agent:** Each team consists of eleven agents. Team play is required, but sharing memory is not permitted by the RoboCup simulator league rule except “saying” in the field.
- **Incomplete perception:** An agent can not look over its shoulder. Soccer-server gives sensor information including some errors. Agents should be robust.
- **Real-time:** A match is uninterrupted. Agents must choose actions in a short period.

Furthermore, the following difficulties should be overcome when reinforcement learning is applied to this problem:

- **Partially observable Markov decision processes:** Incomplete perception causes POMDPs.
- **Dynamic environment:** The environment is not stationary, and will be changed gradually because team-mates learn simultaneously.
- **Large state space:** The state space is too large to be stored in memory. A large state space requires generalization by function approximators.
- **Distribution of reinforcement signals among team-mates:** The aim of a team is to win the game, but reinforcement signals should be defined in more detail. Various kinds of reinforcement signals are possible.
- **Trade-off between exploration and exploitation:** How an agent behaves during the learning is an important factor. An agent follows its best policy in exploitation oriented learning, and tries wide policies in exploration oriented learning. Multi-agent learning makes this problem more serious.

In this research, we have built a RoboCup team, Andhill, which improves agents with on-line reinforcement learning and refines the position which seems the essence of team play.

2 Andhill Agents

In Pre-RoboCup tournament in 1996, there were two major types of strategies. One strategy is the lump strategy, in which most agents keep chasing the ball in the game. There was not the "stamina" element in Pre-RoboCup rule. The other strategy is the fixed position strategy. An agent reacts only when the ball is within its territory. Once the ball leaves the territory, the agent returns to its home position.

The champion team in Pre-RoboCup, Ogalets, adopted the fixed position strategy. An Ogalets agent has its inherent home position and knows team-mates' home positions. An agent who possesses the ball kicks it to the home position of a team-mate without confirmation of the team-mate's existence. Other strong teams in Pre-RoboCup were also based on the fixed position strategy.

The fixed position strategy is successful because agents behave more effectively than the lump strategy's. In RoboCup97, the "stamina" element is introduced, therefore the fixed position strategy becomes more advantageous because agents don't need much stamina.

This research, however, dares to aim at a better strategy, the dynamic position strategy in which an agent decides its position by itself. The dynamic position team can fit its formation to the opponent team's formation. The structure of Andhill agents is described from Section 2.1 to Section 2.5.

2.1 Fundamental Structure

An agent normally receives a visual message per 300 milliseconds and can execute an action per 100 milliseconds. An agent of other teams in RoboCup97 seems to execute a series of actions (turn – dash – dash, kick – kick – kick, and so on) after it receives a message. Andhill uses timer interruption to ensure executing an action in every turn. Figure 1 shows a rough sketch of agents' execution.

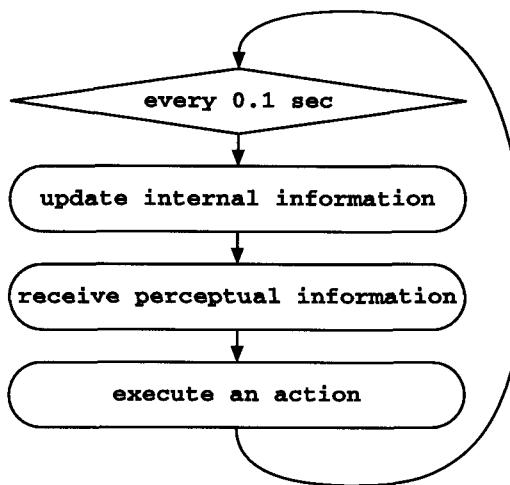


Fig. 1. The flow of an agent's execution

2.2 Internal Information

An agent receives visual information about relative distance, direction, distance-change and direction-change of each object from Soccer-server. The agent can estimate its location from information about fixed objects and the ball's future location from the ball's velocity.

Andhill agents interpret sensor information into internal information such as:

- Current time
- Location of the agent
- Time-stamp of the agent's location
- Velocity of the agent
- Time-stamp of the agent's velocity
- Direction of the agent
- Stamina of the agent
- Location of the ball
- Time-stamp of the ball's location
- Velocity of the ball
- Time-stamp of the ball's velocity
- Expected future location of the ball
- Locations of agents in sight
- The field information
- Time-stamp of the field information

Figure 2 shows an example of the field information. The field is represented by a 16×11 grid. Each unit indicates existence of team-mates and opponents.

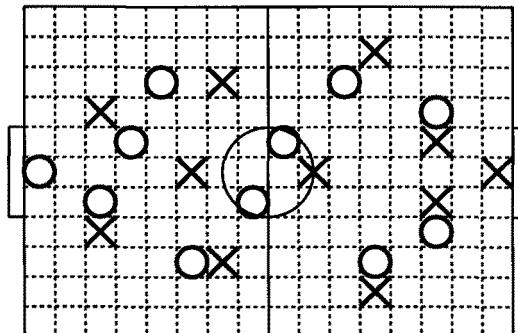


Fig. 2. An example of the field information

Agents share information about the ball's location and the field information using “say” command among team-mates.

2.3 Action Flow

This section describes the detail of the “execute an action” module in Figure 1.

1. **Get sensor information if internal information is old:** If the time-stamp of the agent’s location or the ball’s location is old, the agent turns around according to its visual range.
2. **Kick the ball if possible:** If the ball is within the agent’s kick range, the agent passes, dribbles or shoots. See Section 2.4
3. **Chase the ball if the agent is the closest to the ball:** If the agent is the closest agent of team-mates to the expected future location of the ball, it runs to the place. To avoid all agents’ holding back, the agent runs to the place if the distance is roughly equal, too.
4. **Otherwise return to the agent’s home position:** If the above conditions are not satisfied, the agent returns to its home position. See Section 2.5

Additionally, an Andhill agent “says” its internal information to team-mates periodically.

2.4 Kick Direction

The most important thing for soccer agents when it can kick the ball is to send the ball to a safer place. So the definition of the safety of the place is needed. The safety of the place seems to depend on the distribution of agents and the distances to the both of goals.

In Andhill agents, the safety of the place is determined by the safety function whose input values are the following elements:

1. Distance to the goal of the agent's side
2. Distance to the goal of the opponents' side
3. Distance to the closest team-mate
4. Distance to the closest opponent

The safety function is a neural network shown in Figure 3. When the input values are the distances from the ball's location, the output value indicates the safety of the ball's location. The network is only hand-adjusted but the network represents that the closer place to the goal of the agent's side is the more dangerous place, the closer place to a team-mate is the safer place, and so on.

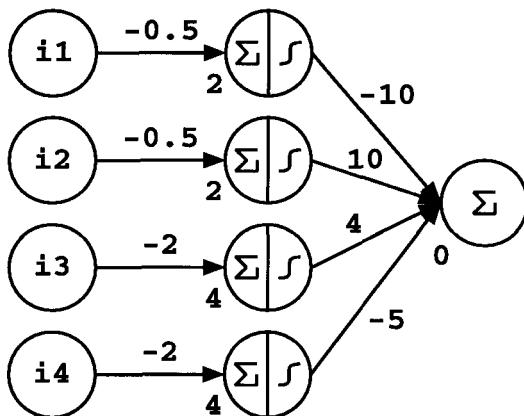


Fig. 3. The safety function: Curves in hidden units denote the sigmoid functions.

Figure 4 shows an example of output values of the safety function. The circle symbols denote team-mates and the cross symbols denote opponents. The circle team defends the left side goal. The larger number means the safer place. The safer place is the more desirable place to kick to.

An agent who can kick the ball estimates the safeties of all reachable places, and kicks the ball to the safest place. Dribbling is not prepared explicitly, but the agent sometimes dribbles because the safest place is just its front. The agent shoots if it is within the shoot range.

2.5 Positions

What the agent should do when it is far away to the ball is one of the most difficult tactics to plan. An agent using the fixed position strategy doesn't move from its home position. In real world soccer games, however, a player moves around in the field incessantly because its optimal position is changed dynamically. It must be applicable to RoboCup agents.

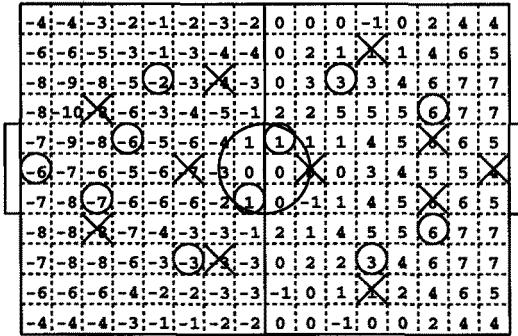


Fig. 4. The output values of the safety function: This figure omits real numbers into integer.

There are some strategies according to the player's role in a real world soccer team.

- **Goal-keeper:** The goal-keeper stays in front of the goal of its side. There are no privileged players in RoboCup97, but this strategy is good for capturing the ball because opponents kick to the place frequently.
- **Defender:** The defender's role is mainly to capture the ball which an opponent possesses. So a defender should be in a place where the ball tends to come when the ball is possessed by an opponent. Therefore, a defender usually keeps marking an opponent.
- **Forward:** The forward's role is mainly to get a goal. This task is realized only when the player's team keeps the ball from the opponents'. So a forward should be far away from opponents.

These strategies seem also useful in RoboCup agents. That is to say, the following elements are necessary to define the strategy about the position.

1. Distance to the goal of the agent's side
2. Distance to the goal of the opponents' side
3. Distance to the closest team-mate
4. Distance to the closest opponent
5. Safety of the current location of the ball

Note that which team possesses the ball is represented by the safety of the current location of the ball.

Andhill uses the position function with which the agent decides the position to go. The position function is a neural network that inputs the above-mentioned five elements and outputs the position value of the place. The network structure is shown in Figure 5. This neural network can represent the class of the XOR problem, for example, the strategy in the right side and the strategy in the left side can be opposite.

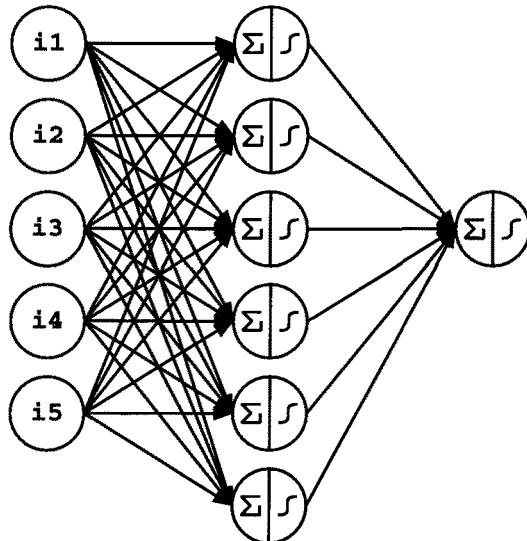


Fig. 5. The structure of the position neural network: Curves in hidden units and an output unit denote the sigmoid functions.

3 Using Reinforcement Learning to Refine the Position Function

Reinforcement learning is the machine learning in which an agent seeks to adapt to an environment by reinforcing the action which gives a greater reward (reinforcement signal). Reinforcement learning differs from supervised learning especially in its unnecessaryness of ideal input/output pairs. It requires on-line performance of the problem, but the RoboCup problem satisfies the request.

This research applies reinforcement learning to refine the position function (Section 2.5).

3.1 Local Reinforcement Signals

Reinforcement signals need to be predefined. This task is difficult but very important.

The most natural definition of reinforcement signals in the RoboCup problem may be that a team receives a reward when it gets a goal and receives a punishment when it loses a goal. The definition, however, remains the credit assignment problem. That is, how the reward should be distributed among team-mates. For example, if the reward is defined to be distributed fairly among all the team-mates, an agent who doesn't work well reinforces its useless actions. If the agent who gets a goal can receive much reward, the goal-keeper receives only little reward regardless of its important role.

In this research, which agent has worked well is defined by the number of its kicking the ball. An Andhill agent kicks the ball to a safer place (Section 2.4). So, kicking the ball can be said to be a profitable action for the team.

3.2 Exploration and Exploitation

Effective sample data can be collected only when an agent behaves consciously for the team play because other agents also learn simultaneously (exploitation). On the other hand, an agent must try various kinds of policies frequently during the learning process because the environment is not stationary and other agents' policies are changed gradually (exploration). There is a difficult trade-off problem between exploitation and exploration.

Moreover, moving the position is a high level action. Even if the position function indicates to go to somewhere, the agent can not always go to the place in time. Furthermore, moving the position costs stamina.

An Andhill agent chooses the best possible policy and the exploration is only incidentally performed on account of the costly trial.

3.3 Algorithm for Updating the Network

Q-learning [6] is one of the most promising reinforcement learning methods. Unfortunately, Q-learning assumes that the environment is stationary and completely observable. This assumption is not realized in the RoboCup problem.

Littman [4] pointed out that Q-learning can be applied in not only MDPs but the framework of Markov games. In the framework of Markov games, the environment is supposed to include opponents. He used a simple soccer game as an example, but the state space was quite small and the environment was completely observable.

Kimura [2] proposed the stochastic gradient ascent (SGA) method in POMDPs. A large state space is permitted in the SGA method. The general form of the SGA algorithm is the following:

1. Observe X_t in the environment.
2. Execute action a_t with probability $\pi(a_t, W, X_t)$.
3. Receive the immediate reward r_t .
4. Calculate $e_i(t)$ and $D_i(t)$ as

$$e_i(t) = \frac{\partial}{\partial w_i} \ln (\pi(a_t, W, X_t)),$$

$$D_i(t) = e_i(t) + \gamma D_i(t - 1),$$

where $\gamma(0 \leq \gamma < 1)$ denotes the discount factor, and w_i does the i^{th} component of W .

5. Calculate $\Delta w_i(t)$ as

$$\Delta w_i(t) = (r_t - b) D_i(t),$$

where b denotes the reinforcement baseline.

6. Policy Improvement: update W as

$$\begin{aligned}\Delta W(t) &= (\Delta w_1(t), \Delta w_2(t), \dots, \Delta w_i(t), \dots), \\ W &\leftarrow W + \alpha(1 - \gamma)\Delta W(t),\end{aligned}$$

where α is a nonnegative learning rate factor.

7. Move to the time step $t + 1$, and go to step 1.

This research adopts the SGA method.

In this research, W is the set of weights of the neural network in Figure 5. Action probability is defined as $\pi(a_t, W, X_t) = o_t / \sum_{s \in S} o_s$, where o_t is the output value when the input values are X_t , and S is the set of squares of the grid in Figure 2.

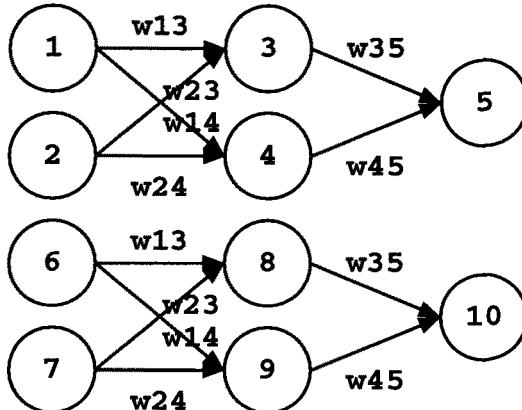


Fig. 6. The neural network is to be reused repeatedly.

The network is applied to all squares $s \in S$. In other words, the network is to be reused repeatedly while the action probability $\pi(a_t, W, X_t)$ is being calculated. Figure 6 shows a simple neural network for explanation of calculating the eligibility e_i . The networks in Figure 6 are identical except their activation values, therefore the activation values v are:

$$\begin{aligned}v_5 &= g(w_{3,5}v_3 + w_{4,5}v_4) \\ v_{10} &= g(w_{3,5}v_8 + w_{4,5}v_9)\end{aligned}$$

where g is the sigmoid function. When action a_5 is selected, eligibility e_i is calculated as follows:

$$e_{3,5} = \frac{\partial}{\partial w_{3,5}} \ln \left(\frac{v_5}{v_5 + v_{10}} \right)$$

$$\begin{aligned}
&= \frac{1}{v_5} \frac{\partial}{\partial w_{3,5}} v_5 - \frac{1}{v_5 + v_{10}} \frac{\partial}{\partial w_{3,5}} (v_5 + v_{10}) \\
&= v_3 \left(\frac{1}{v_5} - \frac{1}{v_5 + v_{10}} \right) g'(in_5) - \\
&\quad v_8 \frac{1}{v_5 + v_{10}} g'(in_{10}) \\
e_{1,3} &= \frac{\partial}{\partial w_{1,3}} \ln \left(\frac{v_5}{v_5 + v_{10}} \right) \\
&= \frac{1}{v_5} \frac{\partial}{\partial w_{1,3}} v_5 - \frac{1}{v_5 + v_{10}} \frac{\partial}{\partial w_{1,3}} (v_5 + v_{10}) \\
&= w_{3,5} v_1 g'(in_3) \left(\frac{1}{v_5} - \frac{1}{v_5 + v_{10}} \right) g'(in_5) - \\
&\quad w_{3,5} v_6 g'(in_8) \frac{1}{v_5 + v_{10}} g'(in_{10})
\end{aligned}$$

where in is the sum of weighted input values of the unit, and g' is the differential sigmoid function $g' = g(1 - g)$.

The above-mentioned update rules are too sensitive to the output values. For example, if the output value v_5 is quite small, eligibilities are very big. This easily causes oscillations. Besides, if all the output values are equal ($v_5 = v_{10}$), eligibilities are very small. This research uses modified eligibilities as follows:

$$\begin{aligned}
e_{3,5} &= v_3(0.1 - v_5)g'(in_5) + \\
&\quad v_8(0.1 - v_{10})g'(in_{10}) + \\
&\quad v_3(0.9 - v_5)g'(in_5) \times 2 \\
e_{1,3} &= w_{3,5} v_1 g'(in_3)(0.1 - v_5)g'(in_5) + \\
&\quad w_{3,5} v_6 g'(in_8)(0.1 - v_{10})g'(in_{10}) + \\
&\quad w_{3,5} v_1 g'(in_3)(0.9 - v_5)g'(in_5) \times 2
\end{aligned}$$

These equations are equivalent to the complementary back propagation's.

4 Experiments

The update module needs about 10 milliseconds computation. Eleven agents' update modules amount to about 110 milliseconds. So, the module can not be executed in every turn (100 milliseconds). The module is executed in every 1100 milliseconds so as to operate safely.

An agent often kicks the ball twice or three times in one catch. An agent receives a reward when it kicks the ball, but it should be given one reward in a catch. So, an agent is limited to receive only one reward in a second.

If agents crowd around the ball, agents in the lump often receive lots of rewards because the ball doesn't get out. An agent can receive a reward only when the distance to other agents is over 5 meters.

As described in Section 3.2, moving the position is costly action. Even if the output values of the position function indicate to go to somewhere distant, the action is not realistic. So an agent goes to a feasible place to which it can go with remaining stamina.

An agent can move the position instantly before kick-off. The best position of an agent is decided by other agents' positions. Therefore, agents move their position instantly by turns.

The learning team learns from a long game playing with a team of the fixed position strategy.

4.1 Incidental Exploration

This section describes the results of learning with the incidental exploration of Section 3.2.

The learning team converged at the two-lumps strategy in which agents tend to crowd to other team-mates and form two lumps shown in Figure 7. One lump is formed in front of the team's goal and the other is formed at the center of the field.

Figure 9 shows the transition of the behavior of an agent of the learning team. The horizontal axis means the learning time by 100 milliseconds. The vertical axis means the regularized feature value of behaviors. Each element corresponds to the input value of the position function in Section 2.5. The lower value means the closer distance. Therefore, the agent prefers the closer place from its goal, the farther place from the opponents' goal, the closer place from a team-mate, and the farther place from opponents. This figure is smoothed to be clear.

Figure 10 shows the transition of the learning team's score rate, the opponent team's score rate, and the rate of the number of rewarded kicks of an agent. These rates are counted in every 10 seconds and smoothed. All of the three rates are neither increasing nor decreasing in this figure. In other words, the learning team did not learn well after all.

There are some reasons why this learning converged at the two-lumps strategy:

- If an agent prefers the closer place to another team-mate, the agent often leads other agents to the lump strategy. The lump strategy is easily reinforced.
- Usually, some agents rush to the ball's location. So, the lump strategy is rarely reduced.
- If the learning team is not so strong, agents tend to reinforce the closer place to the team's goal.
- The center place is also apt to be reinforced because the ball frequently goes through there.

4.2 Observational Reinforcement

The incidental exploration reinforcement learning explores too rarely. In order to get out of the local optimal policy, agents must explore policies more fre-

quently. On the other hand, agents have to also exploit policies because too much exploration disturbs other agents' learning. In conventional reinforcement learning, there is not so good solution for this trade-off. Fortunately, the trade-off can be eased in the soccer-position problem because an agent can estimate the position in which it can kick the ball frequently. An agent can reinforce a hopeful position from observation without practical experience. When the ball goes to somewhere far from team-mates, agents reinforce the place. This section describes the results of this observational reinforcement learning.

From preliminary experiments, it becomes clear that absence of the goal-keeper is a fatal defect in this learning. So, this section adds special strategy for the goal-keeper. The goal-keeper has a fixed position and a fixed territory.

Figure 8 shows one situation in the middle of the learning. The formation of the learning team seems very appropriate to the opponents'. Each agent marks an opponent effectively, and the formation is extended moderately. The learning team was stronger than the fixed position's in this period of the learning.

Unexpectedly, the learning team did not converge at this strategy. Figure 11 shows the transition of the behavior of an agent. This agent learned the strategy in which it prefers the closer place from its goal and the farther place from other team-mates. The transition of the score rate is shown in Figure 12. The learning team led the fixed position team in around 40000, but after that, the learning team did not get stronger but rather get weaker. On the other hand, the kick rate was increased favorably. The kick rate is the rate of rewards. So, the fact is that the learning team learned the admirable position certainly.

5 Conclusions

This paper explained a RoboCup team, Andhill, and applied reinforcement learning to a complex and large problem, the soccer-position problem. This paper especially used observational reinforcement as a solution to the exploration problem.

Andhill participated in the RoboCup97 tournament using the fixed position strategy. Fortunately, Andhill won the second prize. A chance of winning to the champion team seemed to exist in the position fitting. The approach in this research must be promising.



Fig. 7. The two-lumps strategy in Section 4.1

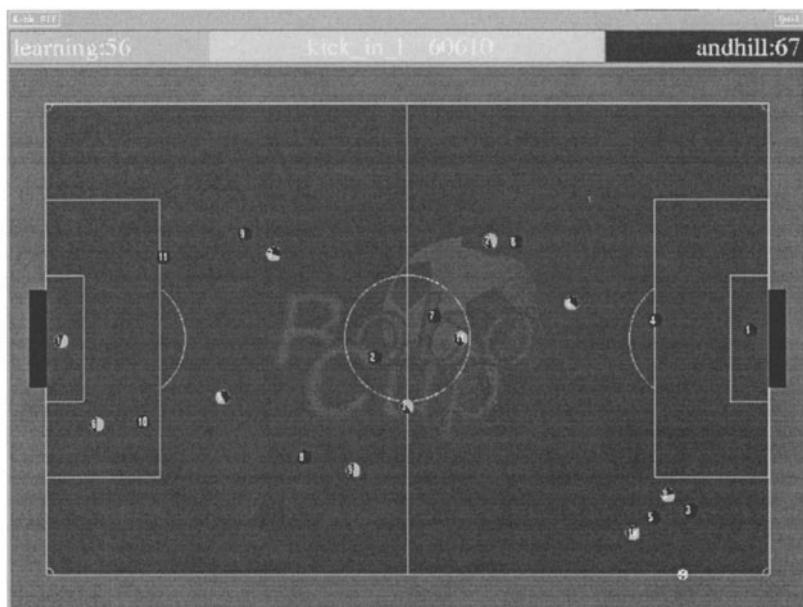


Fig. 8. One situation in the middle of the learning in Section 4.2

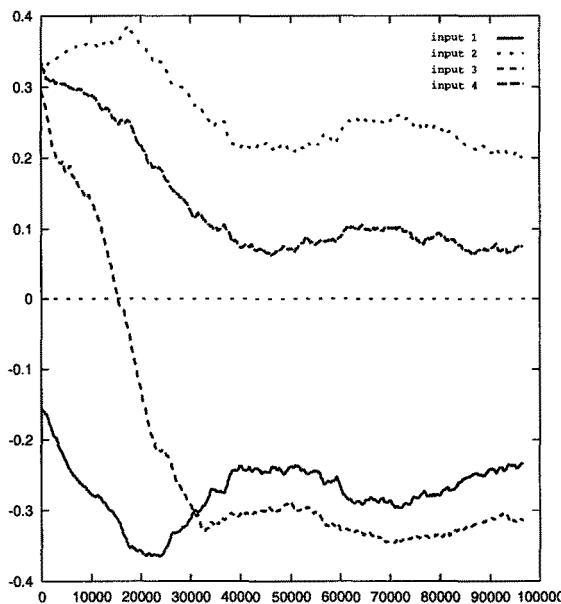


Fig. 9. The transition of the behavior of an agent in Section 4.1: This figure omits the fifth input.

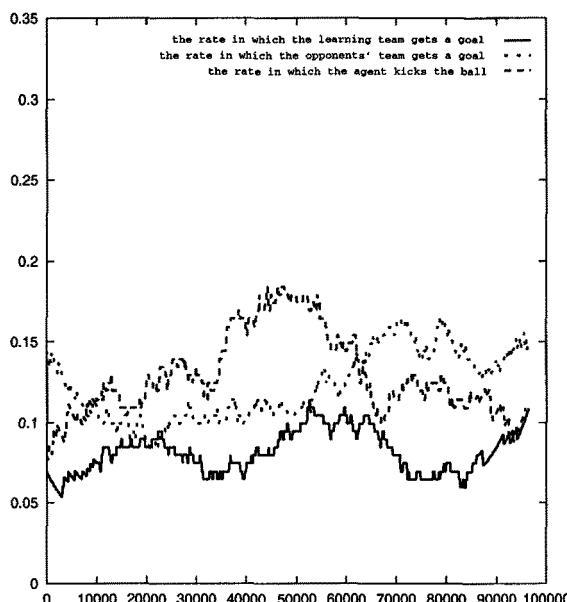


Fig. 10. The transition of the score rate in Section 4.1

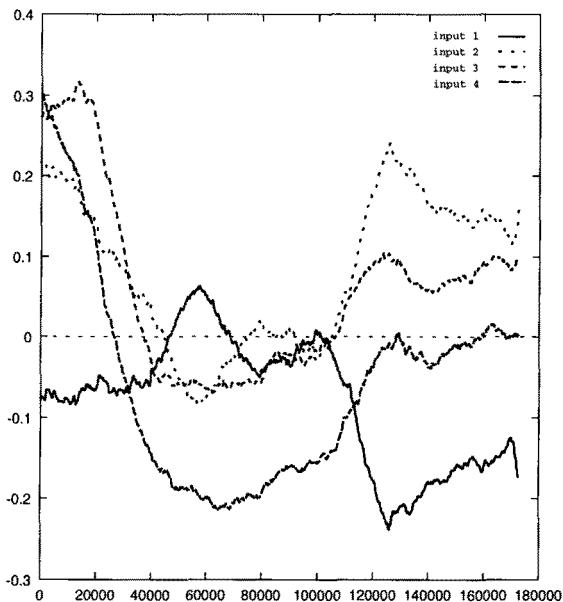


Fig. 11. The transition of the behavior of an agent in Section 4.2: This figure omits the fifth input.

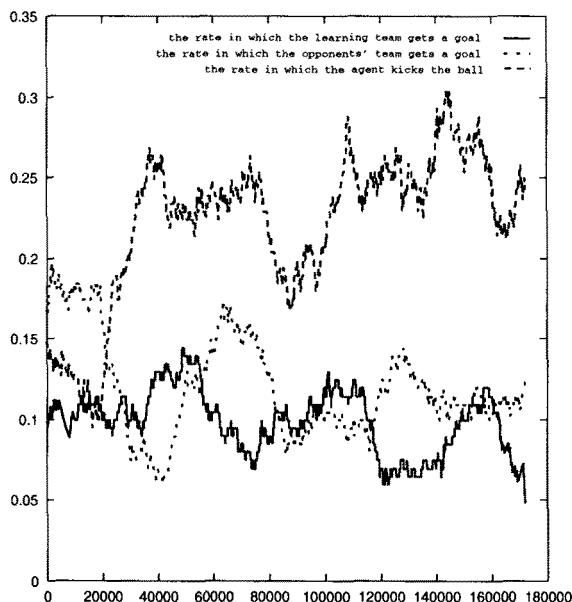


Fig. 12. The transition of the score rate in Section 4.2

References

1. Kaelbling,L.P., Littman,M.L. and Moore,A.W.: "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research* 4, pp.237 – 285 (1996).
2. Kimura,H., Yamamura,M. and Kobayashi,S.: "Reinforcement Learning in Partially Observable Markov Decision Processes", *Journal of Japanese Society for Artificial Intelligence*, Vol.11, No.5, pp.761 – 768 (1996 in Japanese).
3. Kitano,H., Asada,M., Kuniyoshi,Y., Noda,I. and Osawa,E.: "RoboCup: The Robot World Cup Initiative", *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife* (1995).
4. Littman,M.L.: "Markov games as a framework for multi-agent reinforcement learning", In *Proceedings of the Eleventh International Conference on Machine Learning*, pp.157 – 163 (1994).
5. Noda,I.: "Soccer Server : a simulator of Robo Cup", *JSAI AI-Symposium 95: Special Session on RoboCup* (1995)
6. Watkins,C.J.C.H. and Dayan,P.: "Technical Note Q-Learning", *Machine Learning* 8, pp.279 – 292 (1992).

The CMUnited-97 Simulator Team

Peter Stone and Manuela Veloso

Computer Science Department

Carnegie Mellon University

Pittsburgh, PA 15213

{pst^{one},veloso}@cs.cmu.edu

<http://www.cs.cmu.edu/{^pst^{one},^mmv}>

Abstract. The Soccer Server system provides a rich and challenging multiagent, real-time domain. Agents must accurately perceive and act despite a quickly changing, largely hidden, noisy world. They must also act at several levels, ranging from individual skills to full-team collaborative and adversarial behaviors. This article presents the CMUnited-97 approaches to the above challenges which helped the team to the semifinals of the 29-team RoboCup-97 tournament.

1 Introduction

The Soccer Server system [5] used at RoboCup-97 [2] provides a rich and challenging multiagent, real-time domain. Sensing and acting is noisy, while inter-agent communication is unreliable and low-bandwidth.

In order to be successful, each agent in a team must be able to sense and act in real time: sensations arrive at unpredictable intervals while actions are possible every 100ms. Furthermore, since the agents get local, noisy sensory information, they must have a method of converting their sensory inputs into a good world model.

Action capabilities range from low-level individual skills, such as moving to a point or kicking the ball, to high-level strategic collaborative and adversarial reasoning. Agents must be able to act autonomously, while working together with teammates towards their common overall goal. Since communication is unreliable and perception is incomplete, centralized control is impossible.

This article presents the CMUnited-97 approaches to the above challenges which helped the team to the semifinals of the 29-team RoboCup-97 simulator tournament. Section 2 introduces our overall agent architecture which allows for team coordination. Section 3 presents our agents' world model in an uncertain environment with lots of hidden state. Section 4 lays out the agents' hierarchical behavior structure that allows for machine learning at all levels of behavior from individual to collaborative to adversarial. Our team's flexible teamwork structure, which was also used by the CMUnited-97 small-size robot team [7], is presented in Section 5. Section 6 concludes.

2 Team Member Architecture

Our new teamwork structure is situated within a team member architecture suitable for domains in which individual agents can capture locker-room agreements and respond to the environment, while acting autonomously. Based on a standard agent architecture, our team member architecture allows agents to sense the environment, to reason about and select their actions, and to act in the real world. At team synchronization opportunities, the team also makes a locker-room agreement for use by all agents during periods of low communication. Figure 1 shows the functional input/output model of the architecture.

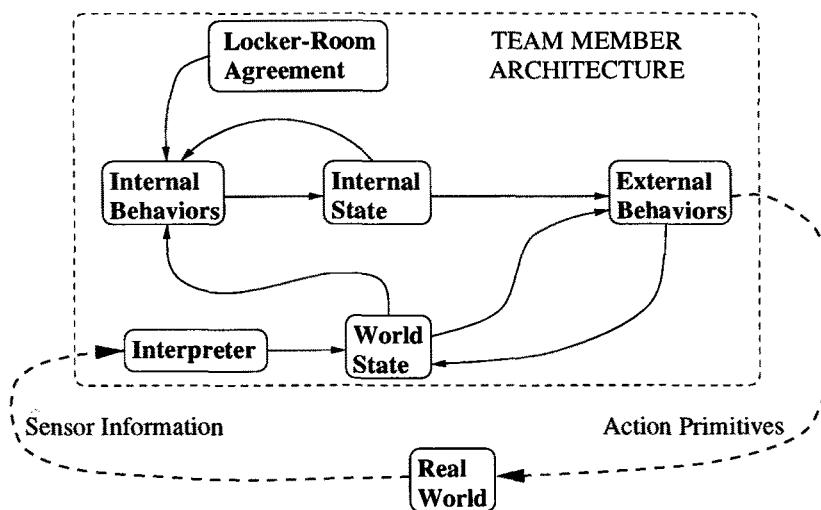


Fig. 1. The team member architecture for PTS domains.

The agent keeps track of three different types of state: the *world state*, the *locker-room agreement*, and the *internal state*. The agent also has two different types of behaviors: *internal behaviors* and *external behaviors*.

The World State reflects the agent's conception of the real world, both via its sensors and via the predicted effects of its actions. It is updated as a result of processed sensory information. It may also be updated according to the predicted effects of the external behavior module's chosen actions. The world state is directly accessible to both internal and external behaviors.

The Locker-Room Agreement is set by the team when it is able to privately synchronize. It defines the flexible teamwork structure as presented below as well as inter-agent communication protocols. The locker-room agreement may change periodically when the team is able to re-synchronize; however, it generally remains unchanged. The locker-room agreement is accessible only to internal behaviors.

The Internal State stores the agent's internal variables. It may reflect previous and current world states, possibly as specified by the locker-room agreement. For example, the agent's role within a team behavior could be stored as part of the internal state, as could a distribution of past world states. The agent updates its internal state via its internal behaviors.

The Internal Behaviors update the agent's internal state based on its current internal state, the world state, and the team's locker-room agreement.

The External Behaviors reference the world and internal states, sending commands to the actuators. The actions affect the real world, thus altering the agent's future percepts. External behaviors consider only the world and internal states, without direct access to the locker-room agreement.

Internal and external behaviors are similar in structure, as they are both sets of condition/action pairs where conditions are logical expressions over the inputs and actions are themselves behaviors as illustrated in Figure 2. In both cases, a behavior is a directed acyclic graph (DAG) of arbitrary depth. The leaves of the DAGs are the behavior types' respective outputs: internal state changes for internal behaviors and action primitives for external behaviors.

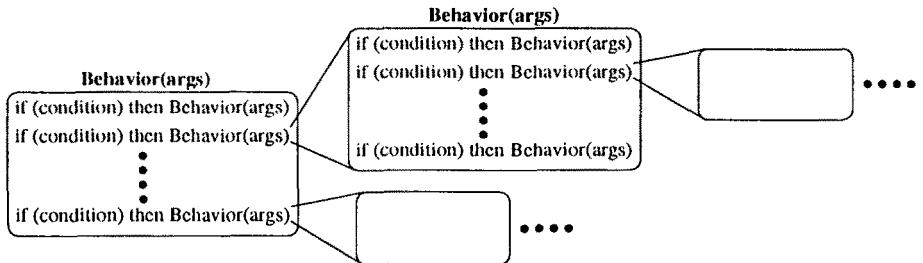


Fig. 2. Internal and external behaviors are organized in a directed acyclic graph.

Our notion of behavior is consistent with that laid out in [4]. In particular, behaviors can be nested at different levels: selection among lower-level behaviors can be considered a higher-level behavior, with the overall agent behavior considered a single “do-the-task” behavior. There is one such *top-level* internal behavior and one top-level external behavior; they are called when it is time to update the internal state or act in the world, respectively. The team structure presented in Section 5 relies and builds upon this team member architecture.

3 Predictive Memory

Based on the sensory information received from the environment, each agent can build its own world state. We developed a predictive memory model that builds a probabilistic representation of the state based on past observations. By

making the right assumptions, an effective model can be created that can store and update knowledge even when there are inaccessible parts of the environment. The agent relies on past observations to determine the positions of objects that are not currently visible. We conducted experiments to compare the effectiveness of this approach with a simpler approach, which ignored the inaccessible parts of the environment. The results obtained demonstrate that this predictive approach does generate an effective memory model, which outperforms a non-predictive model [1].

4 Layered Learning

Once the world model is successfully created, the agents must use it to respond effectively to the environment. As described in Section 2, internal behaviors update the internal state while external behaviors produce executable actuator commands. Spanning both internal and external behaviors, *layered learning* [6] is our bottom-up hierarchical approach to client behaviors that allows for machine learning at the various levels (Figure 3). The key points of the layered learning technique are as follows:

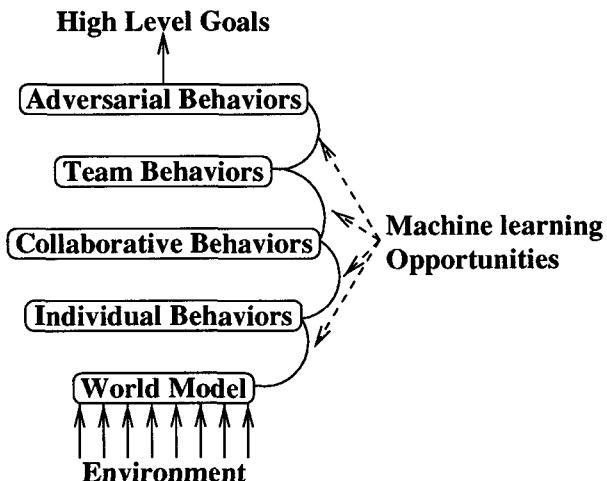


Fig. 3. An overview of the Layered Learning framework. It is designed for use in domains that are too complex to learn a mapping straight from sensors to actuators. We use a hierarchical, bottom-up approach

- The difficult aspects of the domain determine which behaviors are to be learned.
- The learned behaviors are combined in a vertical fashion, one being used as a part of the other.

Table 1 illustrates possible behavior levels within the robotic soccer domain. Because of the complexity of the domain, it is futile to try to learn intelligent behaviors straight from the primitives provided by the server. Instead, we identified useful low-level skills that must be learned *before* moving on to higher level strategies. Using our own experience and insights to help the clients learn, we acted as human coaches do when they teach young children how to play real soccer.

Layered Strategic Level	Behavior Type	Examples
Robot-ball	individual	intercept
Action selection	individual	pass or dribble
One-to-one player	collaborative	pass, aim
One-to-many player	collaborative	pass to teammate
Team formation	team	strategic positioning
Team-to-opponent	adversarial	strategic adaptation

Table 1. Examples of different behavior levels.

The low-level behaviors, such as ball interception and passing, are external behaviors involving direct action in the environment. Higher level behaviors, such as strategic positioning and adaptation, are internal behaviors involving changes to the agent's internal state. The type of learning used at each level depends upon the task characteristics. We have used neural networks and decision trees to learn ball interception and passing respectively [6]. These off-line approaches are appropriate for opponent-independent tasks that can be trained outside of game situations. We are using on-line reinforcement learning approaches for behaviors that depend on the opponents. Adversarial actions are clearly opponent-dependent. Team collaboration and action selection can also benefit from adaptation to particular opponents.

5 Flexible Team Structure

One approach to task decomposition in the Soccer Server is to assign fixed positions to agents.¹ Such an approach leads to several problems: i) short-term inflexibility in that the players cannot adapt their positions to the ball's location on the field; ii) long-term inflexibility in that the team cannot adapt to opponent strategy; and iii) local inefficiency in that players often get tired running across the field back to their positions after chasing the ball. Our introduced formations allow for flexible teamwork and combat these problems.

¹ One of the teams in Pre-RoboCup-97 (IROS'96) used and depended upon these assignments: the players would pass to the fixed positions regardless of whether there was a player there.

The definition of a position includes *home coordinates*, a *home range*, and a *maximum range*, as illustrated in Figure 4(a). The position's home coordinates are the default location to which the agent should go. However, the agent has some flexibility, being able to set its actual home position anywhere within the home range. When moving outside of the max range, the agent is no longer considered to be in the position. The home and max ranges of different positions can overlap, even if they are part of the same formations.

A formation consists of a set of positions and a set of units. The formation and each of the units can also specify inter-position behavior specifications for the member positions. Figure 4(b) illustrates the positions in one particular formation, its units, and their captains. Here, the units contain defenders, midfielders, forwards, left players, center players, and right players.

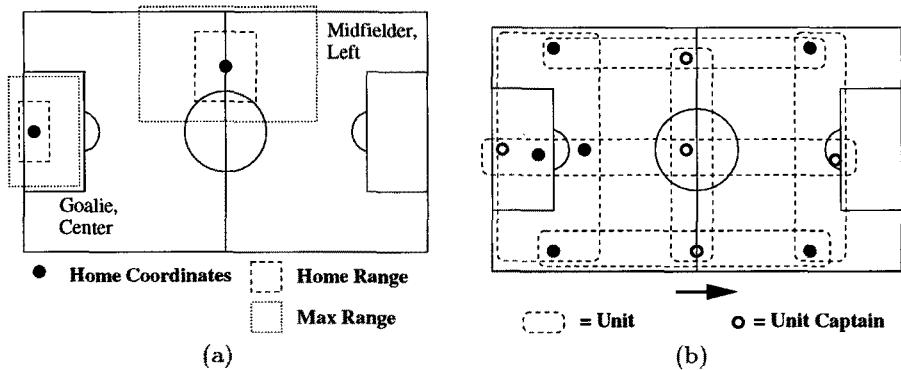


Fig. 4. (a) Different positions with home coordinates and home and max ranges. (b) Positions can belong to more than one unit.

Since the players are all autonomous, in addition to knowing its own position, each one has its own belief of the team's current formation along with the time at which that formation was adopted, and a map of teammates to positions. Ideally, the players have consistent beliefs as to the team's state, but this condition cannot be guaranteed between synchronization opportunities.

Our team structure allows for several significant features in our simulated soccer team. These features are: (i) the definition of and switching among multiple formations with units; (ii) flexible position adjustment and position switching; (iii) and pre-defined special purpose plays (set plays).

5.1 Dynamic Switching of Formations

We implemented several different formations, ranging from very defensive (8-2-0) to very offensive (2-4-4).²

The full definitions of all of the formations are a part of the locker-room agreement. Therefore, they are all known to all teammates. However during the periods of full autonomy and low communication, it is not necessarily known what formation the rest of the teammates are using. Two approaches can be taken to address this problem:

- **static formation** - the formation is set by the locker-room agreement and never changes;
- **run-time switch of formation** - during team synchronization opportunities, the team sets globally accessible run-time evaluation metrics as formation-changing indicators.

The CMUnited RoboCup-97 team switched formations based on the amount of time left relative to the difference in score: the team switched to an offensive formation if it was losing near the end of the game and a defensive formation if it was winning. Since each agent was able to independently keep track of the score and time, the agents were always able to switch formations simultaneously.

5.2 Flexible Positions

In our multiagent approach, the player positions itself flexibly such that it *anticipates* that it will be useful to the team, either offensively or defensively.

Two ways in which agents can use the position flexibility is to react to the ball's position and to mark opponents. When reacting to the ball's position, the agent moves to a location within its range that minimizes its distance to the ball. When marking opponents, agents move next to a given opponent rather than staying at the default position home. The opponent to mark can be chosen by the player (e.g., the closest opponent), or by the unit captain which can ensure that all opponents are marked, following a preset algorithm as part of the locker-room agreement.

Homogeneous agents can play different positions. But such a capability raises the challenging issue of when the players should change positions. The locker-room agreement provides procedures to the team that allow for coordinated role changing. In our case, the locker-room agreement designates an order of precedence switching among positions within each unit. By switching positions within a formation, the overall joint performance of the team is improved. Position-switching saves player energy and allows them to respond more quickly to the ball.

5.3 Pre-Planned Set Plays

The final implemented improvement facilitated by our flexible teamwork structure is the introduction of set plays, or pre-defined special purpose plays. As a part of the locker-room agreement, the team can define multi-step multiagent

² Soccer formations are typically described as goalie-defenders-midfielders--forwards [3].

plans to be executed at appropriate times. Particularly if there are certain situations that occur repeatedly, it makes sense for the team to devise plans for those situations.

In the robotic soccer domain, certain situations occur repeatedly. For example, after every goal, there is a kickoff from the center spot. When the ball goes out of bounds, there is a goal-kick, a corner-kick, or a kick-in. In each of these situations, the referee informs the team of the situations. Thus all the players know to execute the appropriate set play. Associated with each set-play-role is not only a location, but also a behavior. The player in a given role might pass to the player filling another role, shoot at the goal, or kick the ball to some other location.

We found that the set plays significantly improved our team's performance. During the RoboCup-97 competitions, several goals were scored off of set plays.

6 Conclusion

The Soccer Server system provides a wide range of AI challenges. Here we have described our approaches to problems ranging from world modelling to multiagent cooperation. Machine learning techniques are used throughout to improve performance of individual and team behaviors. Our successful implementation reached the semifinals of RoboCup-97.

Acknowledgements

This research is sponsored in part by the Defense Advanced Research Projects Agency (DARPA), and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-95-1-0018 and in part by the Department of the Navy, Office of Naval Research under contract number N00014-95-1-0591. Views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the Air Force, of the Department of the Navy, Office of Naval Research or the United States Government.

References

1. Mike Bowling, Peter Stone, and Manuela Veloso. Predictive memory for an inaccessible environment. In *Proceedings of the IROS-96 Workshop on RoboCup*, November 1996.
2. Hiroaki Kitano, Yasuo Kuniyoshi, Itsuki Noda, Minoru Asada, Hitoshi Matsubara, and Eiichi Osawa. RoboCup: A challenge problem for AI. *AI Magazine*, 18(1):73–85, Spring 1997.
3. Michael L. LaBlanc and Richard Henshaw. *The World Encyclopedia of Soccer*. Visible Ink Press, 1994.

4. Maja J. Mataric. Interaction and intelligent behavior. MIT EECS PhD Thesis AITR-1495, MIT AI Lab, August 1994.
5. Itsuki Noda and Hitoshi Matsubara. Soccer server and researches on multi-agent systems. In *Proceedings of the IROS-96 Workshop on RoboCup*, November 1996.
6. Peter Stone and Manuela Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *To appear in Applied AI Journal*, 1998.
7. Manuela Veloso, Peter Stone, Kwun Han, and Sorin Achim. CMUnited: A team of robotic soccer agents collaborating in an adversarial environment. In H. Kitano, editor, *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Springer Verlag, Berlin, 1998, in this volume.

Co-evolving Soccer Softbot Team Coordination with Genetic Programming

Sean Luke
seanl@cs.umd.edu

Charles Hohn
hohn@mailb.harris.com

Jonathan Farris
jfarris@wam.umd.edu

Gary Jackson
garyj@wam.umd.edu

James Hendler
hendler@cs.umd.edu

Department of Computer Science
University of Maryland
College Park, MD 20742 USA

Abstract. In this paper we explain how we applied genetic programming to behavior-based team coordination in the RoboCup Soccer Server domain. Genetic programming is a promising new method for automatically generating functions and algorithms through natural selection. In contrast to other learning methods, genetic programming's automatic programming makes it a natural approach for developing algorithmic robot behaviors. The RoboCup Soccer Server was a very challenging domain for genetic programming, but we were pleased with the results. At the end, genetic programming had produced teams of soccer softbots which had learned to cooperate to play a good game of simulator soccer.

1 Introduction

The RoboCup competition pits robots against each other in a simulated soccer tournament [Kitano *et al.*, 1995]. The aim of the RoboCup competition is to foster an interdisciplinary approach to robotics and agent-based Artificial Intelligence by presenting a domain that requires large-scale cooperation and coordination in a dynamic, noisy, complex environment.

The RoboCup competition has two leagues, a “real” robot league and a “virtual” simulation league. In RoboCup’s “virtual” competition, players are not robots but computer programs which manipulate virtual robots through RoboCup’s provided simulator, the RoboCup Soccer Server [Itsuki, 1995]. The Soccer Server provides a simulator environment with complex dynamics, noisy and limited sensor information, noisy control, and real-time play. To win a soccer match in the Soccer Server, players must overcome these issues and cooperate as a team in the face of limited communication ability and an incomplete world-view.

Given the RoboCup Soccer Server’s model of loosely-distributed agent coordination and its dynamic environment, a reactive behavior-based approach is an appealing way to coordinate a softbot soccer team. However, there are a wide variety of possible behaviors (even very simple ones), and the number of permutations of behavior combinations

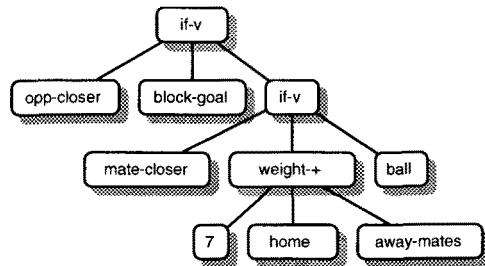


Fig. 1. A typical (small) GP algorithm tree

amongst eleven independent players can be quite high. Instead of hand-coding these behaviors for each agent, we think it is attractive to have the agents *learn* good behaviors and coordination on their own. Beyond their interesting AI and Alife aspects, when agents learn on their own, they may find interesting solutions to the problem that hand-coding might overlook. The dynamics of the RoboCup soccer simulator are complex and difficult to optimize for. With enough time, a learned strategy can evaluate a broad range of different behaviors and hone in on those most successful.

Most learning strategies (neural networks, decision trees, etc.) are designed not to develop algorithmic behaviors but to learn a nonlinear function over a discrete set of variables. These strategies are effective for learning low-level soccer-player functions such as ball interception, etc., but may not be as well suited for learning emergent, high-level player coordination. In contrast, genetic programming (GP) uses evolutionary techniques to learn symbolic functions and algorithms which operate in some domain environment [Koza 1992]. This makes GP natural for learning programmatic behaviors in a domain like the Soccer Server. This paper describes GP and how we used it to evolve coordinated team behaviors and actions for our soccer softbots in RoboCup-97.

Genetic programming has been successfully applied many times in the field of multiagent coordination. [Reynolds, 1993] used GP to evolve “boids” in his later work on flocking and herd coordination. [Raik and Durnota, 1994] used GP to evolve cooperative sporting strategies, and [Luke and Spector, 1996], [Haynes *et al.*, 1995], and [Iba, 1996] used GP to develop cooperation in predator-prey environments and other domains. Even so, evolutionary computation is rarely applied to a problem domain of this difficulty. As such, our goal was not so much to use evolutionary computation to develop finely-tuned soccer players as it was to see if evolving a fair team was even possible. As it turned out, we were pleasantly surprised with the results. Our evolved teams learned to disperse throughout the field, pass and dribble, defend the goal, and coordinate with and defer to other teammates.

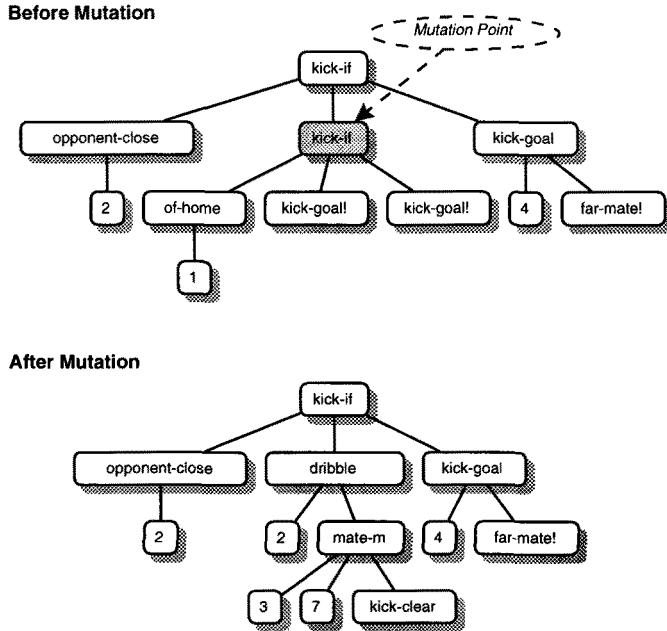


Fig. 2. The point mutation operator in action. This operator replaces one subtree in a genome with a randomly-generated subtree.

2 Genetic Programming

Genetic programming is a variant of the Genetic Algorithm [Holland, 1975] which uses evolution to optimize actual computer programs or algorithms to solve some task. In GP parlance, these algorithms are known as “individuals” or “genomes”. The most common form of genetic programming is due to John Koza [Koza, 1992]. This form optimizes one or more LISP-like “program-trees” formed from a primordial soup of atomic functions. These trees serve both as the genetic material of an individual and as the code for the resultant algorithm itself; there is no intermediate representation.

An example GP tree is shown in Figure 1. A GP genome tree can be thought of as a chunk of LISP program code: each node in the tree is a function, which takes as arguments the results of the children to the node. In this way, Figure 1 can be viewed as the LISP code

```
(if-v opp-closer block-goal
  (if-v mate-closer
    (weight 7 home away-mates) ball))
```

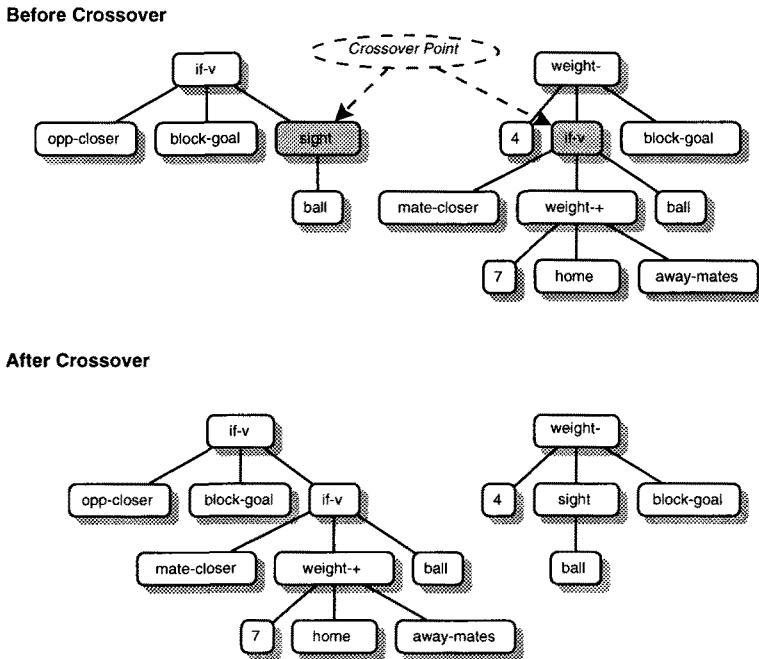


Fig. 3. The subtree crossover operator in action. This operator swaps subtrees among two genomes.

Genetic programming optimizes its genome trees with a process similar to the Genetic Algorithm. The user supplies the GP system with a set of atomic functions with which GP may build trees. Additionally, the user provides an *evaluation function*, a procedure which accepts an arbitrary GP genome tree, and returns an assessed fitness for this genome. The evaluation function assesses the fitness of a tree by executing its code in a problem domain.

The GP system begins by creating a large population of random trees for its first generation. It then uses the evaluation function to determine the fitness of the population, selects the more fit genome trees, and applies various breeding operators to them to produce a new generation of trees. It repeats this process for successive generations until either an optimally fit genome tree is found or the user stops the GP run.

GP's breeding operators are customized to deal with GP's tree-structured genomes. The three most common operators are *subtree crossover*, *point mutation*, and *reproduction*. GP's mutation operator (shown in Figure 2) takes a single fit genome, replaces an arbitrary subtree in this tree with a new, randomly-generated subtree, and adds the resultant tree to the next generation. GP's crossover operator swaps random subtrees in two fit trees to produce two new trees for the next generation, as shown in Figure 3. GP's reproduction operator simply takes a fit tree and adds it to the next generation. Our implementation used crossover and mutation, but not reproduction.

3 The Challenge of Evolutionary Computation

As if the soccer server's complex dynamics didn't make evolving a robot team hard enough, the server also adds one enormously problematic issue: time. As provided, the soccer server runs in real-time, and all twenty-two players connect to it via separate UDP sockets. The server dispenses sensor information and receives control commands every ten milliseconds; as a result, games can last from many seconds to several minutes. Game play can be sped up by hacking the server and players into a unified program (removing UDP) and eliminating the ten-millisecond delay. Unfortunately, for a variety of reasons this does not increase speed as dramatically as might be imagined, and if not carefully done, runs the risk of changing the game dynamics (and hence "changing" the game the players would optimize over).

The reason all this is such a problem is that evolutionary computation, and genetic programming in specific, typically requires a huge number of evaluations, and each new evaluation is another soccer simulator trial. In previous experiments with considerably simpler cooperation domains [Luke and Spector, 1996], we have found that genetic programming may require at least 100,000 evaluations to find a reasonable solution. We suspected the soccer domain could be much worse. Consider that just 100,000 5-minute-long evaluations in serial in the soccer server could require up to a full year of evolution time.

Our challenge was to cut this down from years to a few weeks or months, but still produce a relatively good-playing soccer team from only a few evolutionary runs. We accomplished this with a variety of techniques:

- We tried various evolutionary computation techniques (for example, co-evolution) to cut down on the number of evaluations.
- We customized GP in a variety of ways to be able to use smaller population sizes and numbers of generations.
- After initial experimentation, we designed the function set and evaluation criteria to promote better evolution in the domain.
- We performed simultaneous runs with different genome styles to promote diversity.
- We sped up play by performing up to 200 single-player evaluations and up to 32 full-team game evaluations in parallel. We also cut down game time from 10 minutes to between 20 seconds and one minute.

4 Using Genetic Programming to Evolve Soccer Behaviors

The basic function set with which our soccer softbots were built consisted of *terminal functions* of arity 0 which returned sensor information, and *nonterminal functions* which operated on this data, provided flow-control, or modified internal state variables. We used Strongly-Typed GP [Montana, 1995] to provide for a variety of different types of data (booleans, vectors, etc.) accepted and returned by GP functions. Strongly-Typed GP allows the user to assign a *type* to the arguments and the return value of each function. It then forms GP trees with the constraint that types of child and parent nodes must match appropriately. This allowed us to include a rich set of GP operators, but still constrain the possible permutations of function combinations.

Function Syntax Returns Description

(home)	v	A vector to my home.
(ball)	v	A vector to the ball.
(findball)	v	A zero-length vector to the ball.
(block-goal)	v	A vector to the closest point on the line segment between the ball and the goal I defend.
(away-mates)	v	A vector away from known teammates, computed as the inverse of $\sum_{m \in \{\text{vectorstoteammates}\}} \frac{\max - \ m\ }{\ m\ } m$
(away-opps)	v	A vector away from known opponents, computed as the inverse of $\sum_{o \in \{\text{vectorstoopponents}\}} \frac{\max - \ o\ }{\ o\ } o$
(squad1)	b	t if I am first in my squad, else nil.
(opp-closer)	b	t if an opponent is closer to the ball than I am, else nil.
(mate-closer)	b	t if a teammate is closer to the ball than I am, else nil.
(home-of i)	v	A vector to the home of teammate <i>i</i> .
(block-near-opp v)	v	A vector to the closest point on the line segment between the ball and the nearest known opponent to me. If there is no known opponent, return <i>v</i> .
(mate <i>i</i> <i>v</i>)	v	A vector to teammate <i>i</i> . If I can't see him, return <i>v</i> .
(if-v <i>b</i> <i>v1</i> <i>v2</i>)	v	If <i>b</i> is t, return <i>v1</i> , else return <i>v2</i> .
(sight <i>v</i>)	v	Rotate <i>v</i> just enough to keep the ball in sight.
(ofme <i>i</i>)	b	Return t if the ball is within $\frac{i}{\max}$ units of me, else nil.
(ofhome <i>i</i>)	b	Return t if the ball is within $\frac{i}{\max}$ units of my home, else nil.
(ofgoal <i>i</i>)	b	Return t if the ball is within $\frac{i}{\max}$ units of the goal, else nil.
(weight-+ <i>i</i> <i>v1</i> <i>v2</i>)	v	Return $\frac{v1(i) + v2(9-i)}{9}$.
(far-mate <i>i</i> <i>k</i>)	k	A vector to the most offensive-positioned teammate who can receive the ball with at least $\frac{i+1}{10}$ probability. If none, return <i>k</i> .
(mate-m <i>i1</i> <i>i2</i> <i>k</i>)	k	A vector to teammate <i>i1</i> if his position is known and he can receive the ball with at least $\frac{i2+1}{10}$ probability. If not, return <i>k</i> .
(kick-goal <i>i</i> <i>k</i>)	k	A vector to the goal if the kick will be successful with at least $\frac{i+1}{10}$ probability. If not, return <i>k</i> .
(dribble <i>i</i> <i>k</i>)	k	A "dribble" kick of size $\frac{i}{20}(\max)$ in the direction of <i>k</i> .
(kick-goal!)	k	Kick to the goal.
(far-mate!)	k	Kick to the most offensive-positioned teammate. If there is none, kick to the goal.
(kick-clear)	k	Kick out of the goal area. Namely, kick away from opponents as computed with (away-opps), but adjust the direction so that it is at least 135 degrees from the goal I defend.
(kick-if <i>b</i> <i>k1</i> <i>k2</i>)	k	If <i>b</i> is t, return <i>k1</i> , else return <i>k2</i> .
(opponent-close <i>i</i>)	b	Return t if an opponent is within $\frac{\max}{(1.5)i}$ of me.
0,1,2,3,4,5,6,7,8,9	i	Constant integer values.

Table 1. Some GP functions used in the soccer evaluation runs. Other functions included internal state, magnitude and cross-product comparison, angle rotation, boolean operators, move history, etc. *max* is the approximate maximum distance of kicking, set to 35. *k* is a kick-vector, *v* is a move-vector, *i* is an integer, and *b* is a boolean.

Before evolving a team, we had to create the set of low-level “basic” behavior functions to be used by its players. Table 1 gives a sampling of the basic functions we provided our GP system with which to build GP trees. We decided early on to enrich a basic function set of vector operators and if-then control constructs with some relatively domain-specific behavior functions. Some of these behaviors could be derived directly from the Soccer Server’s sensor information. This included vector functions like (kick-goal!), or (home). Others behaviors were important to include but were hand-coded because we found evolving them unsuccessful, at least within our limited time constraints. These included good ball interception (a surprisingly complex task), which was formed into (ball), or moving optimally to a point between two objects (forming (block-near-opp), for example).

We used genetic programming to evolve other low-level behaviors. Most notably, we used the GP technique of *symbolic regression* to evolve symbolic functions determining the probability of a successful goal-kick or pass to a teammate, given opponents in various positions [Hohn, 1997]. Symbolic regression evolves a symbolic mathematical expression which best fits a set of data points, using only basic mathematical operators such as (+ ...), (sin ...). Our symbolic regression data points were generated by playing trials in the soccer server. The evolved results formed the mechanism behind the decision-making functions (kick-goal ...), (mate-m ...), and (far-mate ...).

To meet the needs of the soccer softbot domain, we made some significant changes to the traditional GP genome. Instead of a player algorithm consisting of a single tree, our players consisted of two algorithm trees. One tree was responsible for making kicks, and when evaluated would output a vector which gave the direction and power with which to kick the ball. The other tree was responsible for moving the player, and when evaluated would output a vector which gave the direction and speed with which to turn and dash. At evaluation-time, the program executing the player’s moves would follow the instructions of one or the other tree depending on the following simplifying state-rules:

- If the player can see the ball and is close enough to kick the ball, call the kick tree. Kick the ball as told, moving the player slightly out-of-the-way if necessary. Turn in the direction the ball was kicked.
- If the player can see the ball but isn’t close enough to kick it, call the move tree. Turn and dash as told; if the player can continue to watch the ball by doing so, dash instead by moving in reverse.
- If the player cannot see the ball, turn in the direction last turned until the player can see it.

Given a basic function set, there are a variety of ways to use genetic programming to “evolve” a soccer team. An obvious approach is to form teams from populations of individual players. The difficulty with this approach is that it introduces the *credit assignment problem*: when a team wins (or loses), how should the blame or credit be spread among the various teammates? We took a different approach: the genetic programming genome is an entire team; all the players in a team stay together through evaluations, breeding, and death.

Given that the GP genome is the team itself, this raises the question of a *homogenous* or *heterogeneous* team approach. With a homogenous team approach, each soccer player

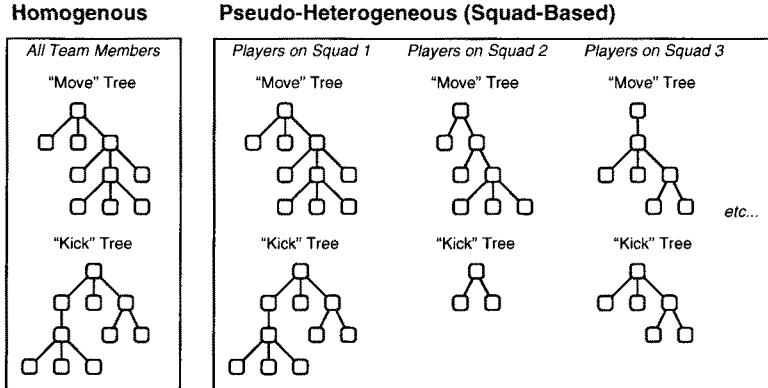


Fig. 4. Homogeneous and Pseudo-Heterogeneous (Squad-Based) genome encodings

would follow effectively the same algorithm, and so a GP genome would be one kick-move tree pair. With a heterogeneous approach, each soccer player would develop and follow its own unique algorithm, so a GP genome would be not just a kick-move tree pair, but a forest of such pairs, one pair per player. In a domain where heterogeneity is useful, the heterogeneous approach provides considerably more flexibility and the promise of specialized behaviors and coordination. However, homogenous approaches take far less time to develop, since they require evolving only a single algorithm rather than (in the case of the soccer domain) eleven separate algorithms.

To implement a fully heterogeneous approach in the soccer domain would necessitate evolving a genome consisting of twenty-two separate GP trees, far more than could reasonably evolve in the time available. Instead, we ran separate runs for homogeneous teams and for hybrid *pseudo-heterogeneous* teams (see Figure 4). The hybrid teams were divided into six squads of one or two players each; each squad evolved a separate algorithm used by all players in the squad. This way, pseudo-heterogeneous teams had genomes of twelve trees. Each player could still develop his own unique behavior, because the primordial soup included functions which let each player distinguish himself from his squadmates.

Because our genomes consisted of forests of trees, we adapted the GP crossover and mutation operators to accommodate this. In our runs, crossover and mutation would apply only to a single tree in the genome. For both homogeneous and pseudo-heterogeneous approaches, we disallowed crossover between a kick tree and a move tree. For pseudo-heterogeneous approaches, we allowed trees to cross over only if they were from the same squad: this “restricted breeding” has in previous experience proven useful in promoting specialization [Luke and Spector, 1996]. We also introduced a special crossover operator, *root crossover*, which swapped whole trees instead of subtrees. This allowed teams to effectively “trade players”, which we hoped would spread strategies through the population more rapidly.

Another issue was the evaluation function needed to assess a genome's fitness. One way to assess a team would be to play the team against one or more hand-created opponent teams of known difficulty. There are two problems with this approach. First, evolutionary computation strategies typically work more efficiently when the difficulty of the problem ramps up as evolution progresses, that is, as the population gets better, the problem gets harder. A good ramping with a suite of pre-created opponents is very difficult to gauge. Second, unless there are several opponents at any particular difficulty level, one runs the very common risk of evolving a team optimized to beat that *particular set* of hand-made opponents, instead of generalizing to playing "good" soccer.

We opted instead for evolving our teams with *co-evolution*: teams' fitnesses were assessed based on competition with peers. In our implementation, to evaluate the fitness of all the teams in the population, the GP system first paired off teams in the population, then played matches with each pair. Team fitness was based on the game score and other game criteria.

Co-evolution is chaotic and so can occasionally have undesirable effects on the population, but more often than not we have found it natural for many "competitive" domains (such as robot soccer). Co-evolution naturally ramps problem difficulty because teams in the population play against peers of approximate ability. Co-evolution can also promote generalization, because the number of "opponents" a population faces is the size of the population itself.

We initially based fitness on a variety of game factors including the number of goals, time in possession of the ball, average position of the ball, number of successful passes, etc. However, we found that in our early runs, the entire population would converge to very poor solutions. Ultimately, we found that by simply basing fitness on the number of goals alone, the population avoided such convergence. At first glance, such a simplistic fitness assessment would seem an overly crude a measure, as many early games might end with 0–0 scores. Luckily, this turned out to be untrue. We discovered that initial game scores were in fact very high: vectors to the ball and to the goal were fundamental parts of the function set, so teams did simple offense well, but defense poorly. Only later, as teams evolved better defensive strategies, would scores come down to more reasonable levels.

We performed our GP runs in parallel using a custom strongly-typed, multithreaded version of *lil-gp 1.1* [Zongker and Punch, 1995], running on a 40-node DEC Alpha cluster. To speed up evolution run time, we used population sizes between 200 and 400 (small for GP), which required a large amount of mutation (up to 70%) to prevent premature convergence.

We ran the final runs for forty generations, at which time we re-introduced into the population high-fitness individuals from past generations. We then continued runs up to the time of the RoboCup-97 competition (for twelve generations). Just prior to the competition, we held a "tournament of champions" among the twenty highest-performing teams at that point, and submitted the winner. While we feel that, given enough evolution time, the learned strategies of the pseudo-heterogeneous teams might ultimately outperform the homogeneous teams, the best teams at competition time (including the one we submitted) were homogeneous.

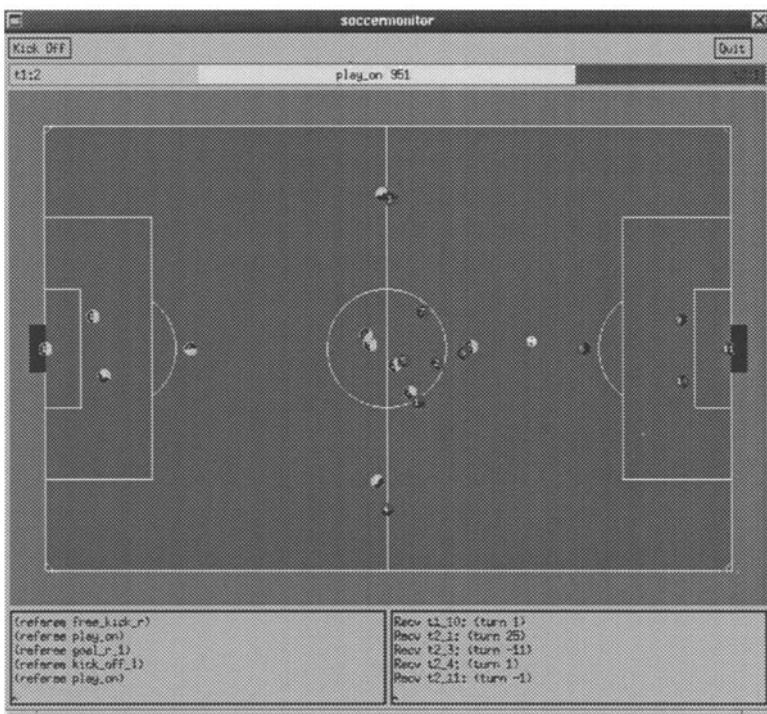


Fig. 5. A competition between two initial random teams.

5 A History of Evolution

One of the benefits of working with evolutionary computation is being able to watch the population learn. In a typical evolutionary computation experiment one would conduct a large number of runs, which provides a statistically meaningful analysis of population growth and change. For obvious reasons, this was not possible for us to do. Given the one-shot nature of our RoboCup runs (the final run took several months' time), our observations of population development are therefore admittedly anecdotal. Still, we observed some very interesting trends.

Our initial random teams consisted primarily of players which wandered aimlessly, spun in place, stared at the ball, or chased after teammates. Because (ball) and (kick-goal!) were basic functions, there were occasional players which would go to the ball and kick it to the goal. These players helped their teams rack up stratospheric scores against completely helpless opponents. Figure 5 shows two random teams in competition.

Early populations produced all sorts of bizarre strategies. One particular favorite was a (homogeneous) competition of one team programmed to move away from the ball, against another team programmed to move away from the first team. Thankfully, such strategies didn't last for many generations.

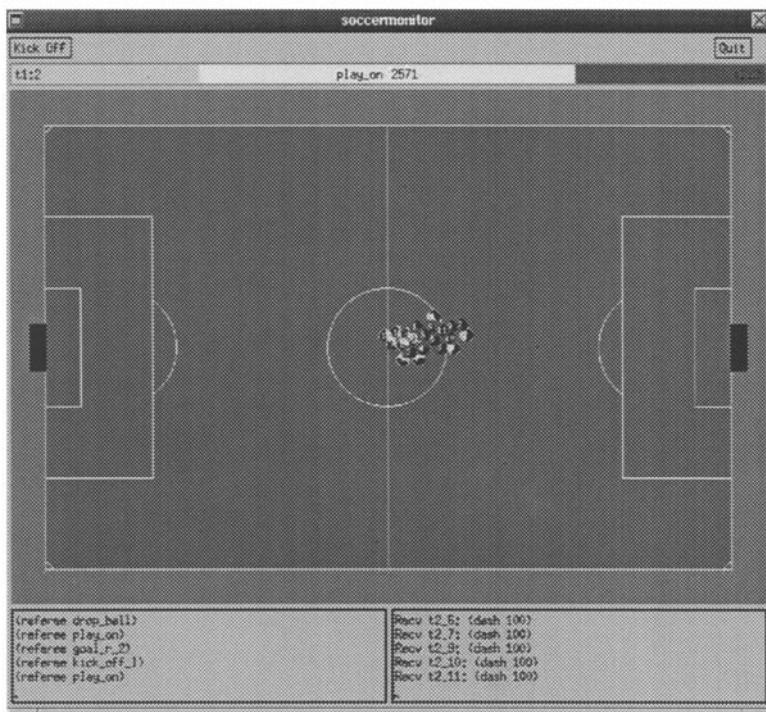


Fig. 6. “Kiddie-Soccer”, a problematic early suboptimal strategy, where everyone on the team would go after the ball and try to kick it into the goal. Without careful tuning, many populations would not escape this suboptima.

One suboptimal strategy, however, was particularly troublesome: “everyone chase after the ball and kick it into the goal”, otherwise known as “kiddie-soccer”, shown in Figure 6. This strategy gained dominance because early teams had effectively no defensive ability. Kiddie-soccer proved to be a major obstacle to evolving better strategies. The overwhelming tendency to converge to kiddie-soccer and similar strategies was the chief reason behind our simplification of the evaluation function (to be based only on goals scored). After we simplified the evaluation function, the population eventually found its way out of the kiddie-soccer suboptima and on to better strategies.

After a number of generations, the population as a whole began to develop rudimentary defensive ability. One approach we noted was to have a few players hang back near the goal when not close to the ball (Figure 7). Most teams still had many players which clumped around the ball, kiddie-soccer-style, but such simple defensive moves effectively eliminated the long-distance goal shots which had created such high scores in the past.

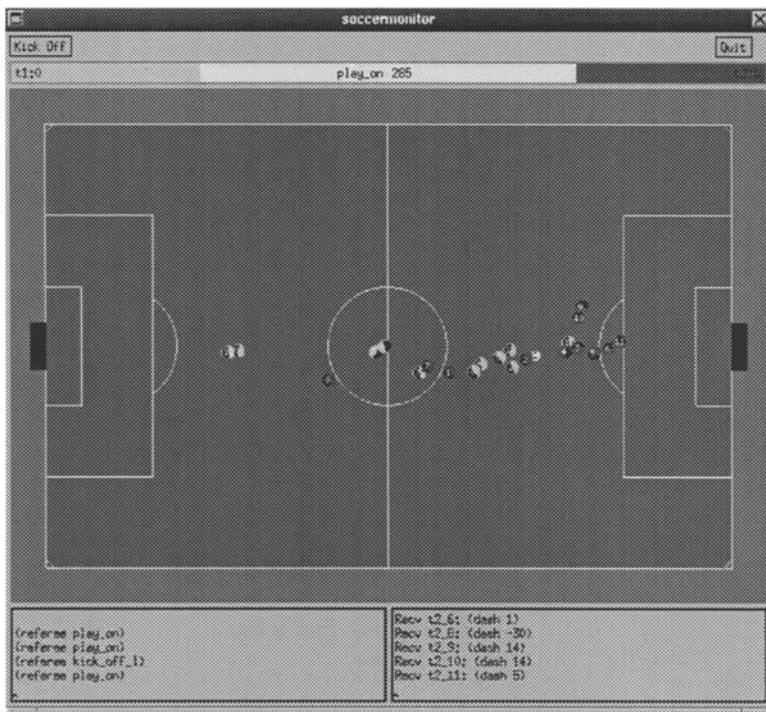


Fig. 7. Some players begin to hang back and protect the goal, while others chase after the ball.

Eventually teams began to disperse players throughout the field and to pass to teammates when appropriate instead of kicking straight to the goal, as shown in Figure 8. Homogeneous teams did this usually by using players' home positions and information about nearby teammates and ball position. But some pseudo-heterogeneous teams appeared to be forming separate offensive and defensive squad algorithms. It was unfortunate that the pseudo-heterogeneous teams were not sufficiently fit by the time RoboCup arrived; we suspect that given more time, this approach could have ultimately yielded some very good strategies.

6 Conclusions and Future Work

This project was begun to see if it was even possible to successfully evolve a team for such a challenging domain as the RoboCup Soccer Server. Given the richness of the Soccer Server environment and the very large amount of evolution time required to get reasonable results, we are very pleased with the outcome. Our evolved softbots learned to play rather well given the constraints we had to place on their evolution. As such, we think the experiment was a success.

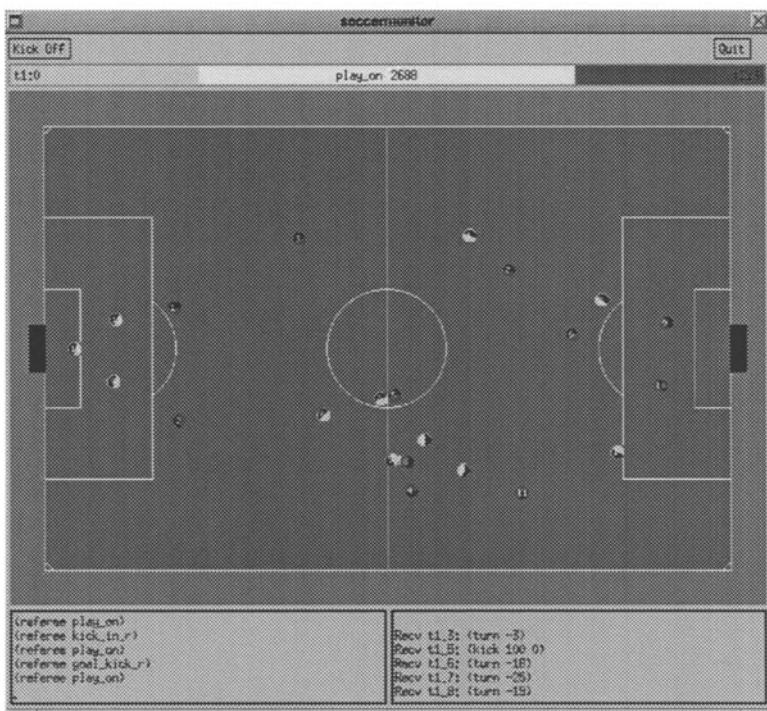


Fig. 8. Teams eventually learn to disperse themselves throughout the field.

Still, we had to compromise in order to make the project a reality. The function set we provided was heavy on flow-control and functional operation, with little internal state. In the future we hope to try more computationally sophisticated algorithms. We also hope to perform longer runs with significantly larger population sizes. Finally, we were disappointed that the pseudo-heterogeneous teams could not outperform our homogeneous teams by RoboCup competition-time. In the future we hope to try again with heterogeneous teams, which we suspect might yield better results.

While genetic programming has been very successful in some domains, it is surprisingly difficult to adapt it to others, especially domains where evaluations are expensive. Still, we think our work shows that considerably more can be done with this technique than is usually thought. As the price of computer brawn continues to drop, we feel evolutionary computation may become attractive in a variety of areas which are currently the purview of human programmers only.

7 Acknowledgements

This research is supported in part by grants to Dr. James Hendler from ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), ARL (DAAH049610297), and ARPA contract DAST-95-C0037.

Our thanks to Lee Spector, Kilian Stoffel, Bob Kohout, Daniel Wigglesworth, John Peterson, Shaun Gittens, Shu Chiun Cheah, and Tanveer Choudhury for their help in the development of this project, and to the UMIACS system staff for turning their heads while we played soccer games on their supercomputers.

References

- [Haynes *et al*, 1995]. Haynes, T., S. Sen, D. Schoenefeld and R. Wainwright. 1995. Evolving a Team. In *Working Notes of the AAAI-95 Fall Symposium on Genetic Programming*. E. V. Siegel and J. R. Koza, editors. 23–30. AAAI Press.
- [Hohn, 1997] C. Hohn. Evolving Predictive Functions from Observed Data for Simulated Robots. Senior Honor's Thesis. Department of Computer Science, University of Maryland at College Park. 1997.
- [Holland, 1975] J. H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1996.
- [Iba, 1996] H. Iba. Emergent Cooperation for Multiple Agents using Genetic Programming. In J. R. Koza, editor, *Late Breaking Papers of the Genetic Programming 1996 Conference*. Stanford University Bookstore, Stanford CA, pages 66–74, 1996.
- [Itsuki, 1995] N. Itsuki. Soccer Server: a simulator for RoboCup. In *JSAI AI-Symposium 95: Special Session on RoboCup*. December, 1995.
- [Kitano *et al*, 1995] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The Robot World Cup Initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/ALife*, 1995.
- [Koza, 1992] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge MA, 1992.
- [Luke and Spector, 1996] S. Luke and L. Spector. Evolving Teamwork and Coordination with Genetic Programming. In J. R. Koza et al., editors, *Proceedings of the First Annual Conference on Genetic Programming (GP-96)*. The MIT Press, Cambridge MA, pages 150–156, 1996.
- [Montana, 1995] D. J. Montana. Strongly Typed Genetic Programming. In *Evolutionary Computation*. The MIT Press, Cambridge MA, 3(2):199–230, 1995.
- [Raik and Durnota, 1994] S. Raik and B. Durnota. The Evolution of Sporting Strategies. In R. J. Stonier and X. H. Yu, editors, *Complex Systems: Mechanisms of Adaption*. IOS Press, Amsterdam, pages 85–92, 1994.
- [Reynolds, 1993] C. W. Reynolds. An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion. In J.-A. Meyer *et al.*, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. The MIT Press, Cambridge MA, 384–392, 1993.
- [Zongker and Punch, 1995] D. Zongker and B. Punch. *lil-gp 1.0 User's Manual*. Available at <http://isl.cps.msu.edu/GA/software/lil-gp> 1995.

Learning Cooperative Behaviors in RoboCup Agents

Masayuki Ohta

Dept. of Mathematical and Computing Sciences,
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
m-oota@is.titech.ac.jp

Abstract. In the RoboCup environment, it is difficult to learn cooperative behaviors, because it includes both real-world problems and multi-agent problems. In this paper, we describe the concept and the architecture of our team at the RoboCup'97, and discuss how to make this agent learn cooperative behaviors in the RoboCup environment. We test the effectiveness using a case study of learning pass play in soccer.

1 Introduction

Recently, multi-agent systems have become a large field of artificial intelligence.[3] Because of the complexity of multi-agent systems, machine learning techniques are indispensable.

Simulation of soccer game has become one of a standard problem of multi-agent systems, and the first RoboCup[1] was held. The official simulator for the RoboCup, Soccer Server[2], presents the following problems.

– Real-world problems

- **Noisy environment**

Perceptional information includes some errors. And action of agent is unreliable.

- **Limited field of vision**

Each agent can see only 90 degrees angle of forward. Then it have to estimate the backward situation

- **Real-time processing**

Because the ball will move, agents can not capture the ball at the current ball position. And, opponents obstruct our plan, one after another.

– Multi-agent problems

- **Cooperative behaviors are advantageous**

Some cooperative behaviors are very profitable[4]. The most obvious example here is pass play.

- **No shared memory**

When the agents cooperate, they have to make consensus of the way of cooperation. But in this case an agent have to guess another agent's action to cooperate.

- **No global view exist**

Each agent have to decide the next action only from the perceptual information from their own standpoint. This make it difficult to behave cooperatively.

We aim at acquiring effective cooperative behaviors with machine learning in such environment. In such multi-agent environment, forming agreement is necessary. Regarding the real-time processing as important in this case, we inquire into the model that make agreement without direct negotiation.

2 Team Description

In this section, we describe our team which entered the RoboCup'97. We show noteworthy points to create soccer agents in the RoboCup environment, explain the relation to our agents' architecture, and discuss how can we improve our team based on the result of the RoboCup'97.

2.1 Design Issue

As mentioned above, the RoboCup environment includes a lot of features, and all of them are obstacle when we create soccer agent. To build strong soccer team, we programmed our agents paying attention to the following two points.

Cooperation using perceptual data from different stand point

There are a lot of approaches to make agreement with negotiation among the agents. But in this case, as we mentioned before, real-time processing is important. (Otherwise, opponents will steal the ball while our agents negotiate about next strategy.) Therefore our agents do not negotiate each other, and each agent decide next action only with information from their standpoint to succeed in the cooperative work.

Main problem here is the following two.

- An agent occasionally differ with others in assessment of the situation, because of observing from different standpoint,
- Agent's field of vision is restricted, and each agent gets limited visual information.

Learning cooperative behaviors

It is very difficult to describe agents' cooperative behavior accurately. Especially, in the RoboCup environment, there are no global view and forming agreement is going far difficult. So, it is necessary to get the way to behave cooperatively with machine learning technique. But in multi-agent environment, following new problems will occur, which do not take place in single-agent environment.

- All cooperating agents have to choose a correct strategy. If one of these agents choose incorrect strategy, the collaboration will fail. Then, some agents, which choose correct strategy, will learn that it is bad selection in this case, even if it is good.
- All cooperating agents have to choose the same strategy. Even if each cooperating agent chose a correct strategy, the collaboration will fail again, unless all agents have chose the same strategy. This problem could occur when there are more than two correct strategies.
- Each agent should learn only one strategy for a situation. If some good strategies have almost the same effectiveness, an agent can not fix the strategy for the situation. Then it is hard for cooperating agents to choose the same strategy, and causes reduction of efficiency.

2.2 Action Flow

Here, we show the details of our agents' action, and mention the relation to the issues previously stated.

Our agents' action is based on the rules of perception-action pair. When an agent receives visual information, it acts as the following flow.

- **Convert visual information into internal data structure**
- **If the ball is in the kickable range, kick it to the best way**
 - If it is near the opponent's goal, shoot the ball to the best way between the left goal post and the right goal post.
 - Else if there are some teammate whom the agent can pass the ball in safety, pass the ball to the farthest teammate.
 - Else kick the ball to the best way between the opponent's left corner and the right corner.

The function to calculate the best way have three argument, left, right and distance. This function outputs the most open direction between left side and right side in the distance (Figure 1).

Considering the real-time processing, our agents decide the best way only with the information already have. Then it occasionally pass the ball to opponents, because of lacking blind spot data. But sometimes succeed in a quick pass and shoot.

We are planning to implement this module with machine learning.

- **Else if the agent have some special strategy, do the strategy**
Special strategy is action, such as run to the open space in order to receive the pass. This module, also should be implemented by machine learning, but this strategy can succeed only when the cooperation with another agent which will pass the ball (using machine learning as previously stated) succeed. Then, the problems of learning cooperative behaviors will occur.

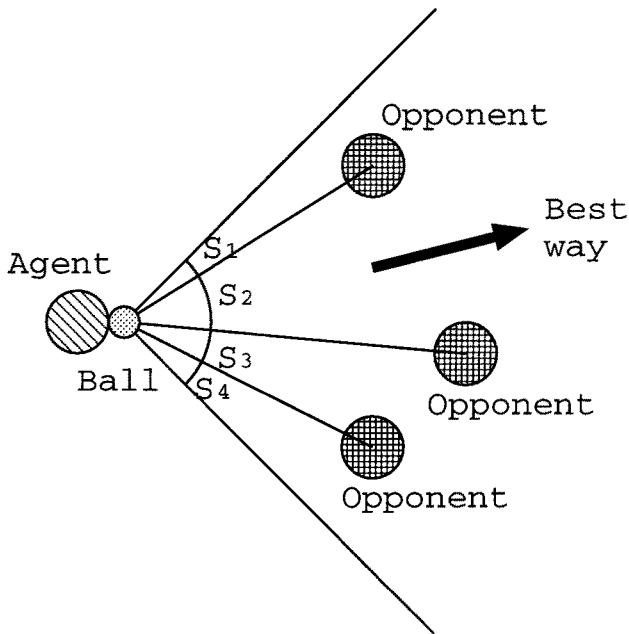


Fig. 1. Best way

- **Else if the agent is closest or second, chase the ball**

Two agents chase the ball, in case one miss the ball. Our agents estimate future location of the ball considering current location and speed of it. Then it goes to the estimate position and capture the ball.

If it does not use the estimate position, it lose the ball so often.

- **Else, trace the ball a while, then go back to their home position**

In the RoboCup environment, agents sometimes lose the ball, because perceptual information is not reliable. Not to go back their home position immediately in case an agent lose track of the ball, it trace the ball a while.

2.3 Looking back over the RoboCup'97

Simulator league of the RoboCup'97 had 29 various types of teams, but all stronger teams had following features.

- They are very good at capturing the ball
- They move very fast

Our team was defeated by AT-Humboldt-97(World Champion team) in quarter final by a score of 14 to 7. In this match, we were beaten in the following pattern, though our agents also move quickly.

1. Can not find a teammate whom the agent can pass the ball safety.
2. Kick the ball to the most safety direction and far from our goal.
3. The ball is intercepted, when an opponent reached there earlier.

In this situation, if an agent can expect the position to where the teammate will kick the ball, it can go there earlier and could get the ball. Therefore, to make strong a team, besides the agent which have the ball learn where to kick it, other agents should learn the place where the ball will be kicked to.

3 Experiments

Learning cooperative behaviors in the RoboCup environment have many problems as we mentioned above. In this section, we show our approaches to the problems using a case study of making consensus about pass courses.

3.1 Learning of the pass play

We carried out the following situation.

- Two agents (A_1, A_2) are learning combination play against two opponents (O_1, O_2): A_1 pass the ball to A_2 through O_1 and O_2 , and then A_2 shoot the ball (shown in Figure 2).
- There are two strategies (S_1, S_2).
 - S_1 : while A_1 pass through between O_1 and O_2 A_2 run forward to the point P ,
 - S_2 : while A_1 pass beside O_2 A_2 stop waiting the ball.

In this situation, we make A_1 and A_2 learn the cooperative strategy for the opponents' various position pair. The kick power of A_1 and the dash power of A_2 are constant. Then, A_1 have to learn only the direction to kick, and A_2 have to learn only the number of times to dash.

3.2 Our Approach

This learning includes many issues that we mentioned in section 2.1. We approached for each problems as following.

- **Noisy environment**
We use the neural network for the learning method, because it comparatively works well in a noisy environment.
- **Cooperation using the information from different standpoint**
Neural network can cope with this problem, too. In this experiment, we tested whether it is possible or not.

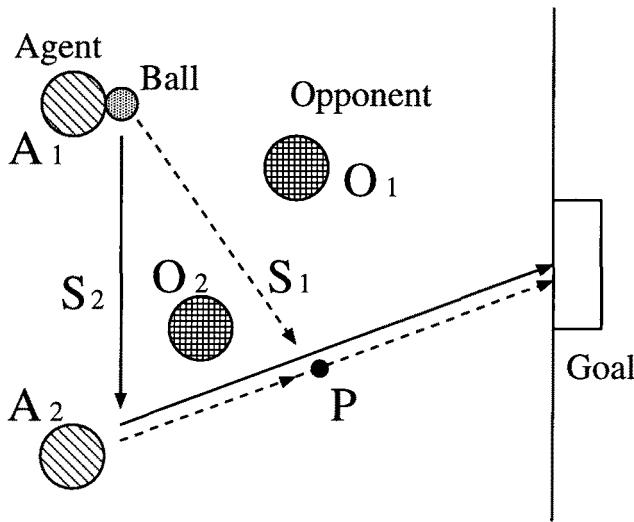


Fig. 2. situation used in this experiment

– **Limited field of vision**

Now my agents use the last data for the lacking data. Machine Learning in such environment is one of the future works.

– **All agents have to select a same and correct strategy**

In this experiment, agent try random action repeatedly. When the trial was successful, learn the pair of action and perceptual data using the back propagation.

The neural network for A_1 consists of 4 input-units, 50 first-hidden-units, 50 second-hidden-units and 20 output-units. And the neural network for A_2 consists of 4 input-units, 50 first-hidden-units, 50 second-hidden-units and 8 output-units. The inputs of these networks are relative direction and distance of O_1 and O_2 . The number of output units means that, we separate the direction into 20 ways, and restrict the number of dash from 0 to 7.

One simulation is limited to 10 seconds. Then if the agents select different strategy, such as A_1 choose S_1 and A_2 choose S_2 , they cannot shoot in time.

3.3 Results

In this experiment, we chose 5 square lattice points for possible position of each opponent. In this 625 situations, the agents learn about 100 random but success-

ful examples. This 100 examples include some examples in the same situations but ideal actions are different each other.

To learn these 100 examples completely, the neural network needed 10000 times of back propagation, for each of these 100 examples.

Table 1 shows the result of this experiment. Each column means the following.

- “goal”: shoot action was success
- “near miss”: pass seems succeeded but fail in shoot
- intercepted: intercepted by opponent
- failure: pass play failed at all

So, the sum of “goal” and “near miss” can be treated as the number of successful examples.

This result shows that the agents improved their success rate of pass play from 32% to 55%, with learning only 16% of noisy examples. Furthermore, A_1 learned that where is the best direction to kick the ball. Then A_2 could capture the ball at good place, then the number of the “goal” remarkably raised.

Therefore we can see that the agents could learn cooperative behaviors only with the perceptual information from their own stand point.

Table 1. Result of the experiment

	goal	near miss	intercepted	failure
before	34	165	98	328
	199 (32%)		426 (68%)	
after	126	215	88	196
	341 (55%)		284 (45%)	

4 Conclusion

The experiment shows the following things.

- Each agent can learn cooperative behaviors with the information from different stand points.
- After learning, agents can guess suitable answer for some situation which have not learned.
- Neural network is useful for the learning in such noisy environment.

Then, my agent can learn cooperative behaviors, if the function of the best direction and the special strategy are replaced with the neural network.

But many problems as followings are still remaining.

- In this experiment, the teammate does not move. If the agent learns for about a lot of situations of teammates, bigger neural network will be needed.

- The agent can not compensate for the lacking input. In this environment, the agent's field of vision is limited. So the lacking input will sometimes occur.
- This method takes a lot of time. Now it is far from adapting to cope with opponents in the game.

Therefore, main future works should be these three in this order.

- Experiment with changing the position of offensive agents. Then the agent can be used for soccer match.
- Deal with not only two against two but many kind of situation.
- Treat the situation, where sometimes lack the input data of neural network.

References

1. H.Kitano, M.Asada, Y.Kuniyoshi, I.Noda and E.Osawa "Robocup: The robot world cup initiative" IJCAI-95 Workshop on Entertainment and AI/Alife pages 19-24 August 1995.
2. Itsuki Noda. "Soccer server: a simulator of robocup" In Proceedings of AI symposium '95, pages 29-34 Japanese Society for Artificial Intelligence December 1995.
3. Peter Stone and Manuela Veloso. "Multiagent systems: A survey from a machine learning perspective" IEEE Transactions on Knowledge and Data Engineering, June 1996.
4. Ming Tan. "Multi-Agent Reinforcement Learning: Independent Vs. Cooperative Agents" Proceedings of the Tenth International Conference on Machine Learning pages 330-337 June 1993.

Individual Tactical Play and Action Decision Based on a Short-Term Goal

~ Team Descriptions of Team Miya and Team Niken ~

Harukazu Igarashi, Shougo Kosue, Masatoshi Miyahara, Toshiro Umaba
Kinki University, Higashi-Hiroshima City, 739-21, Japan

Abstract. In this paper we present descriptions of our two teams that participated in the simulator league of RoboCup 97. One of the teams is characterized by soccer agents that make individual tactical plays without communicating with each other. The other team is characterized by the use of an action-decision algorithm based on a short-term goal and current information. The two teams were among the best of 8 and 16 teams at the competition.

1 Introduction

A chess champion was recently defeated by a computer. What is the next challenging problem to be solved by a computer? Robot soccer is one of the relevant candidates for the standard challenging problems in Artificial Intelligence.

At the Faculty of Engineering at Kinki University, we constructed two teams of synthetic soccer agents and then participated in the simulator league of RoboCup 97(The World Cup Robot Soccer 97). In this short paper, we present technical descriptions of our two teams, Team *Miya* and Team *Niken*. Team *Miya* and Team *Niken* are characterized by *individual tactical play* and action decision based on short-term goals and current information. Many researchers often emphasize the communication between multiple agents. In a dynamically changing environment like a soccer game, however, there is not enough time for agents to communicate with each other and confirm their teammate's explicit intention. Furthermore, hostile agents interfere with the communication by "jamming" the other team's agents. Thus, we tried to invent agent-control algorithms which do not require the communication or any exhausting calculations.

2 Team Miya

2.1 Team Miya Objective

The main objective of our research with Team *Miya* is developing an agent model that satisfies the following two requirements. The first requirement is that the cooperative actions of the agents should be expressed in a simple form that can be modified easily. The second requirement is that the actions of the agents are quick and smooth under a real-time environment.

2.2 Architecture

In Fig. 1, we show the architecture of a soccer client program controlling an agent on Team Miya. A client program receives visual and auditory information, decides on an action, and *compiles* the action into a series of basic commands prescribed by RoboCup 97 regulations.

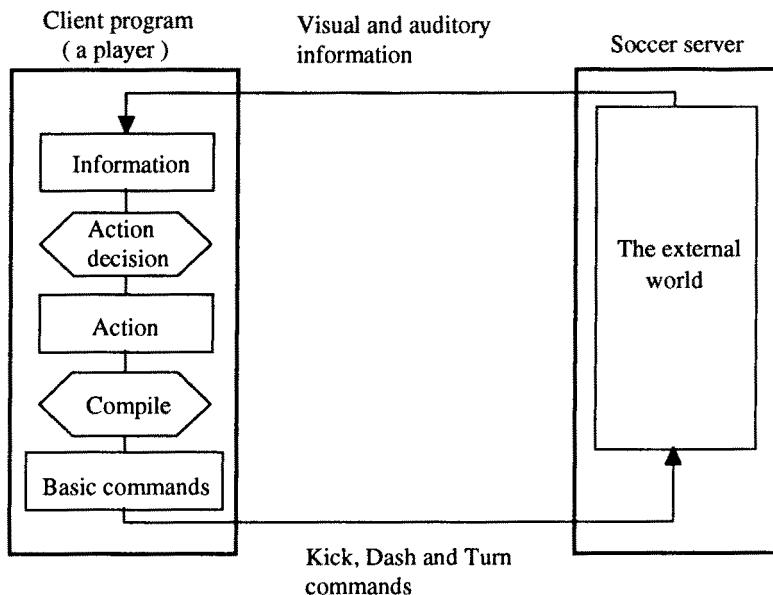


Fig. 1 The architecture of a client program controlling a soccer agent on Team Miya.

2.3 Hierarchy of Actions

The second feature of Team Miya is a hierarchy of actions. Actions are generally classified into four levels: strategy, tactics, individual play and basic commands (as shown in Table 1). A higher-level action includes more players and requires information in a wider range of time and space than a lower-level action. Coradeschi et al.[CK1] and Tambe[T1] expressed the relationship between actions as a decision tree. We call such a decision tree an *action tree*. A soccer agent selects an action from the action tree at each action cycle by analyzing visual and auditory information and by considering the agent's current state. The action is *compiled* into a series of basic commands: kick, turn and dash.

Table 1. A hierarchy of actions.

	Action	Definition	Examples
Level 4	Strategy	Cooperative team action.	Rapid attack, Zone defense
Level 3	Tactics	Cooperative action by a few players for a specific local situation.	Centering pass, Post play, Triangle pass
Level 2	Individual play	Individual player skill.	Pass, Shoot, Dribble, Clear
Level 1	Basic command	Basic commands directly controlling soccer agents.	Kick, Turn, Dash

2.4 Individual Tactical Play

The action tree contains information on compiling. Tactic actions, however, require communication between players and often slow the reactions of players in a real-time environment like a soccer game. Therefore we decided to remove tactics and introduce *individual tactical plays* into the action tree. The hierarchy of the modified action tree is shown in Table 2.

Table 2. The modified hierarchy of actions.

	Action	Definition	Examples
Level 4	Strategy	Cooperative team action.	Rapid attack, Zone defense
Level 2	Individual tactical play	Action of an individual player for a specific local situation without communication, but expecting cooperation from a teammate.	Safety pass, Post play, Centering pass
	Individual play	Individual player skill.	Pass, Shoot, Dribble, Clear
Level 1	Basic command	Basic commands directly controlling soccer agents.	Kick, Turn, Dash

Individual tactical play is introduced to reduce the delay time between decisions and actions. The individual tactical play is defined as an action that an individual plays in a specific local situation without communication from a teammate. But an agent expects some cooperation from a teammate in an individual tactical play. For Team Miya, we implemented three actions as individual tactical plays: the safety pass, the centering pass and the post play. The three plays speed up the tactical actions of the safety pass between two players, the centering pass from a wing player, and the post play of a forward player.

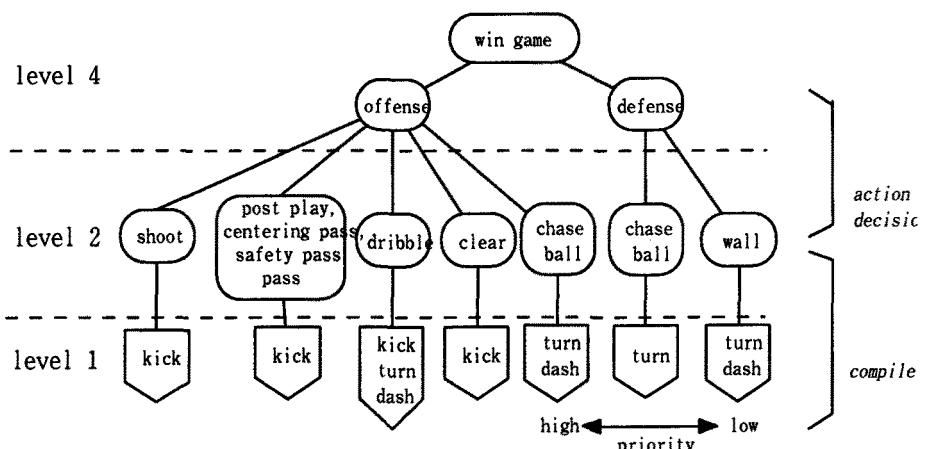


Fig. 2 An action tree used by Team Miya.

2.5 Action Tree

According to the role given to the agent, each agent will have its own action tree based on a modified hierarchy shown in Table 2. An agent's next action is specified by prioritized rules organized into its own action tree. An example of an action tree is shown in Fig. 2. In Fig. 2, if the node offense is selected, the firing conditions of action nodes at level 2 are checked from the left side to the right side. The more to the left, the higher the priority is. The action is compiled into a series of basic commands at level 1. The process of deciding an action at level 2, and the compiling procedure, were described as *action decision* and *compile* in Fig. 1.

2.6 Safety Pass and Safety Kick

The actions of level 2 are not unrelated to one another. The actions: shoot, centering pass, post play and dribble, consist of two basic skills, the *safety pass* and the *safety kick*. The *safety pass* is a skillful pass to a receiver so that it is not easily intercepted by the opponents. Pass direction determined by machine learning techniques is quite laborious[SV96]. But the effectiveness of a learning system may depend on its learning data, in this case, the opponents' behavior. For this reason, we use the following rules to determine which teammate to pass to. First, the distance between passer and receiver should be larger than 8 and smaller than 25. Second, a receiver should be forward of the passer and in the visual field of the passer. Third, the angle D, which is an angle between a receiver and the nearest opponent, should be more than five degrees. These three conditions are illustrated in Fig. 3.

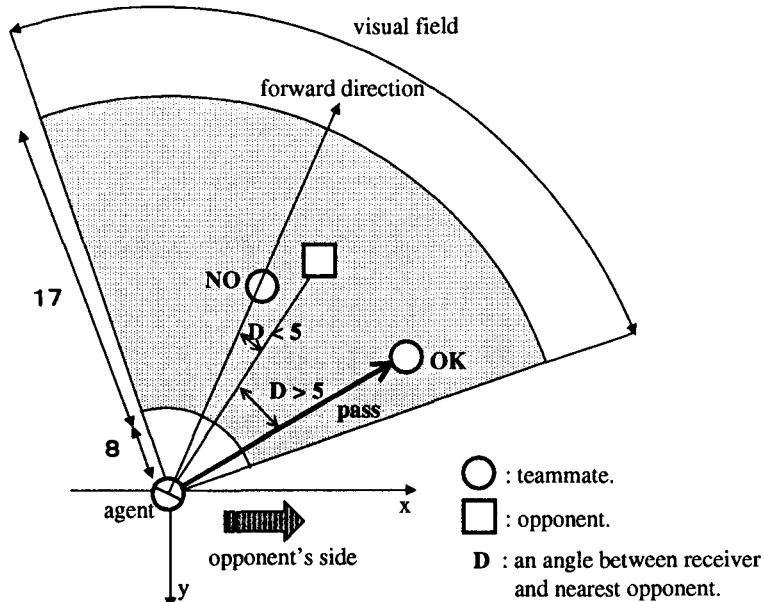


Fig. 3 Safety pass used by Team Miya.

The *safety kick* is a skillful kick, without being intercepted by the opponents, in the direction of the objective. In Fig. 4, an agent has a visual field with an angle of 90 degrees in its forward direction. An agent will kick a ball in the forward direction. As illustrated in Fig. 4, we divide a region with an angle of -35 degrees to +35 degrees in the forward direction into seven equal fans. An agent searches the seven fan regions for opponents and selects the fan that has no opponents and is closest in the forward direction. The agent kicks the ball into the center of the selected fan region.

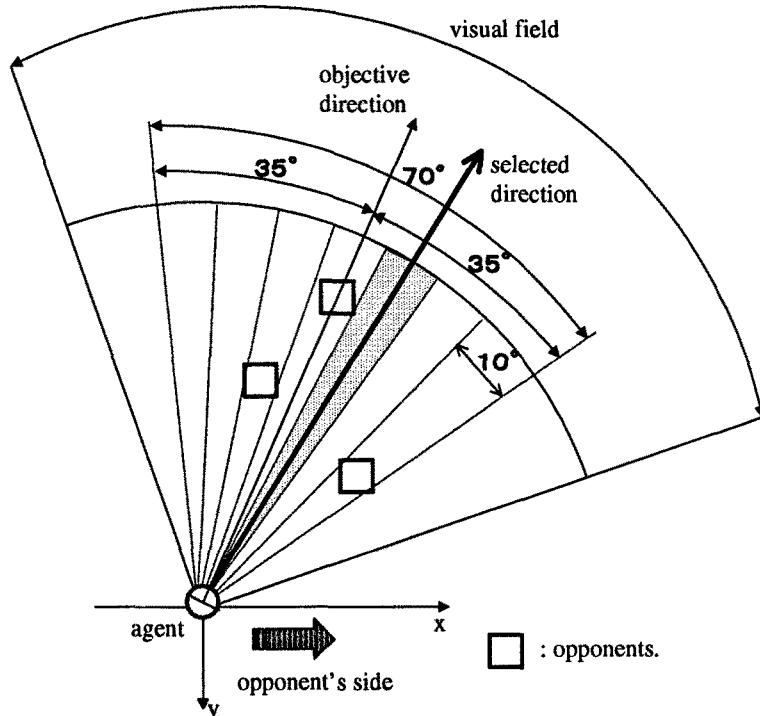


Fig. 4 Safety kick used by Team Miya.

2.7 Team Miya Results at RoboCup 97

In an algorithm used by Team Miya, information for controlling soccer agents is expressed in an action tree. It is easier to change an agent's control due to the action tree expression. In addition, individual tactical plays do not require communication between players, so the speed of passing was rapidly increased in RoboCup 97 games, and the team sometimes behaved as if it had been taught some tactical plays. Team Miya proceeded to the quarterfinal match and was one of the best 8 teams in the simulator league.

3 Team Niken

3.1 Technical features of Team Niken

Team Niken has the following three technical features in order to speed up reactions and increase flexibility in action decision. First, an agent decides on an action on the basis of a short-term goal and current information. Second, the agent's short-term goal is determined by a hierarchical decision tree called a *goal tree*. Third, Team Niken adopts only simple individual skills for quick reactions. In some cases, an agent passes a ball in a fixed direction specified by the role of the agent.

3.2 Overview of an agent system in Team Niken

An agent's behavior on Team Niken is shown in Fig. 5. Each soccer agent receives visual and auditory information from a soccer server. A short-term goal in the lowest level of the *goal tree*, defined in the next section, is determined by the current information received from the soccer server. When an agent receives a set of new information, it decides the next action from the current short-term goal and the new information. In Team Niken, information in the past is used to symbolize a current short-term goal and is used to decide an agent's next action.

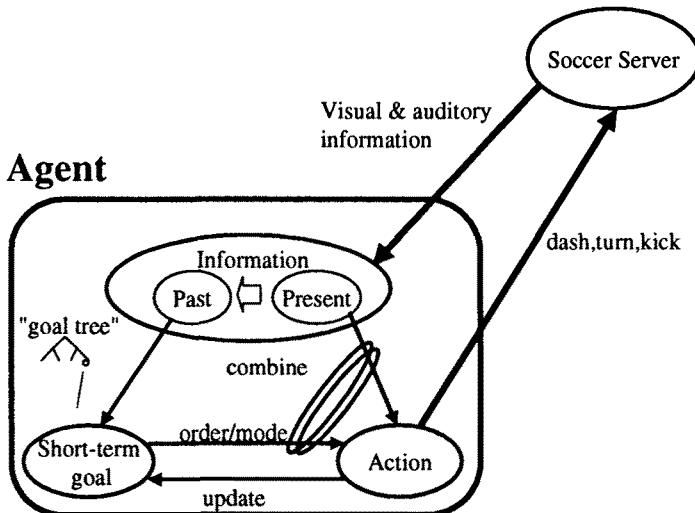


Fig. 5 Overview of an agent system used by Team Niken.

3.3 Goal tree

For Team Niken, we use a *goal tree* (Fig. 6). However, a goal tree is not the action tree used by Team Miya. For the goal tree in Fig. 6, nodes in the lowest level represent short-term goals the agent has to try to achieve. An agent selects a short-term goal from a goal tree by using the current information received from the soccer server agent.

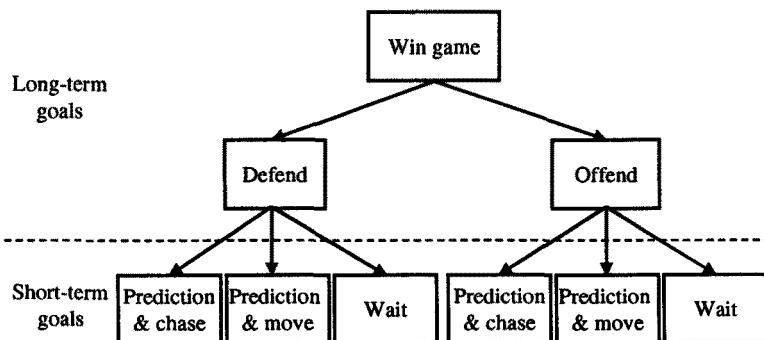


Fig. 6 A *goal tree* used by Team Niken.

3.4 Short-term Goals

The short-term goals used by Team Niken are summarized with definition and objective in Table 3.

Table 3. Short-term goals used by Team Niken.

Goal	Definition	Objective
Prediction and chase	A player predicts a position where the ball will stop and moves there.	Increasing the probability of getting the ball.
Prediction and move	A player predicts a ball's motion, decides on a profitable position, and moves there.	Better positioning for offense or defense.
Wait	A player does not move until the current short-term goal is updated.	Retaining stamina and avoiding an unnecessary chase for a moving ball.

3.5 Team Niken Results at RoboCup 97

Agents on Team Niken showed very quick responses to dynamically changing situations. Team Niken passed the preliminary round and proceeded to the final tournament. As Team Niken lost the 1st round game of the tournament, it resulted in one of the best 16 teams in the simulator league of RoboCup 97. We used only simple individual actions in Team Niken to speed up agent reactions, but we soon realized that more systematic team play is necessary to win games against the high-level teams at RoboCup 97.

4 Summary

Team Miya and Team Niken were designed to quicken reactions in a dynamically changing environment. Therefore, all of the client programs for the 11 agents ran and worked well even on a single workstation at RoboCup 97. However, we believe that our next goal should be the creation of a team of client programs equipped with Artificial Intelligence techniques such as machine learning, inference and coordination in a multi-agent system.

References

- [CK1] Coradeschi, S., Karlsson, L.: A decision-mechanism for reactive and cooperating soccer-playing agents. Workshop Notes of RoboCup Workshop, ICMAS 1996.
- [T1] Tambe, M.: Towards Flexible Teamwork in RoboCup. Workshop Notes of RoboCup Workshop, ICMAS 1996.
- [SV96] Stone, P., Veloso, M.: Using machine learning in the soccer server. Proc. IROS-96 Workshop on RoboCup(1996)19-27.

The Reactive Motion Planning in the Passive Situation

Susumu Takaki

Chukyo University,
101 Tokodate, Kaizu-cho, Toyota, 470-03 JAPAN
E-mail:takaki@grad.sccs.chukyo-u.ac.jp

Abstract. The HAARLEM¹ is a team of the agents which is designed to play soccer. It is put emphasis on decision of behavior in the passive situation and it is set importance on the defense. The tactics of soccer are classified into three levels, individual tactics, group tactics and team tactics. The feature of the defense is embodied in the level of group tactics. The effectiveness of the tactics were demonstrated in the RoboCup-97, because the goal against was relatively low.

1 Introduction

In the world in which multiple agents behave, the agents are distinguished to two types. One is active agent which has initiative action. The other is passive agent which doesn't have initiative action. This distinction of two types varies dynamically in each situation and the degree of initiative also varies dynamically in each situation. The passive agents need to select reactive motion to the behavior of other agents.

The tactics of soccer are classified into three levels.

1. Individual tactics:

The action of single player in each situation
Free-running, pass, shoot and dribble, etc.

2. Group tactics:

Effective combination and application of individual tactics as a group in the situation of both offense and defense
Combination play, one-two pass, etc.

3. Team tactics:

How to attack and defend as a team
Fast attack, man-to-man defense, zone defense, etc.

This paper describes how the behavior of the passive agent is decided in each level.

¹ The team name HAARLEM is originated from the team name to which the famous Dutch player "Ruud Gullit" belonged.

2 Individual tactics

2.1 Free-running to capture the ball

In the situation when an agent must touch an object, for example, the rescue at a fire or an capture of an animal, the agent needs to predict the motion of the object and to get ahead to the predicted position. In the soccer this situation corresponds to the case when the player chases the ball.

When the player is going to kick the ball, the distance between the player and the ball needs to be short. The ball seldom approaches to the player from itself. Usually the player has to approach close to the ball. The player can capture the ball more effectively by predicting the motion of the ball and by getting ahead to the predicted position than by chasing the ball directly.

In the designed system, the client corresponding to the player decide how many steps does it take for the player to catch up the ball according to the distance between the player and the ball. The client also predicts the position of the ball after T, where T is simulation step in Soccer Server, steps by the use of the absolute velocity calculated by the two past position of the ball (Fig.1). The client approaches the ball while modifying the predicted position with every time that the client recognizes the visual information.

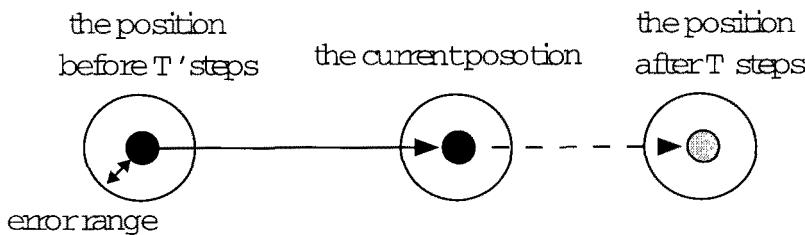


Fig. 1. The predicted position of the ball

2.2 Pass

Like to dribble, to pass is one of the fundamental play to move the ball in the soccer. The client needs to find the straight line to the pass course to avoid colliding the obstacles, that is, other players. To find the course, first, the client will prepare the circles as other players' sphere of influence. The center of the circle is located at the position of each player and the size of radius of the circle is proportional to the distance between each player and the ball. Second, the client draw the tangent lines to the circle. Because there exist two tangent lines to a circle in general, there emerges a region surrounded by two lines and

the circumference. If the center of the circle corresponds to the position of the opponent, the region means where the client must not pass the ball. Conversely, if the center corresponds to the position of the mate, the region means where the client may pass the ball. Third, when there exist more than one region where the client may pass, the client choose one out of them according to the distance between the mate and the goal, the distance between the mate and client itself and the size of the region (Fig.2). Finally, as soon as the client finds the pass course to the mate, the client tell the position to the mate with the *say command* and actually pass the ball by *kick command*.

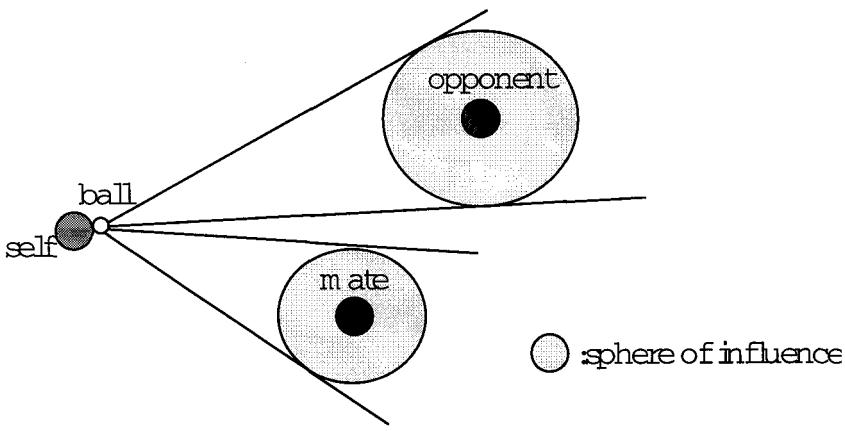


Fig. 2. The passable region and unpassable region

3 Group tactics

It is important that the multiple agents should have the purpose in common and should make appropriate actions in the multi agent system. In soccer game, the case in which the multiple agents have the purpose in common in the passive situation is often found when the team have to defend the goal, that is, in the defensive mode. It is required that the defense have to react properly against the actions which the opponent made in real time. The defense has a major purpose to defend the goal and at the same time it has several sub purposes, for example to recapture the ball, to close the shoot course of the opponent and so on. The agents have the major purpose in common, but they may change the sub-purpose according to the situation. The skill of the defense can be so easily evaluated whether they have points lost or not.

This system concentrated on the defense in front of the goal. The designed system prepared several patterns of formations concentrated to the center in

order to keep more players of the mate than those of the opponent just before the goal. The system selects the pattern of formations in Fig.3 according to the position of the ball, that is, whether the ball comes to the side or to the center. Fig.3 (a) shows the case when the ball is, at the left side near the center, (b) show the case at the right side near the center, (c) show the case at the left side near the touch line and (d) show the case at the right side near touch line. The formation enables to close the shoot course of the opponent or the pass course into the penalty area.

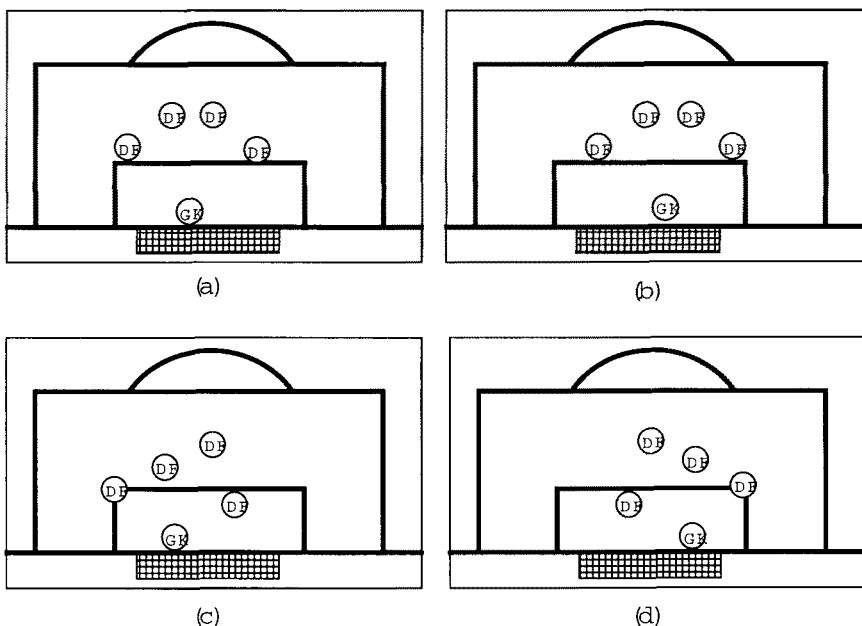


Fig. 3. Defense formation

4 Team tactics

It is important that the roll of each agent can be changed dynamically in the multi-agent system. The designed system can change the roll of the agent in the play by having many elements in common other than the basic position of the player and the priority of the decision of action. In the system, the strength of the opponent might be estimated by the difference of the points obtained and the points lost. The system can change the formation the number of the defense in the middle of the play according to the strength(Fig.4). Because the

system prepared two basic positions each corresponding to the offense and the defense, the players can keep from spreading from the forward to the defender and the distance from the backmost player and the frontmost player can be kept relatively short, thus the more players can be present.

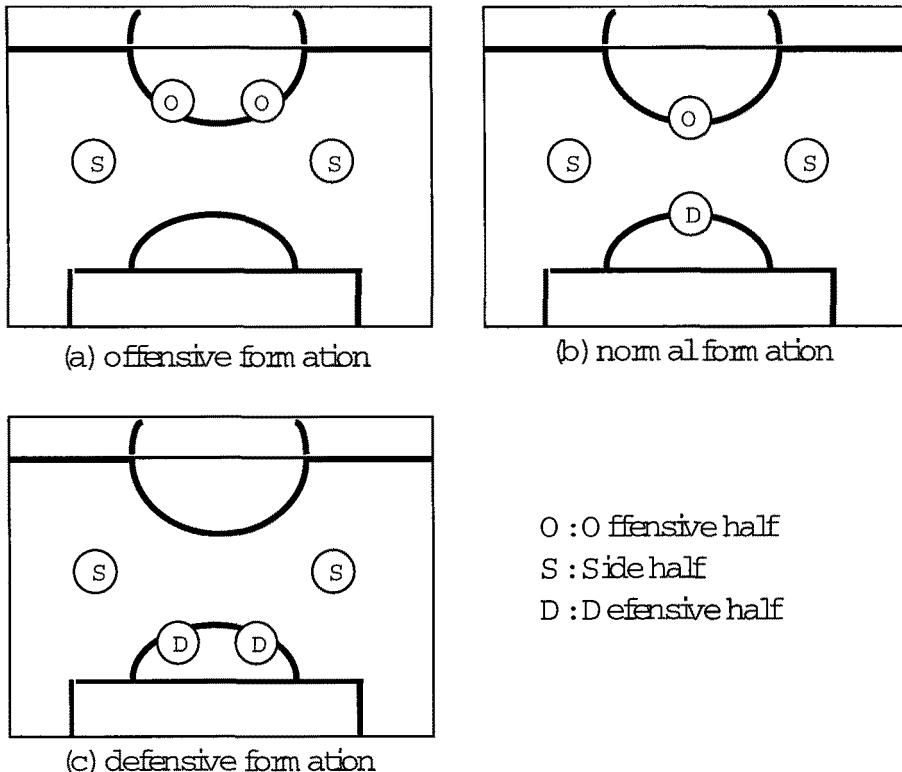


Fig. 4. The formation in the middle of the play

5 Results

5.1 Individual tactics

The designed system calculated the absolute velocity with the use of only two past positions obtained by the visual information. But visual information contains some error, there emerged the difference of the predicted position and the actual position. Then, the designed system often predicted wrong positions (Fig. 5).

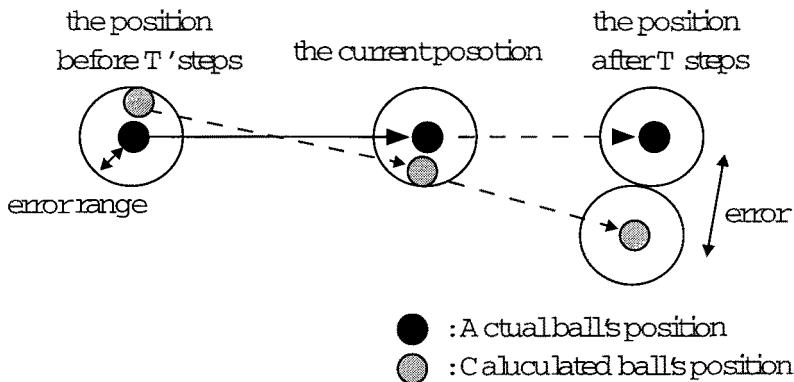


Fig. 5. Error in the predicted position of the ball

5.2 Group tactics

The system selects the prepared patterns of formations. While the selection was made only based on the numerical information, the formation tends to collapse when the ball comes near to the border.

5.3 Team tactics

To keep the distance from the backmost player to the frontmost player shorter, it is often required that the players run back and forth on the field. As a result, the stamina of the player tend to be exhausted. In RoboCup-97, the system gave the players a few seconds' break in out of play mode to recover the stamina.

6 Conclusion

As a whole, the system was successful in reducing the points lost. But the prediction in the free-running in the individual tactics was slightly wrong, then the points lost were much more than expected as a result.

References

- [Noda, 1996] tuki Noda, Yasuo Kuniyoshi. *Simulator track and Soccer Server*. bit, Vol.28, No.5, pp.28-34(in Japanese).
- [Noda, 1997] tuki Noda. *Soccer Server Manual rev2.00.*: <ftp://ci.etl.go.jp/pub/soccer/server/manual.r2.00.ps.gz>. 1996.

A Reactive Architecture for RoboCup Competition

E. Pagello¹², F. Montesello¹, A. D'Angelo³, C. Ferrari¹

¹ Dept. of Electronics and Informatics, Padua University, Italy

² Inst. LADSEB of CNR, Padua, Italy

³ Dept. of Mathematics and Informatics, Udine University, Italy

Abstract. We illustrate PaSo-Team (The University of Padua Simulated Robot Soccer Team), a Multi-Agent System able to play soccer game for participating to the Simulator League of RoboCup competition. PaSo-Team looks like a partially reactive system built upon a number of specialized behaviors, just designed for a soccer play game and generating actions accordingly with environmental changes. A general description of the architecture and a guideline of main ideas is presented in the paper, whereas a more detailed description of actual implementation is given in the appendix.

1 Introduction

We have experienced that a soccer game should successfully deal with the reactive phase before any attempt to explore the reasoning and planning phase of the game. This approach stems from the observation that at each time of the game, *scoring a goal* can be easily recognized by every player to be an individual target, besides a global one. This allows the single agent to try, whenever possible, to score the goal directly. This is not the case of a situation such as a robot engaged to reach a fixed location in a well structured environment, like a floor inside a building with many offices. Before starting any useful action, such a robot should acquaint with the world building a map it shall use later to find out how to fulfil this task. Thus, it needs a planning phase because it has no way to know immediately what action to perform towards the achievement of the goal. In the simulated soccer game, instead, at each time of the game every player knows how to do to realize its task, by using only perceptive information. This means that it is possible to consider every player to be able to bring to completion his own task. In this perspective we have faced the design of a MAS starting with a reactive approach.

In this paper we illustrate how we solve the problem of coordination among agents using an implicit communication approach whose motivation stems from the simplified assumption that a number of low level reasoning capabilities can be endowed into the system..

2 Arbitration and Implicit Coordination

As it has been pointed out in literature [1] a sound arbitration mechanism is the base for an appropriate performance of a behavior-based autonomous sys-

tem. In our case a further difficulty arises, due to the simultaneous presence of several playing agents in the same environment. Starting from the pioneeristic subsumption architecture originally devised by Brooks [3], a number of innovative behavior-based systems have been proposed in literature (Connell [4], Maes [7], Anderson [1], Kaelbling [6]).

Their proposals are dominated by the concept of arbitration which results in an either spatial or temporal ordering of behaviors. The former causes the concurrent activation of a set of primitive reflexive behaviors, also referred to as static arbitration, the latter brings about a sequential activation of different sets of primitive reflexive behaviors, also referred to as dynamic arbitration.

However, because the inclusion of temporal ordering appears too problematic when it is devised within a general multiagent framework, we have implemented a static arbitration as a special purpose behavioral module where pre-processed sensor data are always channeled to discriminate a candidate skill to be enabled as a response to typical perceived patterns. Every time sensor data are directly channeled between the perception block and the selected behavior, this behavior is activated whereas the remaining ones are inhibited.

The resulting architecture, shown in fig. 1, resembles partly the proposal of Anderson and Donath [1] and partly that of D'Angelo [5] in what the collection of boolean values (flags), updated using information supplied from sensor data pre-processing, defines a coarse-grained global state of an agent which controls behavior switches as a rough inhibitor/activation mechanism.

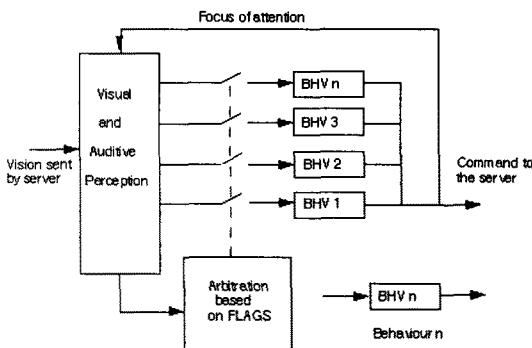


Fig. 1. agent arbitration

At each time of the game, every team player will be enabled with a behavior which depends on both its current position and orientation in the soccer game field and the pattern of the objects the agent is aware of. So, though every agent has the same cloned structure, it does not need to be activated with the same behavior. This means that is the world that makes differences among the agents.

Now how can we get any *coordination* among agents equipped with this

sketched architecture? A MAS fully distributed architecture that uses a behavior-based approach is proposed by Parker [9] that consider only the cooperative side of coordination among heterogeneous mobile robots, with attention to fault tolerance.

Another approach is proposed by Rude [10] with the IRoN architecture. He considers two kind of cooperation among robots, namely via implicit or explicit communication. With implicit and explicit communication we intend, as in Rude a passing of information respectively non-intentional and intentional. The first is realized “looking” at the external behavior of the other agents, without those agents “planned” to transmit whatever information. The second is realized sending volunteerly explicit coordination massage to the other agents. In our approach we intend to fully exploit only the *implicit communication* to perform a higher-level quality of cooperation and competition too.

3 The Basic Behaviors

The behaviors used in building the body structure of the agents have been developed to capture the abilities of a real soccer player. We have mainly tried to reproduce low-level skills.

In a soccer game individual skills are the base that every team needs before trying any kind of team strategy. Furthermore, we have noticed that the subgoal of scoring a goal is performed, at last, by a single player and this means that though it is wrong to consider a soccer as a single-player game, the acquisition of individual skills becomes a primary task.

The system implemented by Veloso and Stone [11] allows the players to learn low-level skills, as shooting to the goal, or intercept the moving ball, using neural networks. On the contrary, we have chosen an analytical approach, using the motion laws featured by the server to realize the simulation. The behaviors are described in appendix.

4 The Coordination Model

To extract significant information useful for coordination and representing a sort of *global* state of the agent or of the system, we use, immediately after every received visual or auditive message, a function named *estimate_state*, that builds for each player the data structure representing the world as the player saw it last time. Every player needs an individual execution of this function, because every player has his own view of the world. This data structure contains the last absolute seen positions, speed and orientation information of each mobile object in the field and other useful information. Using this data structure a player is able to extract, via dedicated functions, the flags used in the arbitration, like *kickable*, *ball_stolen*, *near_ball*, *passage*. Those represent individual states for a single agent. Another flag is significant to understand the arbitration: the *attack/defence flag*. This flag, that is one of the descriptors of the *global* state of the system, beside

that of the agents, is used to identify the attack or defence states of the team (*attack/defence playmode*). The flag is set when a team-mate become owner of the ball. Subsequently the same player sends an auditive command to all the players presents in the hearing area. This command, sent to the server, causes an auditive perception to all the players standing up within a 50 meters radius circle, and consequently sets the attack/defence flag.

The basic coordination mechanisms induced by the arbitration involve usually two behaviors at a time. In the attack playmode, such behaviors characterize two players belonging to the same team, that is the ball-holder and a potential receiver of the ball. We want to realize the simultaneous activation of the behavior *playball* for the player with the ball and the behavior *smarcamento* for the next ball-holder candidate. The coordination arises through the simultaneous activation of the pair {bhv_1, bhv_2} for any pair of players, where, during attacking, bhv_1 = *playball*, bhv_2 = *smarcamento*, whereas, during defending, bhv_1 = *bhv_X*, bhv_2 = *interdict*. The behavior *bhv_X* is not a real implemented behavior, but it is referred to the apparent behavior of the opponent. Actually, the opponent estimated behavior is *chasing the ball*, so our player must compete against it.

To realize this kind of interaction we have built a rigid arbitration which choose a candidate behavior looking at the *global flags* representing particular states of the whole team of agents, and at some *local flags* related to states of a single agent. In this way the emerging cooperative behavior appearing during the game may be considered as an *eusocial behavior*, a collective behavior due to the interaction of “genetically” determined individual behavior, as discussed in McFarland [8]. The proposed arbitration is an hint for the emerging of a social behavior, like an ant colony that seems to be a relatively smart being even if formed by a finite number of pure instinctive individuals. The emergence of some sort of intelligent behavior like triangulations or non explicit pass arise mainly from the interaction between the single players and the environment, namely in our case between our players and the opponents, exploiting advantageously their dynamics, as shown in fig. 2. In (1) an enemy defender chase the ball (hypothized reasonable behavior assumed by the enemy player nearest to the ball) owned by a friend player activated on the *playball* behavior. In this way the defender creates a free area in its rear. So in (2) a forward, activated on the *smarcamento* behavior, find the new hole (closed dotted line) and go to take position. With (3) the coordinated action is closed. The ball’s owner see the forward in a good position and make the pass. Without communication.

There are other two kind of coordination realized, this time only among our players. The first concerns the coordination among our player in defence play mode. When the opponents are playing the ball, our players apply a negotiation mechanism to select the teammate that must make pressing, chasing for the ball. This lets the other teammates free to assume another useful defensive behavior like *interdict*.

The second one concerns the coordination of the whole team, during the game. It is realized changing dynamically the default_pos of every player in the

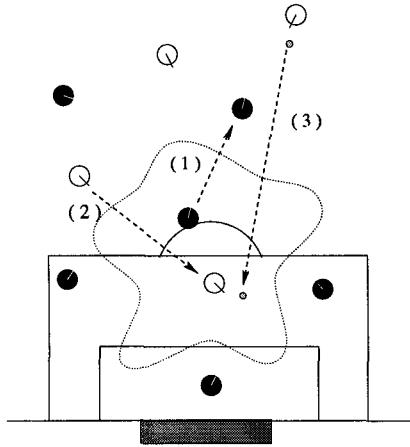


Fig. 2. observed *emergent* pass

team, accordingly with the ball position. This drag all the players, according to their roles, to follow the ball in case of attack or to back gradually to the defence position in case of defence.

5 The PaSo-Team

A full description of the implementation of PaSo-Team Clients (The University of Padua Simulated Robot Soccer Team) is given in [2]. For a brief description see the appendix.

6 Conclusions

We started by assuming that the main characteristic of soccer game is highly reactive, and from the fact that in the robot soccer competition all clients have a clear understanding of their task. We solved the problem of coordination among agents using an implicit communication approach, without using any form of reasoning about agent's intentions. We used this approach for designing PaSo-Team, our multi-agent system for RoboCup, relying on the following criteria:

- The coordination among agents is realized through a cooperative or competitive interactions respectively with team-mates and opponents.
- Flags setting gives flexibility to PaSo-Team performance, allowing to generate different soccer team able to play games with different ability.
- Arbitration is done separately over each agent, but cooperative coordination is obtained between pair of agents by ball exchanges, among several agents by pressing negotiating, and over all team by dynamically changing default positions.

- The team architecture really exploits the interaction dynamics among the agents and the environment, and if possible exploits the emergent collective behavior, without using explicit communication to realize coordination.

Acknowledgements

This research could not be done without the enthusiastic participation of the students of Electronics and Computer Eng. Undergr. Div. of Padua University Eng. School. Financial support has been provided by both CNR, under the Special Research Project on "Real-Time Computing for Real- World" and Murst, under the 60% Grants. A particular thank is due to the Industrial Firm Calearo S.R.L., a car aerials and cables manufacturer, located in Isola, Vicenza(Italy), that have provided coverage of all expenses for participating at IJCAI Conference. We like to thank also Padua Branch (Italy) of Sun Microsystems, that provided us freely a SUN ULTRA1 for developing PaSo-Team.

References

1. T.L. Anderson and M. Donath. Animal behaviour as a paradigm for developing robot autonomy. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 145–168. The MIT Press, Cambridge (MA), 1990.
2. F. Bidinotto, A. Bissacco, M. Dal Santo, W. Frasson, S. Griggio, A. F. Grisotto, S. Marzolla, F. Montesello, and E. Pagello. Implementing a soccer client team for robocup '97 competition. Technical report, LADSEB-CNR, Padua (I), 1997.
3. R. Brooks. A layered intelligent control system for a mobile robot. *IEEE J. on Rob. and Aut.*, RA-2:14–23, Apr. 1986.
4. J. H. Connell. *Minimalist Mobile Robotics*. Number 5 in Perspective in Artificial Intelligence. Academic Press, 1990.
5. A. D'Angelo. Using a chemical metaphor to implement autonomous systems. In M. Gori and G. Soda, editors, *Topics in Artificial Intelligence*, volume 992 of *Lecture Notes in A.I.*, pages 315–322. Springer-Verlag, Florence (I), 1995.
6. L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 35–48. The MIT Press, Cambridge (MA), 1990.
7. Pattie Maes. Situated agents can have goals. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 49–70. The MIT Press, Cambridge (MA), 1990.
8. D. McFarland. Towards robot cooperation. In *From Animals to Animats 4, Int. Conf. on Simulation of Adaptive Behavior (SAB-94)*, Brighton, 1994.
9. L.E. Parker. Alliance: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *Proc. of IROS'97*, pages 776–783, 1994.
10. M. Rude, T. Rupp, K. Matsumoto, S. Sutedjo, and S. Yuta. Iron: An inter robot network and three examples on multiple mobile robots' motion coordination. In *Proc. of IROS'97*, pages 1437–1444, Grenoble, Sept. 1997.
11. P. Stone and M. Veloso. A layered approach to learning client behaviours in the robocup soccer server. In <http://www.cs.cmu.edu/afs/cs/user/mmv/www/papers/AAI96.ps.gz>, 1997.

Appendix

Co-authored by *F. Bidinotto, A. Bissacco, M. DalSanto, W. Frasson, S. Griggio, A.F. Grisotto, S. Marzolla, F. Montesello, E. Pagello* - Dept. of Electronics and Informatics, The University of Padua, Italy

Client Structure

Our software client built for RoboCup competition is composed of various elements, each working at different steps. An important aspect to care of is the reception and transmission of commands from and to the server, that takes place using a socket. The communication via socket requires a continuous control of the state of the socket itself, because of the unpredictability of the exact instant of the visual perception sent by the server. Each client controls the socket looking at a SIG-IO, given by C++ library, that is activated when messages are present in the receiving queue. A signal handler pops the message from the queue and sends it to the parser which extracts the suitable information. All communication are realized using the UDP/IP protocol.

Every behavior, activated by a client, generates commands for the server, that are stored in another queue, the command list, using the same protocol. The timing of reading this list is provided every 100 ms by a timer synchronised with the SIG-IO controlling the socket. In such a way, commands are sent to the server without loosing a single command.

Estimators

When the information are received from the parser, which stores the relative visual information for each player, another function, named *estimate_state* extracts from the relative data the absolute ones. These data allow to reason about the absolute *position*, *orientation* and *speed* of the players and every other object in the field.

The estimate of the positions and orientation, using the relative information sent by the server, is done with a geometrical approach, looking for every fixed object and making triangulation to recover the absolute ones. Extracting this information is not simple, because sometimes, when the player is in the neighbourhood of field border, the only visible objects are the side lines, which provide too few information to allow to retrieve a reasonable estimate.

Therefore, tree different situations are evaluated, relatively to the available relative visual information:

- The first case happens when a player watches two fixed objects. This allows a good estimate of the absolute position followed by the estimate of the absolute orientation
- The second case happens when a player watches only one fixed object and a border line. Then, we estimate first the absolute orientation and later the absolute position. The estimate is worse than the previous case.

- The third case, the worst one, happens when no fixed objects are visible, or one single fixed object only is available, but no border lines. Then, it is impossible to retrieve whatever kind of estimate. Thus, we use the memory of the past sent commands to estimate the current position from the available last one.

For acting in and reacting to the environment we used both absolute and relative information. For chasing the ball, turning with ball, we used relative information. We used the relative approach for this skills because of the better accuracy of the estimated values. With this approach we can evaluate the trajectory of the moving objects (usually the ball) predicting the future positions of the ball itself and trying to intercept or control it.

Behaviors

The following ones are the behaviors implemented to provide the interaction with the environment. The only behavior that does not generate commands for the server is the *arbitrate* behavior. The remaining behaviors are all sending commands.

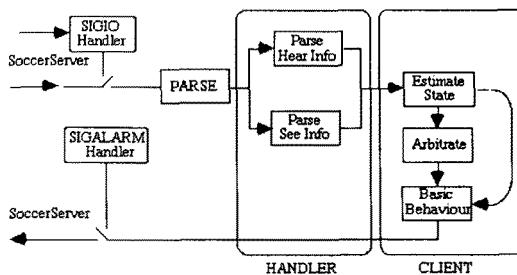


Fig. 3. agent architecture

arbitrate() It is always active. It analyses the flag structure and activates the right behavior. It is implemented in a rigid manner, actually using sequences of *if* statements.

playball() It provides a player with the commands able to realize the correct actions when it owns the ball.

This means:

- 1. taking the ball in a *free from enemies* direction
- 2. deciding if passing or not the ball
- 3. testing dangerous situations. If running with the ball (or passing it) becomes too dangerous, then the client throws the ball in a *free from enemies* direction.

chase() It forces the player to chase the ball.

This may happen in two ways, named stamina-preserving and stamina-consuming. The former is invoked, when the ball is far whereas the latter, when the distance to the ball is lower than 7 meters.

goto_defaultpos() It moves players to the default position according to the invoked *schema*.

The *schema* depends on the ball position. This means that for an attacking team the player's default positions change with the ball position, so that the team itself is lined-up in the right manner to perform a coordinated attack action.

pressing() Activated in defence playmode, it allows the coordination of a team when the problem to be solved is *who must chase the ball?*.

We solved this problem via a negotiation based on visual and auditory information. The selected player makes pressing, chasing the ball that is in possess of the opponents.

defence() Activated in defence playmode, it provides the defenders, the wings and the midfielder with the skill of controlling and breaking the enemy's play.

This task is realized in two ways:

- 1. wings and midfielder: the players move dynamically between the ball and the enemies, to interdict the play. A mechanism select which one, among our players, has to take care of each enemy.
- 2. defenders: the outsiders dynamically keep a position between the enemy forwards and the nearest goalpost, to avoid a direct shoot to goal. The insiders keep symmetrically a position that allows to control an approaching central enemy, avoiding field areas to be not defended.

line-up() It is a defensive behavior for forwards and moves themselves to a default position.

smarcamento() It is an attacking behavior for the players without ball. It moves dynamically the players toward the most free area, preparing them to receive a passage. It is based on a random search algorithm that looks for a minimum of a function in a 2-D space. This function represents the degree of freedom of the player in the field.

Team: *Kasuga-bitos* with Modulation of Playing

Tomoo Inden and Tomoichi Takahashi

Chubu University, 1200 Matsumoto Kasugai-shi Aichi 487, JAPAN

Abstract. Two points are paid attention in implementing our team *Kasuga-bitos*. One is an agent changes its playing style as a game proceeds. The other is that agents cooperate each other using only visual information without any programmed protocols. Their effects are estimated by games with Ogalets, the champion team in PreRoboCup 96.

1 Introduction

A soccer game is a kicking game which object is to score more goals than an opponent team. A soccer game is played with two teams. A team is composed of 11 players. One player is a goalkeeper. The goalkeeper is a special player who is in front of the goal and defends his goal from the opponent team's attack. The other 10 players play in the field to get a goal. The game is mainly made up by the players in the field.

The techniques of players are classified into several levels.

fundamental techniques: Each player should master techniques of controlling the ball. The controlling techniques are kicking, dribbling, passing, intercepting, etc.

advanced techniques: Good players can decide immediately which play is most suitable to changing situations. They shift from one technique to another smoothly, for example from dribbling to passing.

tactics of game: Players should cooperate each other to get a goal and to defend their goal. Coaches dictate players how to move cooperatively during the game.

The techniques in the first two levels are players' abilities. The last level refers to the abilities as a team. These techniques are common to real games and simulation games.

This paper discusses fundamental and advanced techniques in section 2. In real soccer games, coaches outside of the field suggest players to move in a better way. Playing style of a soccer agent also should reflect the situations which change as a game proceeds. Player's style modulation are discussed in section 3. In section 4, the agent movement is discussed in terms of computer resource allocation problem. Our playing style modulation methods are discussed using games between our team, *Kasuga-bitos*, and Ogalets. Ogalets is the champion team in PreRoboCup. [1]

2 Player's techniques

Agents in simulation track are provided with fundamental techniques as control commands such as **kick**, **dash**, etc. The agent should be programmed

- to select suitable commands and to set their parameters,
- to combine the fundamental techniques.

2.1 Players basic model

Each player decides these procedures in primitive (**sense - plan - action**) cycle. The planning procedure is modeled as a finite state automaton (Q , Σ , δ , d , g). Fig 1 shows state transition diagram, δ . Circles indicate player's states, $Q = (a, b, c, d, g)$, where d is an initial state and g is a final state.

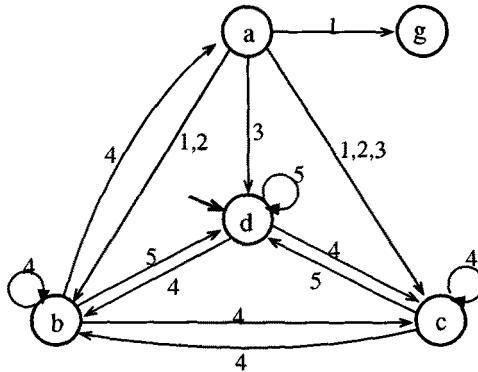


Fig. 1. state transition diagram

- a: The agent can kick the ball. The parameters of kick - direction, power - are calculated by the player's position.
- b: The ball is near the agent. The agent should dash to control the ball.
- c: The ball is out of the region which is assigned to the agent. The agent returns to and stays the default position.
- d: The ball is out of the agent's sight. The agent turns to look for the ball.
- g: The ball is in the goal.

Table 1 shows input alphabet Σ . Input alphabet is sensor information edited in a five-tuple form - (shoot condition, pass condition, obstacle-in-sight condition, ball-in-sight condition, ball-in-field condition). The agent can shoot or pass if shoot condition or pass condition is satisfied. Obstacle-in-sight and ball-in-sight conditions indicate agents of the opponent team or the ball is in the sight.

Table 1. sensor conditioning input alphabets

condition	1	2	3	4	5
<i>shoot</i>	o	x	x	x	-
<i>pass</i>	-	o	x	x	-
<i>obstacle-in-sight</i>	-	-	o	-	-
<i>ball-in-sight</i>	o	o	o	o	x
<i>ball-in-field</i>	o	o	o	o	o

o: condition satisfied.

x: condition not satisfied.

Ball-in-field condition indicates whether the ball is dead or not. The condition in right columns constrain more globally than the condition in left columns.

The finite state automaton decides an agent's action sequence as a real player predicts consequences of his actions. The other players move autonomously. As a result, the decided action is not suitable for the new situation. The player's action sequence does not necessarily follow the transition diagram of the automaton.

2.2 Implementation of advanced techniques

Pass is to kick a ball to another player of own side. Pass' success is judged by whether another player receives the ball or not. Communication between players is required to succeed passes. There are two ways for agents to communicate with each other.

- using voice(**say-command**), one agent send information to other agents.
- using vision(**see-information**), agents know the situations.

Using **say-command** as acknowledge procedure is a reliable procedure, but it takes time and other players may be farther than voice can reach. We think that most of kicking and receiving a ball in real game are done see-information.¹

The other technique of pass is the direction of kick. Real players kick the ball with predication the next position of the other players. In addition to the kick player's prediction, receiving players expect that the ball will be passed when the other player's state is the same one as their kick state. These predictions are similar to the agents movements in the simulation game. Fig.2 shows one of such situations. There are three agents ,a1, a2, a3. They dash to a ball and one of them (for example, a1) holds it. Then the other two players should stop the dash and change other actions. They may turn and dash to the goal, excepting the ball will be passed to oneself.

¹ We admit this is disputable. According to Mr. Ando, the runner-up in RoboCup-97, *Andhill* use **say-command** as to inform other players the next action.

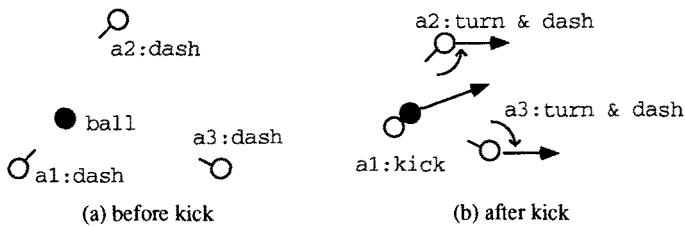


Fig. 2. Example of advanced techniques

3 Playing style modulation

Game tactics vary in the process of a game. The tactics as a team controls playing style of a player who moves autonomously.

3.1 Factors of playing style

There are several factors to change the playing style. The followings are examples of such factors.

- distance to ball:
The player near the ball should play quickly to get a goal or to defend his goal. On the other hand, the players far from the ball may move more slowly. They keep their positions for the next coming situations.
- gap between score of teams:
When score gap increases as the game proceeds, coaches give suggestions to the players from outside of the field in the real games. The suggestions are “mark the player number X”, or “change attack or defense formation to Y”, etc. The coach suggestion model should be given in advance and the agents change their own playing styles using the suggestion model.
- elapsed time of game:
When games are coming to end and the score of our side is over the opponent, the players are inclined to play defensively to keep the lead. To the contrary, the players play offensively in a case that the score is vice versa.

3.2 playing styles

There are several playing styles. Table 2 shows examples of styles.

The first position style is determined by the player's position. The second play style reflects player's characteristics. These two styles are the same ones as real players have. They are assigned to the client when it is called.

The last style indicates how quickly players move. In real games, quick movement is one of good player's characteristics². It is related to each player's physical

² soccer server version 3.05 later has a parameter, indicating an agent's stamina.

Table 2. examples of playing style

<i>factor</i>	playing style	
<i>position</i>	forwards	backs
<i>play type</i>	offensive	defensive
<i>response</i>	quickly	slowly

strength, while the agent's quick movement is also related to communication rare to the soccer server in the RoboCup games.

The last two styles change during the game. They are modeled as a function of distance to the ball and elapsed time. The combination of the modes control the agent's global movement.

4 Resource Allocation Problem

4.1 Players Sensing-Action Cycle

Including an opponent team, 22 clients join the game. The agent knows its position and game situation through communications with the soccer server. The communication are done in a form of (*sense - plan - action*) cycles with following conditions.

- c1:** sense and action are associated with communication with the soccer server.
- c2:** queuing to the soccer server is equal to all clients.
- c3:** CPU time allocation is equal to all clients.

The player's speed is proportional to communication rate between the corresponding client and the server. Under amount of communication is restricted, the game is resource allocation game among clients as well as a game of kicking. [2]

4.2 Tactics to allocation problem

Response factor indicates one tactics that players nearer to the ball move more quickly. In order to play quickly, the following requirements are satisfied.

- a1:** Agents near the ball should communicate with server more frequently than other agents.
(Agents near the ball should have higher priority than other agents.)
- a2:** Our team should communicate more frequently than the opponent.

Our idea is to allocate more resource than other agents by controlling the agent's CPU running time. Fig 3 shows the heuristic function between sleep command parameter and distance to ball. If the distance between agent and the ball is below `field_width/6`, then the agent is not slept. The distance is greater than `field_width`, one sleep unit in plan cycle is allocated to the agent.

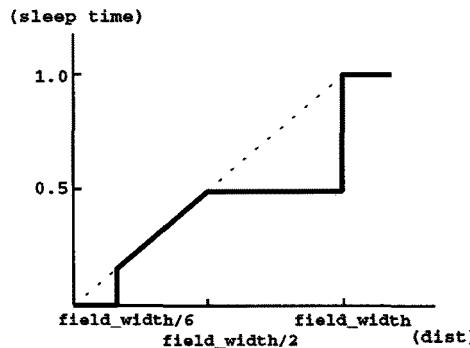


Fig. 3. sleep ratio vs. distance.

5 Experiment Result

Table 3 shows the games history of between *Kasuga-bitos* and Ogalets.[1] Ogalets is a champion team in PreRoboCup in 1996. Clients for game 1 are programmed by using only the automaton in Fig 1. Response mode changing by controlling CPU running time is installed in clients for game 2. Clients in game 3 are capable of passing the ball.

Table 3. Scores of games between Ogalets

No.	score	<i>implemented skills</i>
1.	0-15	only fundamental ones.
2.	0-9	response mode.
3.	4-3	positioning, passing between agents.

The game records show that

1. *Kasuga-bitos* can decrease the score gap in No.2 match. This score imply that response mode is effective,
2. In No.3 match, the agents can pass a ball and try to keep its position in a better point. This makes *Kasuga-bitos* a team comparable to Ogalets.

6 Summary and conclusion

Ball games players move differently as the situation changes. Our principle in making *Kasuga-bitos* is to imitate real player's motion, - its play style changes

in a process of game and the information comes mainly from its eyes. In this paper, a player's techniques is classified into three levels and the relationship between response of player's movement and resources allocation are discussed. Movement response modulation and passing skill are implemented. Their effects are estimated in games with Ogalets.

The scores show that they make *Kasuga-bitos* a good team, however it is not sufficient to be an excellent team. In the first stage of RoboCup-97, *Kasuga-bitos* belong to Group D , where the champion team *AT Humboldt* and the runner-up *Andhill* are. Matching them makes the difference to the excellent teams. Modifying finite state automaton model during games is our next purpose.

Finally, we would like express our gratitude to the RoboCup's committee for holding this Cup.

References

1. <http://ci.etl.go.jp/noda/soccer/client.html>
2. Noda Itsuki, Kuniyoshi Yasuo. *Simulator track and Soccer Server*. bit, Vol.28, No.5, pp.28-34.

Team Sicily

John Fry, Lyen Huang and Stanley Peters

Stanford University
Center for the Study of Language and Information
Stanford, CA 94305-4115 USA

Abstract. Team Sicily, our entry in the RoboCup-97 simulator track, is designed to achieve cooperative soccer behavior using a realistic and efficient balance of communication and autonomy. The team is implemented in the multithreading logic programming language Gaea, and builds on the dynamic subsumptive architecture model developed by Noda and Nakashima [3].

1 Background and Goals

For the past few years our Situated Agents group at CSLI has been working on the general problem of cooperative agent behavior, and especially on the role of communication in cooperative software architectures. In the process we have studied a few paradigmatic problems, notably cooperative elevator dispatch in an office building ([4] and [5]) and cooperative taxi dispatch in a city [5]. Our elevator simulations, for example, demonstrated that an autonomous but cooperative architecture with a small amount of communication is more efficient than the top-down centralized controller model. However, constant n -way communication among n agents is generally unnecessary and even counterproductive for all but the most complex tasks.

Our RoboCup-97 entry, Team Sicily (*Sicily* is one pronunciation of *CSLI*) represents a somewhat more sophisticated application for cooperative agents. Nonetheless, in our Team Sicily implementation we have attempted to preserve all the advantages of a cooperative agent architecture, and specifically the following five properties:

1. **Efficient Communication** Agents should communicate in an appropriate and timely manner, but only as often, and with as much information, as required to act cooperatively.
2. **Fault Tolerance** The failure of individual agents should cause graceful degradation of system performance.
3. **No Single Point of Breakdown** There should be no one component, such as a central controller, whose failure can bring down the entire system.
4. **Equitable Load Distribution** The work load should be evenly divided among cooperating agents, with no single overloaded component acting as a bottleneck.

5. Open System The system should be flexible enough to easily accommodate more agents, and the addition of new agents should not require existing ones to be reprogrammed.

The nature of the RoboCup problem (playing real-time soccer against an opponent) and the physical implementation of the Soccer Server both present interesting new challenges for a cooperative architecture approach. Team Sicily is designed to explore the role of *explicit communication* in a multi-agent system where players sequentially use a non-sharable resource (the ball) in cooperative way to obtain a specified result (get it into the opponent's goal without allowing it to get into one's own goal).

We contrast explicit communication between agents (players) with implicit exchange of information by means of reasoning from other agents' actions to their intentions. Our plan was to measure the value of explicit communication by comparing our team's performance when communication was enabled against its performance with communication disabled. Of course, the team had to conform to the RoboCup rules, an important one being that soccer players are only allowed to communicate through the soccer server—clients cannot communicate directly, but only indirectly using *say* and *hear* commands. Messages are restricted to ASCII strings of 512 bytes and only one command per simulation cycle is accepted. In such a restricted and real-time reactive system, then, the issues of *when* and *how much* to communicate take on paramount importance. Rather than issuing a *say* command at every cycle, a player agent should communicate only when there are significant changes in the environment—for example, when he becomes free and wants the ball passed to him, or is being blocked and wants to pass.

2 Program Design

2.1 Subsumptive Architecture and Gaea Programming Language

Team Sicily is implemented in Gaea [2], a multithreading logic programming language. A Gaea program consists of (1) multithreading *processes* and (2) *cells* that store fragments of programs. The cell is the level at which name to content mapping and background conditions are implemented. The *environment* of a given process is the collection of cells which are currently active for it (Figure 1).

In general we have found Gaea useful as an implementation language because (1) like Prolog, it facilitates rapid prototyping, and (2) it allows cooperative agents to be implemented straightforwardly as processes which can be forked, suspended, resumed, and killed during the course of an application.

For the Team Sicily implementation, the most important feature of Gaea is its dynamic version of Brooks' subsumptive architecture [1]. For a given process, programs in deeper cells in the environment are valid unless they are explicitly negated by programs in shallower cells. Thus the structure of the cells can be viewed as a dynamically formed inheritance hierarchy, permitting *dynamic subsumptive programming*: programs in higher functional layers override programs

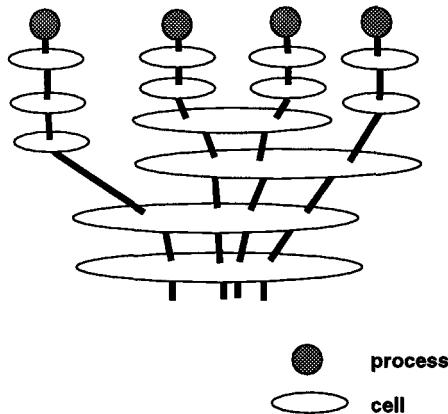


Fig. 1. Multiple Gaea processes in their own environments

in lower layers, and the system can backtrack to lower levels when a higher level program fails to perform its function. Furthermore, the cell modules can be pushed, popped, and swapped on the fly in order to adapt to new conditions without recompiling the whole system.

This technique was originally exploited for soccer in Itsuki Noda's 1996 RoboCup team [3]. Noda's implementation used Gaea's layering of modules in order to flexibly handle the interactions between high-level modes (e.g. *offense*, *defense*) and low-level modes (e.g. *chase-ball*, *dribble*, *shoot*, *pass*).

2.2 Structure of Individual Players

Our basic design uses four controlling threads for each player: *command*, *watchdog*, *listener* and *sensor* threads. Primary control belongs to the *command* thread. Here, a set of individual action cells is loaded and unloaded on top of a basic set of Gaea cells in which technical information (server address, player memory, etc.) and general team information is stored, as illustrated in Figure 2. The top cell has the highest importance in the execution tree; if no rules apply from the top cell, then the next cell is analyzed, and so on. This lets lower level cells to control basic player actions such as running and maintaining positions, unless overruled by higher cells.

The *watchdog* thread keeps track of the passage of time and the strategic development of the game. As the game progresses, the watchdog adjusts the strategy cells in the command thread in order to change the player's positions and passing routes. Similarly, the *listener* thread takes incoming messages and, after analyzing the messages and deciding whether or not to accept the communicated command, changes the cells in the command thread accordingly.

Based on each round of sensory information, processed by the *sensor* thread,

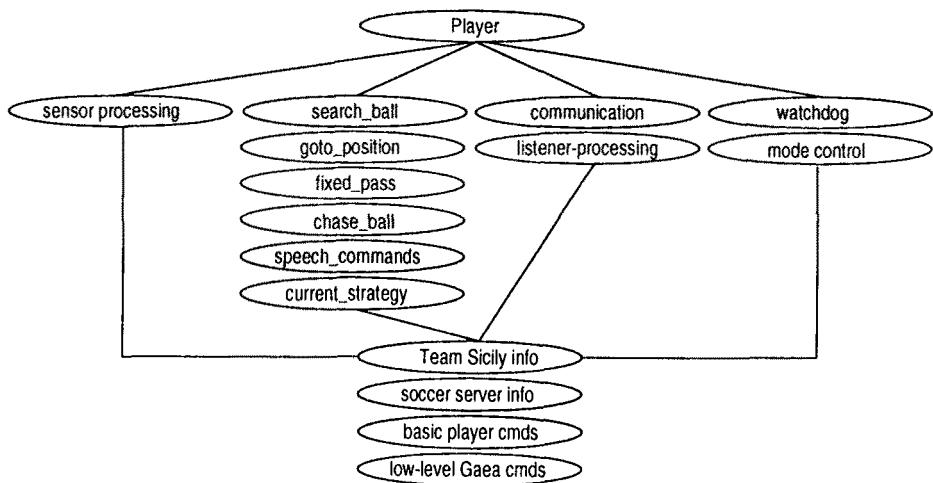


Fig. 2. Process and cell structure of a player

the player can either push an action cell onto the command thread, in order to react to a new situation, or change the priorities of actions by swapping the order of cells in the thread. This design also allows a player to delay executing a play in order to take care of more immediate concerns, like avoiding an enemy player; once the interruption has passed, the top cell is popped and the tactical or strategic cell again takes highest priority.

3 Lessons Learned

In the first round of the tournament we had the misfortune to face Humboldt University (Germany) and Tokyo Institute of Technology, the teams who went on to finish in first and second place in the tournament, respectively. In any case, Team Sicily lost all three games it played, scoring 0–25 against Humboldt, 1–4 against Chubu University (Japan), and 0–23 against Tokyo Institute of Technology.

3.1 Difficulties with Gaea

Perhaps the greatest handicap our team faced was the sluggish speed of our implementation. Since the Gaea language, which is itself implemented in Lisp, is still under development, it lacks some of the standard optimizations found in production-quality logic programming languages. Bugs and brittleness in Gaea would occasionally cause stack overflows in threads, causing individual players

to crash. While the team could still function with diminished numbers of players, it was at a definite disadvantage missing three defenders, as happened early in the tournament. One lesson that can be drawn regarding the implementation language is that for real-time applications like RoboCup, the standard logic programming operations of backtracking, recursion and unification are only appropriate for high-level reasoning. Lower-level tasks like I/O and position calculations would have been better implemented in appropriately lower-level languages.

3.2 Effects of the RoboCup Domain

Another conclusion we have drawn is that, paradoxically, increased explicit communication improves teamwork (i.e. produces behavior closer to real soccer), yet degrades performance within the RoboCup domain. This is because the constraints imposed by the RoboCup rules and the soccer server implementation serve to discourage communication. Player communication is limited to one message per round, and there is no guarantee that the intended target will hear the message if another player in the area speaks at the same time. Moreover, communicating expends as much time and effort as moving or kicking. In order to overcome these bottlenecks, we were forced to limit communications severely, to prevent players from wasting precious cycles communicating instead of chasing the ball. In the end, our players spoke between one and ten percent of the times they had the opportunity to, depending on circumstances. This compromise allowed just barely enough communication to facilitate teamwork.

3.3 Effects of Communication on Player Actions

From qualitative data based on observation of our team's performance with and without explicit communication, we observed two primary differences between the communications-enabled condition and the non-speaking one. The first difference lay in the communicating team's ability to react more quickly to a developing situation. On offense, for example, if a pass did not arrive at its target location, the players could turn and react more quickly to the ball's new location and movement when they could inform each other about the ball's position. And when the defense was able to communicate the ball's position to a player, he would react to it more effectively, especially if the ball had been kicked out of his view. Instead of wasting time searching 360 degrees for the ball, the player was able to quickly turn to the ball's last reported location and begin searching there.

The second improvement lay in the team's strategy and tactics. Without communications, the players were more predictable and less effective in dealing with their environment. Passes and movement were executed only in patterns we hard-coded into their behavior, so each successive game tended to appear similar to previous ones. With communications enabled, the games had the same overall strategic patterns as originally coded, but the individual and small-group tactical play became significantly more dynamic. The team tended to deal with enemy

players more effectively, passing around and defending against them, and this was reflected in the increased proportion of time the communicating team spent in control of the ball.

We are currently in the process of validating the above qualitative observations with a suite of quantitative tests. For example, we are looking at the difference between soccer players that store position information in an agent-centered way (as in Itsuki Noda's original program) compared with players that convert the information to an absolute (or at least, bird's eye) coordinate system and store it that way. Originally we found that the latter was less efficient because agents always had to reconvert the information to self-centered form, based on their current (possibly new) position, before using it. We hope to confirm this with statistical tests comparing situated vs. unsituated memory for this type of task.

4 Some Reflections on RoboCup and Communication

An important point brought out at RoboCup 97 is the difference between implicit and explicit communication. During the workshop, many participants downplayed the need for explicit communication and concluded that teamwork was best coordinated through implicit understanding. We are convinced, however, that teamwork and coordination are better managed by explicit rather than implicit communication, and we can't help but wonder how strongly their conclusion was biased by the RoboCup guidelines.

Consider real-world professional soccer players. Certainly verbal communication is not always an option for players to coordinate their teamwork. Among other problems, distances on a soccer field are large, and messages can sometimes be intercepted by the eavesdropping opposing team. Nevertheless, real-world soccer players do communicate extensively, albeit not always verbally. To illustrate this point: Suppose one took one player from each of several professional teams and combined them into a new team. At first they would communicate verbally quite a bit as they adapted to each other's playing style. The more they played together, the less verbal communication they would need. This does not entail, however, that the overall amount of communication has decreased; in fact, the content of the messages can remain constant even as the *medium* changes. Instead of a relatively long, slow verbal message, 'body language' becomes more prevalent. In this new 'protocol', implicit communication—based on understanding what a player intends when he makes a movement or pass in a certain way—establishes a space of possibilities within which explicit communication—now 'encrypted' into a faster medium of gestures—coordinates the details of the actual execution of the play.

To illustrate, consider the give-and-go play. Player A approaches Player B while dribbling the ball. Opponent C stands between Player A and his target goal. Player A makes a quick pass and then runs to the reflected pass from Player B. The positions of the players provide the implicit understanding on which cooperation in the situation can be based. Nevertheless, without eye contact, a

verbal cue or perhaps even the words “give and go!”, the actual coordination of the play will be difficult to effect.

Since such a protocol is heavily dependent on the familiarity players build up with each as they play together for long periods of time, the code is very difficult for the opposing team to read. Verbal communication is still necessary in situations where the target player may not be able to ‘listen’ visually. These include situations where an opposing player is approaching from behind (‘man on!’), where a teammate is nearby and ready to receive a pass but is not in the direct view of the player with the ball, and warnings about the ball’s location relative to a player who has lost track of it.

4.1 Whither RoboCup?

An observation frequently made by competitors at RoboCup 97 was that explicit communication requires too many steps to coordinate teamwork, since it requires one player to propose a play and other players to confirm its execution. If the conclusion is correct, it puts us squarely in the dilemma of determining exactly what is the focus of RoboCup. If the aim is to model the real world, then the current system of communication, we agree, constrains teams too much from using explicit communication as a key element in their strategy. The soccer server’s communication mechanism is too costly in terms of player actions (with speaking equivalent to running or kicking) as well as too limited in number of channels (allowing a player to hear only one message from his team per cycle).

If, however, RoboCup is not intended to model the real world but only to provide a forum for software and hardware agents, then explicit communication can be made an even more effective tool. Communication between software agents can take place much more quickly than verbal communication between humans. Hardware robots have the option of using wireless communications to transmit data at very high rates. Under this view, RoboCup should take advantage of the available bandwidth and enable the player agents to not only propose strategies, but also negotiate and even converge on consensus about them.

Acknowledgements This work was supported in part by the Information Technology Promotion Agency, Japan, as part of the R&D of Basic Technology for Future Industries “New Models for Software Architecture” sponsored by NEDO (New Energy and Industrial Technology Development Organization), and in part by a summer internship for Symbolics Systems students sponsored by CSLI at Stanford. We’re very grateful to Itsuki Noda, Stefan Kaufmann and Hideyuki Nakashima for their assistance.

References

1. R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–160, 1991.
2. H. Nakashima, I. Noda, K. Handa, and J. Fry. GAEA programming manual. Technical Report ETL-TR-96-11, Electrotechnical Laboratory, Tsukuba, Japan, 1996.

3. I. Noda and H. Nakashima. Multiagent soccer on Gaea. In *Proceedings of the International Symposium on New Models for Software Architecture IMSA-96*, pages 9–14, Tokyo, December 1996. IPA.
4. S. Peters and J. Fry. Real-world applications for situated agents. In *Proceedings of the International Symposium on New Models for Software Architecture IMSA-95*, pages 7–14, Tokyo, October 1995. IPA.
5. S. Peters, J. Fry, and S. Kaufmann. Communication strategies for cooperative behavior. In *Proceedings of the International Symposium on New Models for Software Architecture IMSA-96*, pages 27–36, Tokyo, December 1996. IPA.

Team Description: Building Teams Using Roles, Responsibilities, and Strategies

Simon Ch'ng and Lin Padgham
(schng@cs.rmit.edu.au) (linpa@cs.rmit.edu.au)

Department of Computer Science
Royal Melbourne Institute of Technology
Melbourne, Victoria, Australia

Abstract. The Royal Melbourne Knights are designed to interact as a team of soccer playing agents. This paper introduces a framework for modelling agents using the concepts of roles, responsibilities and strategies in its control of the agent's motivation, attention and behaviour, respectively. Through the use of strategies, an agent may form teams that coordinate their actions. The high performance team of agents designed for the Royal Melbourne Knights will compete at the real-world soccer simulation tournament, RoboCup'97.

1 Introduction

The Royal Melbourne Knights are designed to interact as a team. Using a methodology for agent-oriented design based on the abstraction of different known behaviours of an agent, called "roles", a high level specification of teamwork is produced that guides the agent's behaviour.

The motivation for specifying an agent's behaviour using roles is illustrated in team sports like soccer. Soccer has the roles of defender, mid-fielder, attacker, et cetera. that represent well defined characteristics of a players behaviour. A soccer player selects a role that creates opportunities for his team to win and reduce his opponent's opportunities. Rather than communicating strategies or tactics between players, a player identifies the role of other players and adjusts its role to create opportunistic situations for executing pre-planned strategies that coordinate their action.

Soccer show the advantage of a set of players that adopt appropriate roles based on the situation and the roles of other players. Teams are robust to changes in an player's state, e.g. player's sent off the soccer field for fouls, because the roles required to carry out the operation can be fulfilled by other players. The ability for roles to be adopted by any player and the coordination of a group players through pre-planned strategies using roles, provides teamwork with a great degree flexibility. The abstraction from the underlying mechanisms for communication and the players abilities provides the freedom to adjust the team to suit the conditions of the match without affecting the specifications for teamwork.

The following sections describe a framework for describing teamwork with roles and a model of agents that function using this description. Section 2 begins

with the definition of a role and its representation. The representation is then given an operational semantics in the execution model described in section 3. This model is used to explain the behaviour of several soccer agents in a corner kick scenario, illustrated in section 3.

2 Roles, Responsibilities and Strategies

Roles, responsibilities and *strategies* are high level concepts which we are using to develop our approach to teamwork. These concepts and the supporting concepts of *strategic groups* and *team strategies* are described below. Table 1 shows some of the roles found in soccer games and the associated responsibilities and strategies for those roles.

Role	Responsibility (priority)	Strategies
free-kick-taker	take-corner(2)	pass
attacker	get-ball(5)	pass
		tackle
defender	score-goal(3)	shoot
	defend-goal(5)	block
player		mark
	clear-ball(2)	pass
	position(4)	go-zone
	move-ball(3)	pass

Table 1. Example of roles, responsibilities and strategies from soccer.

A *role* defines the part that an individual agent chooses to play in a scenario, e.g. a goal-keeper is a role in a soccer game. Once an agent has taken on a role, it has a number of *responsibilities* it is required to fulfill as a part of that role. These are persistent goals that detect undesirable aspects of the environment and trigger the agent to respond, e.g. a goal-keeper responds to an opponent's corner kicks because he is responsible for protecting his goal. A particular *strategy* is chosen to fulfill the responsibility depending on the role of the agent, and the conditions of the environment. A strategy can either involve a single role or several roles, called the strategy's *strategic group set*. The former are called *independent strategies* and latter are called *team strategies*. Team strategies are used to control the interaction of several agents, called a *strategic group*, by matching their roles with those in the strategy's strategic group set, e.g. a goal-keeper and several defenders form a strategic group set within a team strategy for defending the goal during an opponent's free-kick.

3 Execution Model

The concepts above are part of an execution model that defines the operational semantics of each concept, i.e. the effect of the concept in deciding the agent's behaviour. Figure 1 represents the flow of information from the external world into the agent and the computations carried out by the agent in its decision making process. The concepts of roles, responsibilities and strategies in the section above feature as the connecting link from some reasoning process that selects the new role, to the behaviour of an agent.

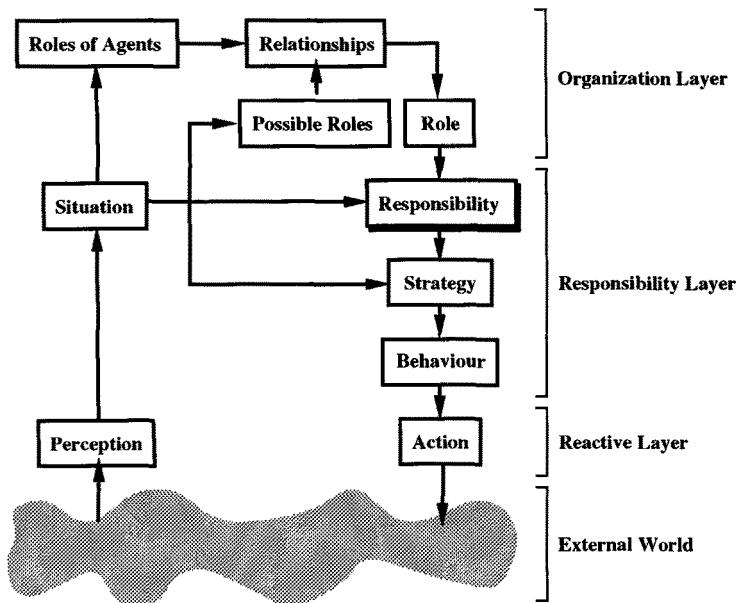


Fig. 1. The agent's execution model.

The figure shows the perception driven nature of our model. The perceptions are used in updating the beliefs of the agent regarding the state of the external world. The picture that the agent has of the world from these and other inferred beliefs describe a situation. Reasoning is performed on the situation to determine the behaviours of other agents and the belief of the role being played by that agent. This set of beliefs are used at several different layers: the organisational layer, the responsibility layer, and the reactive layer, to guide the choice of the agent's behaviour in response to the changing world.

3.1 Organisational Layer

The top layer of the execution hierarchy is called the organisational layer and handles the selection of an agent's role. The decision is based on the situation and other agents' roles. The situation constrains what roles can be adopted by the agent since a role requires special conditions before it can be adopted, e.g. the goal-keeper role is only adopted near the team's goal. The list of possible roles to be adopted is then passed to a selection process that reduces the many possible roles to just one. This process requires some static heuristic information about the relationship of roles to one another.

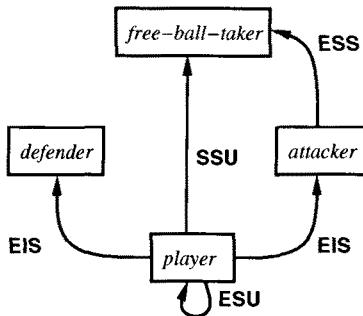


Fig. 2. The rolemap for roles in soccer.

Relation		Values		
Authority	Dominant	Equal	Subservient	
Cooperation	Oppose	Ignore	Support	
Derivation	Generalize	Unrelated	Specialize	

Fig. 3. The three values of a relationship.

A structure containing the relationship information is called a rolemap. The relationships are shown in figure 2 as the labelled vertices. Each letter of the label represents one relationship as shown in figure 3. The first letter represents authority, the second cooperation, and the third, derivation. For example, the link **EIS** indicates the relationship has the properties Equal, Ignore, and Specialize. The heuristic function selects a role that meets the following criteria, in the priority given:

1. the role that serves most dominant roles;

2. the role that supports the greatest number of roles;
3. the role that specialises the current role.

The first relationship is authority. Authority guides an agent's selection of a role towards one that serves the most dominant roles. Dominant roles indicate the focus of future interactions, e.g. a **free-kick-taker** role will be the centre of attention in a corner kick because he decides the strategy of other agents by his behaviour. An agent selects roles that will serve the **free-kick-taker**, e.g. a **player** role is subservient to a **free-kick-taker** because there are strategies for corner kicks that require **player** roles to participate. A role that dominates another role can be thought of as forcing other agents to adjust their role to be part of a team strategy with the dominant role.

The second relationship is cooperation. An agent selects a role that maximises its cooperation with team mates and its opposition to opponents. The support relationship can be thought of as measuring the interactivity of one role to another (by the number of team strategies that the role has that includes the other roles). For example, the **player** role is supported by the **player** role because the **player** has many team strategies that involve other **player** roles. Authority has precedence over cooperation.

The third relationship is derivation. A role that specialises another role has the responsibilities of the other role as a subset of its own responsibilities. We have chosen to build a tree structure from this derivation relationship, i.e. every role has one generalisation except the root which has no generalisation. The heuristic selects the most specialised role for the current role since this reduces the amount of change in the responsibility of the agent. Specialisation is the lowest priority in a rolemap as it deals purely with the single agent and not with other agents (this heuristic favours teamwork).

The set of possible roles are passed into the process of role selection that select one role. The process terminates when a single role remains or the final heuristic returns multiple roles from which one is chosen arbitrarily. The chosen role is then passed to the responsibility layer to be used in activating the role's responsibilities. During the course of the agent's life it may become many different roles depending on the situation, or it may temporarily be without a role when the role terminates and a new role is being selected. The latter will only happen if the responsibility fails in handling the situation and the process of selecting a new role has not completed.

3.2 Responsibility Layer

A set of responsibilities determines the appropriate stimulus from the environment that causes the agent to execute strategies. A responsibility is active while the agent's role supports that responsibility. A change in role results in some responsibilities being deactivated, others being activated, and the rest continuing to be active. A responsibility is either monitoring the environment or it is executing a strategy to fulfil its responsibility in the current situation. The failure of a responsibility to find a strategy for the present situation or the failure

of a strategy to fulfil the responsibility causes the responsibility to fail and the termination of the role.

A strategy, like that shown in figure 4, is executed by a responsibility for a particular situation, defined by its context. There are many strategies for a given responsibility, each placing restrictions on the context for which they are applicable and the roles that are expected to participate in the strategy. Once the appropriate strategy is selected a mapping is created for the agents of the strategic group to the parts of the strategy. Then a set of instructions are executed that provide each part (a member of the strategic group) the means for coordinating their actions. Each instruction is interpreted differently by the parts in the strategy, e.g. (**kick-ball P1 P2**) causes the part P1 to execute (**kick-ball P2**) and the part P2 to execute (**receive-ball**). The interpretation results in a behaviour that is used to guide the actions of the agent in the next level below, the reactive layer.

Strategy: Pass

Strategic group set:
free-kick-taker: FKT,
player: P1,
player2: P2.
Context: (corner-kick)

Plan:
(1) --> (move-to-ball FKT)
(2) --> (move-close P1 FKT)
(3) --> (move-clear P1 FKT)
(4) --> (move-clear-and-goal P2 P1)
(5) --> (kick-ball FKT P1)
(6) --> (kick-ball P1 P2)
(7) --> (shoot P2)

Fig. 4. Strategy for passing a ball between three roles: a **free-kick-taker** and two **players**.

3.3 Reactive Layer

A behaviour defines the type of action to be executed by the agent without considering the subtleties in controlling the uncertainty in the environment. At the reactive layer the behaviours are transformed into sequence of actions that

vary depending on variation in the environment, e.g. the erratic movement of a soccer ball. Using the principle that an agent is better off doing something than nothing, the reactive layer executes reflex actions under unexpected situations.

A reflex action is triggered whenever the conditions for a behaviour are no longer valid, e.g. kicking a ball when the player no longer possesses the ball. These conditions are detected by the reactive layer that causes a hard-coded reflex action to be executed instead of the behaviour. The reflex is based on the agent's role, the agent's precepts, and the behaviour that failed. A reflex attempts to select the best general response to the unexpected situation. For example, a **defender** that wants to pass the ball to a team mate but fail due to an obstructing opponent that is very close is would rather kick the ball away from goals than to wait for its next behaviour. On the other hand, if the opponent was at a non-threatening distance from the agent, the best general action would be to wait for the next behaviour.

3.4 Example: Corner Kick

The following example displays the practical workings of our model in the domain of a soccer simulation. Figure 5 illustrates the resulting movement of agent 2 from the white team, based on the roles, responsibilities, and strategies in table 1 and the rolemap in figure 2. The team strategy that describes the action is shown in figure 4.

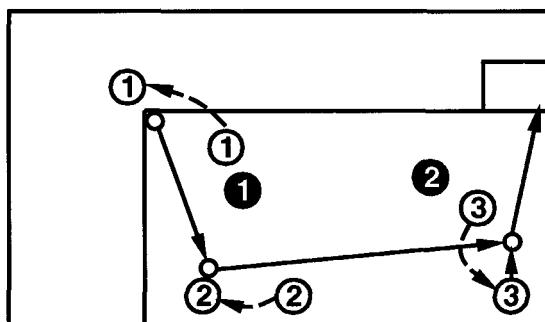


Fig. 5. An example of three agents executing a corner kick.

The trace begin with agent 2 playing an **attacker** role and receiving the message that a corner kick has been given to his team. Agent 1 and 3 are also in an **attacker** role.

1. Receives message that a corner kick given to our team.
2. Agent 1 has adopted the **free-kick-taker** role.

- (a) The agent's new role should be **player** as this role serves the **free-kick-taker** role.
 - (b) **player** responsibilities remain active: **position** and **move-ball**. They are triggered by a goal and a player that can kick the ball to the agent, respectively.
3. Agent 3 has adopted the **player** role.
 4. Agent 1 is close to the ball.
 - (a) The **move-ball** responsibility is triggered because someone can kick the ball to the agent.
 - (b) The pass strategy is selected with the parts mapped to different agents: agent 1 is **FKT**, agent 2 is **P1** and agent 3 is **P2**. Each instruction is then interpreted and executed:
 - i. Waits for agent 1 to move to the ball
 - ii. Executes (**move-close FKT**)
 - iii. Executes (**move-away-from-opponents**)
 - iv. Waits for agent 3 to move into clear
 - v. Executes (**receive-ball**)
 - vi. Execute (**kick-ball-to agent3**)
 - vii. Execute (**go-to-opponents-goal**)
 5. If a goal was scored trigger the **position** responsibility and move to a zone using the strategy **go-to-zone**.

4 Future Work and Conclusion

The main concepts discussed were roles, responsibilities and strategies. These provide the foundation of understanding what motivates an agent to form a team, where the agent's attention is focus, and the behaviours of the agent. The function of these concepts as part of an agent's execution model was discussed. The execution model split the hierarchy of information into three layers: the organisation, the responsibility, and the reactive layers. A trace of an agent's execution state was illustrated by the example of a corner kick scenario involving three soccer playing agents.

The project is currently implementing the team of soccer playing agents that will become the Royal Melbourne Knights. Details of the implementation are described in [Ch'ng and Padgham, 1997]. The next stage is to experiment with various roles, responsibilities, and strategies to create different teams of soccer agents. By competing teams against each other we aim to determine what combinations of strategies produce high performance teams. The best team will compete in the real-world soccer simulator tournament, using the RoboCup Soccer Simulator[Noda, 1995], at RoboCup'97[Kitano *et al.*, 1995].

5 Epilogue

The Royal Melbourne Knights competed at RoboCup'97 but only the reactive layer was complete. A fixed strategy was adopted based on the rules:

- The nearest player chased the ball, others would wait near a preassigned point.
- A player that could kick the ball would shoot the ball to the goal.
- A special goal keeper moved close to the goals and moved to a point relative to the goal that minimised the angle to the goal.

A large portion of development time was spent on improving the players skills, e.g. kicking the ball around the player and avoid other players when running. Players were fixed into a normal rate of perception (rather than a variable rate of perception based on switching viewing quality) and modelled the effects of its actions on the environment in order to act between percepts.

The results at RoboCup'97 based on our rudimentary implementation were below average. The team failed to score any goals and conceded 38 goals in its 4 games. Even though our team was able to move the ball around with a reasonable level of skill, its failure to pass to team members, its reliance on skills that worked under ideal conditions, and the inability to alter the speed of perception, contributed to its unsuccessful attempts to defend against attacks and score goals.

References

- [Ch'ng and Padgham, 1997] Simon H. Ch'ng and Lin Padgham. Role organization and planning team strategies. In *Proceedings of the 20th Australasian Computer Science Conference, Sydney, Australia*, pages 202 – 208, February 5-7 1997.
- [Kitano *et al.*, 1995] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. *Working Notes of IJCAI Workshop: Entertainment and AI/Alife, Montreal, Quebec, Canada*, pages 19 – 24, August 1995.
- [Noda, 1995] Itsuki Noda. Soccer server: a simulator of robocup. *Working Notes of IJCAI Workshop: Entertainment and AI/Alife, Montreal, Quebec, Canada*, August 1995.

A Multi-Layered Behavior Based System for Controlling RoboCup Agents

Paul Scerri

Department of Computer Science
RMIT University,
Melbourne, Australia.
pausc@ida.liu.se

Abstract. We describe a multi-layered behavior based agent architecture which has been applied to the RoboCup domain. Upper layers are used to control the activation and priority of behaviors in layers below, and only the lowest layer interacts directly with the server. The layered approach seems to simplify behavior management. In particular it provides an approach for implementing high level strategic behavior in the soccer agents.

1 Introduction

To be successful in a complex, dynamic domain such as RoboCup [4] an agent must be able to combine fast reactions with strategic decision making. A promising architecture aiming to combine these properties is the behavior based approach advocated by Brooks [2], Maes [5] and others. This type of system allows complex behavior to emerge from the interactions between simple behaviors and their environment.

There are many important claims made about behavior based systems, including their robustness, their adaptability, their ability to handle complex, dynamic domains and their extendibility. Little work has been done on using behavior based systems for complex tasks where strategic planning seems necessary.

In this extended abstract, we describe a system using a layered architecture based on the behavior based model.¹ This system has been applied to the RoboCup domain. Each layer of the architecture consists of an asynchronous, independent behavior based system. Layers are increasingly abstract further up the hierarchy: the lowest layer of the system interacts directly with the world while higher layers have their effect by dynamically setting the behaviors in the level below.² The architecture provides an approach to performing strategic planning within the behavior based paradigm.

¹ Blumberg [1] also proposes a multi-layered architecture.

² Firby [3] similarly describes the use of his RAP system for controlling low-level continuous processes by turning them on and off as is required to achieve specified high-level goals.

2 Single Layer Design

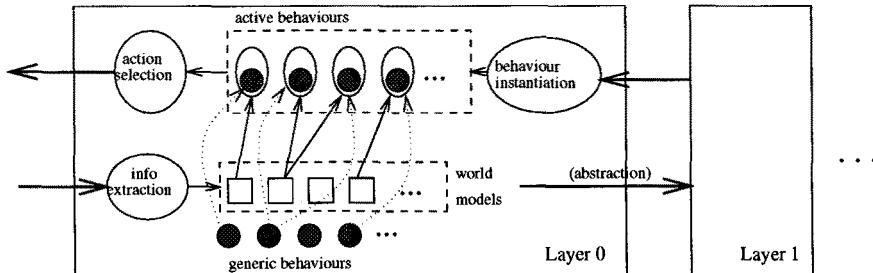


Fig. 1. Layer Model

2.1 Overview

The system is made up of a number of architecturally identical layers with each layer, as shown in Figure 1, consisting of: a number of generic behaviors, with their associated world information; a set of active behaviors; and three controlling processes. The control processes are *action selection*, *behavior instantiation* and *information extraction*. The set of active behaviors changes dynamically over the life of the player but the generic behaviors are fixed.

A behavior is a small, functional control unit that performs a particular action. A group of behaviors interact together and with the world to achieve a *strategy*, or complex style of play. In our design a behavior is created dynamically, via *instantiation*, from a generic behavior and a parameter component. Each behavior acts asynchronously and independently of other behaviors in a layer. Only the command from the controlling behavior, i.e. the behavior chosen by the action selection process, on each layer is executed by the system.

Action selection cycles through the current active behavior set to determine the 'best' behavior. Each behavior consults its associated current world information to determine its own applicability. The action selection process combines the *applicability* factor with the *persistence* and *priority* value of each behavior in order to determine the 'best' behavior. The priority is set by the layer above when a behavior is instantiated. The persistence value is calculated by the action selection mechanism as a means of reducing oscillations between behaviors. The action selection process then obtains a command from the selected behavior and executes that command, either by sending a message to the RoboCup server, in the case of the lowest layer, or by sending a command to the next layer down, in the case of upper layers.

Each generic behavior has associated with it some relevant information about its environment. There is no central, complete or symbolic description of the

world. The information extraction process, working asynchronously to the other processes, takes incoming percepts, extracts relevant information for each generic behavior's world information (performing some simple consistency checking to ensure the reliability of the stored data), then passes the information up to the next level. The information extraction process is transparent to the action selection and behavior instantiation processes.

The behavior instantiation process receives a command from the layer above indicating a new set of behaviors, with associated priorities and parameters, that should be made active. Once the action selection process has completed choosing an action the behavior instantiation process destroys the current active behavior set then combines generic behaviors with parameters to create a new active behavior set. The active behaviors in the topmost layer are constant for the life of the agent.

2.2 Generic and Parameterised Behaviors

A behavior in the architecture is made up of two parts: a generic behavioral part, which implements the functionality of the behavior, and a parameter component. The generic part of a behavior is created once and exists for the life of the agent but parameters and the resulting overall behavior are created dynamically as required. We refer to the process of creating a behavior from a generic behavior and a parameter as *instantiation*.

A generic behavior implements two functions - a *control* function, for issuing a command, and an *applicability* function through which the behavior indicates its applicability in light of the current perceived state of the world. A generic behavior is designed to control a very specific action without regard to any other actions the agent may be involved with. The narrow focus of a behavior allows a designer to develop small, fast implementations.

Each generic behavior has associated with it some simple information about the world which has been extracted from the incoming percepts. Each generic behavior maintains only the specific information it requires - no complete or symbolic representation of the world is kept. Information for higher level behaviors is at a more abstract level than for low level behaviors. For example *defend*, an upper level behavior, may record that the ball is in the back half of the ground while *move-to-ball*, a lower level behavior, may record that the ball is 45 degrees to the left.

The use of parameters means that the same generic behavioral code can be used many times with varying effects. One generic behavior may be part of several behaviors simultaneously, each using different parameters, as is shown in the figure.

A behavior issues commands and calculates its own applicability without regard to the applicability or priority of other behaviors in the system. It is the designer's responsibility to ensure that the resulting interactions between behaviors are such that the required overall behavior emerges.

Implemented low level generic behaviors include *pass* and *move-to*. At the next highest layer are behaviors such as *chaser* and *forward*. At an even higher level of abstraction are *kick-off* and *normal-play*.

2.3 Choosing the Best Behavior

The behavior chosen to be the controlling behavior at any particular level, i.e. the behavior whose command will be executed, is the behavior for which the sum of *applicability*, *priority* and *persistence* is the highest. By manipulating these three factors the designer can set up situations, with a relatively small number of simple behaviors, where a wide range of different complex behaviors will emerge.

A fundamental idea behind a behavior based system is that behaviors will be more or less active depending on the state of the world. In our architecture this 'activation' or 'excitation' is represented by the applicability value. The behavior consults its relevant world information to determine its own applicability factor. The applicability of a behavior is simply a numerical value and will vary as the world changes. For example a *move-to-ball* behavior may have a high applicability when the ball is in view but a low applicability when the ball's position is unknown.

Priorities are the mechanism the designer uses to control the overall behavior of the system. A priority is a property of an instantiated behavior that is constant for the life of that behavior. A player can be made to prefer attacking down the left by increasing the priority of that behavior at design time, but it may still attack down the right if this happens to be a significantly better option in a specific situation. For example, when chosen for execution the upper level behavior *normal-play* could instantiate the behavior *attack-down-left* with a priority value of 6 and the behavior *attack-down-right* with a priority value of 4. The values used for the priorities are chosen at design time. In this example the resulting behavior is to prefer attacking down the left whenever the *normal-play* behavior is active.

Persistence implements the architecture's commitment factor, by reinforcing the previously chosen behavior. Persistence is generally a comparatively small factor which only comes into play when the *applicability* plus *priority* of two or more behaviors is similar. In this case the persistence factor helps to stabilize the system and reduce oscillations between behaviors.

For example, if the system is finely balanced in its choice between two behaviors it may oscillate between the two - e.g. when there are similar numbers of players at each end of the ground a player may oscillate between *attack* and *defend*. The use of persistence helps reduce such oscillations³.

Many behavior based systems use some form of suppression or inhibition of behaviors by other behaviors as another method of control. We do not use inhibition in the current implementation because we have not found it necessary.

³ However problems processing incoming information means that rapid, unwanted switching between behaviors still occurs.

2.4 Commands

High level behaviors issue messages to the next lower layer describing the low level behaviors required to achieve the high level behaviors strategy. For example a high level *attack-down-wing* behavior might request low level behaviors *move-to-the-ball*, *move-to-the-wing*, *pass-to-wing* and *kick-goal* be set, with appropriate priorities, in order to achieve its goal.

Behaviors at the lowest level of the system act directly on the environment. In the RoboCup domain a low-level command consists of sending a ASCII string down a socket to the RoboCup server but in other domains the command may alter the power to an electrical motor or draw to a screen.

3 Multi Layer Design

Our layered model effectively modularises the behavior based architecture and provides a behavior management system without compromising the conceptual framework. The layered model seems to simplify the task of managing the interactions between behaviors because a designer only ever has to handle a small number of behaviors at once.

The idea is that each layer is more abstract than the layer below and acts asynchronously and independently with respect to other layers. Behaviors in higher layers achieve higher level strategies by specifying the lower level behaviors required to implement that strategy. All layers are architecturally identical except in the form of their input and output. A consequence of having asynchronous layers is that layers can be run at different priorities, i.e. lower layers can have high priority in order to ensure quick reactions while higher layers can have lower priority as the most appropriate strategy will not change rapidly.

3.1 Abstraction

The layers provide the designer with an abstraction mechanism to make the design of complex behaviors easier. Abstraction in this architecture is in the level of the behavior description. Behaviors in low levels are simple atomic actions such as *kick* while as we move up the layers more abstract behaviors, such as *attack-down-the-wing*, are specified. This type of abstraction is intuitive and allows a natural design process: a simple agent can be developed and higher level strategies built on top later. This type of development would appear to be especially useful in complex domains because the strategies required may not be obvious until late in development.

3.2 Behavior Management

At first inspection it may seem as though this system differs conceptually from the standard behavior based ideas because only a small number of behaviors are active at once, however the layering system can be thought of as imposing

a dynamic priority system on all possible behaviors. Behaviors that are not currently active have been effectively inhibited, while active ones are at varying levels of excitation.

Minimizing the number of behaviors in a behavior based system simplifies the development of a priority system between the behaviors that implement the required complex behavior. However as the number of behaviors increases, prioritizing behaviors becomes more complex. Our layered model alleviates this problem because only a few behaviors are active at any one time. The problem of creating a priority system among a large number of behaviors is reduced to creating many priority systems each between only a few behaviors. However, even with a reduced number of behaviors, and hence interactions, finding appropriate combinations of behaviors is not a trivial task.

4 Results

4.1 Implementation

The system was initially implemented in C++ under Solaris. Threads were used to create asynchronous layers. The processing of input perceptions was also handled by a separate thread. For RoboCup '97 the system was ported to Java.

Object orientated techniques were used to allow for significant code reuse. Inheritance, abstract base classes and function overloading techniques were used to create hierarchies of generic behaviors. Action selection and behavior instantiation processes at each layer are instances of the same class. Behaviors were implemented as a pointer to a generic behavior class, a pointer to a base parameter class and a priority value.

4.2 Observations

A team of eleven homogenous players competed at RoboCup '97 with a record of one win and two losses. Agents in the team had three layers of behaviors. At any one time there was up to five behaviors active at any layer.

Some key features of the team were :

- Players in the team were usually able to respond to situations even when behaviors had not been designed to handle the specific situation. However there were some situations where the emergent behavior was not desirable, most notably the taking of a free-kick from the sideline which usually resulted in the player kicking the ball straight back out⁴.
- The players are fast enough to decide on an action every time an action could be sent to the server.
- Slightly different combinations of behaviors provided different levels of team structure, varying from chaotic to reasonable organization.

⁴ This emergent behavior has been attributed to the low-level *kick* behavior not being sufficiently intelligent to move around the ball before kicking, hence the result that often the ball would be kicked back out.

- Although the behavior-based approach ideally does not maintain a model of the world we found it necessary to maintain world information, and reason about this information, in order to successfully implement some behaviors. Creating and maintaining a reasonable model of the world caused many, unresolved, conceptual and technical problems⁵ which severely reduced the effectiveness of some behaviors. Behaviors relying on abstracted information, such as *spread-out*, were especially affected.

4.3 The Development Process

The design and development of an agent was surprisingly straightforward once the skeleton of the architecture had been implemented.

Of particular interest with regard to the design process are the following :

- As we hoped when designing the architecture, the system was able to be built up piece by piece without having to make substantial changes to other parts of the system.
- The behavioral abstraction proved to be a very useful tool for designing the system.
- As there were never more than five behaviors active on a level at any time, finding appropriate priority values that resulted in the required complex behavior could usually be done with a reasonably small amount of trial and error experimentation. However, finding appropriate *applicability* functions and effective combinations of behaviors proved to be quite difficult.
- We could make abstract changes to the system very simply. For example, making the team more attacking simply required increasing the priority of the high level *attack* behavior.

4.4 Suggested Enhancements

There are many areas where we think it would be interesting to do further work. These include:

- More experimentation with the priority values, including how they are set, how they are used, (i.e. how they are combined with applicability and persistence) and what appropriate values are.
- A real problem with the players was oscillations between behaviors when information indicated that more than one behavior was applicable. This may be improved by better information processing or improvement of the persistence mechanism.
- Although players attempted to pass to one another as part of some larger strategy, these passes were rarely successful. The problem seemed to be with handling uncertain information, coming at a slow percept rate, about a fast moving object. To improve player's skills would require a significant improvement in the processing of incoming information.

⁵ Some of which have been addressed in [8]

- The current action selection mechanism selects one behavior and ignores all other behaviors, even though there may be other behaviors that are reasonably active. Improved overall behavior may be achieved by allowing behaviors that are active, but not selected, to influence the resulting overall behavior in some way in order to create *compromise* solutions.

Acknowledgements

The author would like to thank Lawrence Cavedon for all his assistance with this paper. Also thanks to Lin Padgham, Simon Ch'ng and Nick Howden for many very interesting discussions. We would also like to acknowledge the support of the Australian AI Institute for the RoboCup project at RMIT.

References

1. Bruce Blumberg and Tinsley Galyean. Multi-level control of autonomous animated creatures for real-time virtual environments. *Siggraph '95 Proceedings*, 1995.
2. R. Brooks. Intelligence without reason. In *Int'l Joint Conf. on AI*, Sydney, 1991.
3. James R Firby. Task networks for controlling continuous processes. *Proceedings of the Second International Conference on AI Planning Systems*, Jun 1994.
4. H. Kitano, M. Asada, Y Kuniyoshi, I. Noda, and E. Osawa. Robocup: the robot world cup initiative. In *IJCAI-95 Workshop on Entertainment and AI/ALife*, Montreal, 1995.
5. P. Maes. Modeling adaptive autonomous agents. *Artificial Life Journal*, 1, 1994.
6. Maja J Mataric. *Interaction and Intelligent Behavior*. PhD thesis, MIT EECS, Aug 1994.
7. Craig W. Reynolds. Flocks, herds, and schools: A distributed behavior model. *Computer Graphics (proceedings of SIGGRAPH'87)*, Jul 1987.
8. Jamie Westendorp. Behavior-based reasoning with dynamic, partial, uncertain information. Honours thesis, RMIT University, October 1997.

Using ABC^2 in the RoboCup Domain

Vicente Matellán, Daniel Borrajo, and Camino Fernández

e-mail: {vmo,dborrajo,camino}@ia.uc3m.es
Departamento de Informática
Universidad Carlos III de Madrid
28911, Leganés, España

Abstract. This paper presents an architecture for the control of autonomous agents that allows explicit cooperation among them. The structure of the software agents controlling the robots is based on a general purpose multi-agent architecture based on a two level approach. One level is composed of reactive skills capable of achieving simple actions by their own. The other is based on an agenda used as an opportunistic planning mechanism to compound, activate and coordinate the basic skills. This agenda handles actions both from the internal goals of the robot or from other robots. This paper describes the work already accomplished, as well as the issues arising from the implementation of the architecture and its use in the RoboCup domain.

1 Introduction

The aim of this paper is to present the multi-agent architecture we are developing at University Carlos III.. This architecture (named ABC^2) is based on pre-defined *skills* that each robot composes in an opportunistic way to achieve an intelligent behavior. The way these basic actions can be combined to get more sophisticated behaviors is also pre-defined. This means that we are not using classical search-based planning to combine the actions. We use instead an agenda to keep a list of pending actions, where each action can require (or not) pre-defined simpler actions.

Actions can be inserted into the agenda by other actions, by events from the environment or by requests received from other robots. Similarly, actions can be accomplished as a result of the execution of other actions, by another robot actions or simply by changes in the world. Let us think, for instance, in an action of the RoboCup [4] domain, like *get-the-ball*. The robot can get the ball, either by its own actions (movements), asking another robot to pass the ball, or by any other event in the world (the opponent accidentally kicked the ball towards the robot).

For the definition of the skills, different types of controllers can be used, such as fuzzy controllers, mathematical calculus, or learned behaviors. For instance, we will use our previously developed software for building fuzzy behaviors. The cooperative part of this architecture is theoretically based on the Speech Acts theories [3].

In this area, our contribution will be in two aspects. First, we will extend these ideas to cope both with a highly dynamic environment (robotic soccer) and with a *real world* environment (in the sense of a sensors and actuators approach). Second, we will apply fuzzy logic both to define basic controllers and to write the heuristics that will control the robots.

We are not worried, by now, about any kind of learning, neither we do not try to figure out what the other team is trying to do, or doing. We also think that any kind of search-based planning is unuseful in such a highly dynamic environment.

In the next section the architecture is presented in more depth, discussing the skills that will be used. Section 3 describes the execution of the whole system, specially the role of the agenda in the control of the robot actions. The last section describes the current state of system development, and future work.

2 Description of the architecture

We have just introduced the goals of the architecture: allow explicit cooperation among the team mates, use opportunistic planning to combine robot actions, and use pre-defined basic reactive skills.

This section presents the general architecture we are developing, which is shown in Figure 1. This architecture is made up of different parts that will be explained in the following subsections.

2.1 Skills

The box labelled as *Skills* represents a set of simple and reactive controllers. These controllers implement pre-defined behaviors that the robot can accomplish. In Figure 1 some of these behaviors are shown.

The popularity reached by Rodney Brooks' work on the subsumption architecture [2] increased the interest in systems based on the composition of reactive *behaviors*. However, the idea of achieving intelligent behavior in robots using a bottom-up approach was not new. There had been many other work in the literature using the same approach, such as V. Braintenberg work [1]; neural networks based behaviors; or the most classical mathematical controllers. We have used fuzzy controllers to implement our behaviors. The main reason for this election was our previous experience using this controllers. We already had general fuzzy reasoning libraries so that we can easily design new controllers.

The design of the behaviors has been done heuristically. This means that we have chosen the rules by hand. However, many "automatic" methods for designing this type of rules can be found in the literature, ranging from the mathematical methods to neural networks or genetic algorithms. For instance, we have obtained good results using the last method [5]. However, most of these methods have been designed to learn in well-defined environments, with few dynamic objects, and they are highly time consuming. Besides, most of them do not bother about the multi-agent aspects of the soccer environment, though

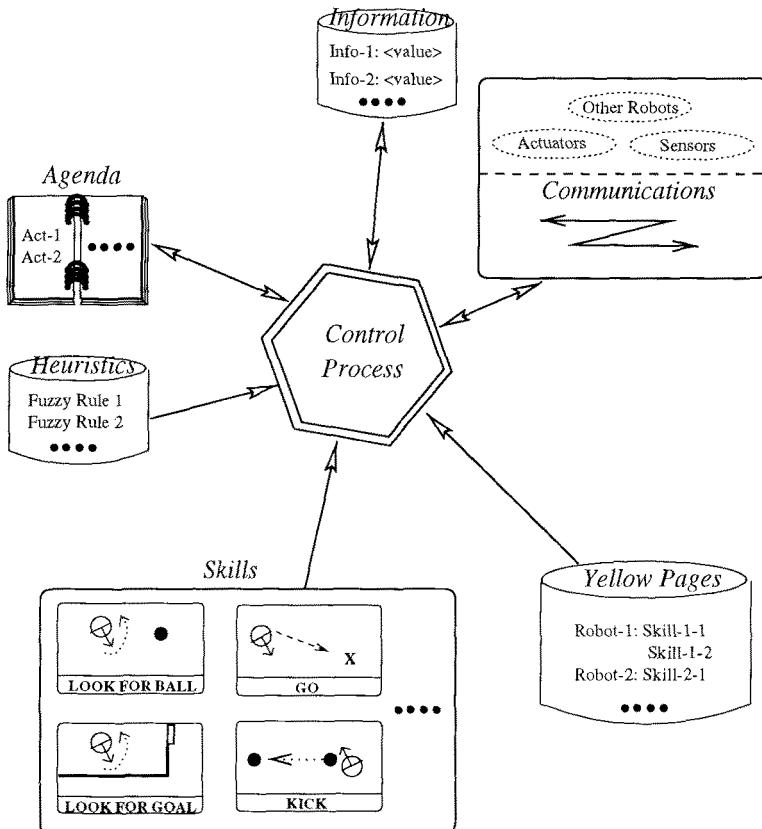


Fig. 1. Architecture of the Robots.

there are some new interesting work in this area [8]. We have decided to used hand-made fuzzy behaviors, because we can easily design them using high level rules, they perform well in the presence of uncertainty, and they can cope with multi-agent problems [6].

2.2 Yellow Pages

The repository named *Yellow Pages* in Figure 1 represents the information that an agent has about the other agents that form its team. This information basically consists of a table made by the name of its team mates, and the name of the skills they can accomplish. These skills will be used in the same way as its own skills.

A skill can be considered as an abstraction of an action that will be accessible to other team-mates. In fact, this means that we are considering that the robot

has meta-knowledge about itself (through its skills definition) and its team-mates (using the yellow pages).

2.3 Information

Classical reactive behaviors compute the outputs for the actuators of an agent directly from the raw numerical data perceived by its sensors. In other environments, like the RoboCup simulator, the inputs are not numerical data obtained from the sensors, but a mixture of linguistic and numerical information. In order to be able to handle this information we will use a reduced language that allows the agent to define the inputs of the skills and to keep significant information about the current state of the world. So, the skills previously defined use this language to represent the information used in the robot inputs. As an example, each agent keeps information about the distance and orientation to the goal.

2.4 Communications

One of the distinctive capabilities of agents is their ability to communicate with other agents. In order to be able to manage the intrinsic complexity of the communication (protocols, queues, etc.) we provide our agents with a specialized entity to cope with it.

Besides, in the RoboCup simulator [7], communication happens among two different agents, but also between an agent and its sensors and actuators. And both types of communication use the same channel in this case (a socket between the agent and the simulator). So, the entity in charge of the communication with the RoboCup simulator will have to be able to distinguish the different types of messages.

2.5 The Agenda

The *Agenda* is a dynamic structure that contains items named *acts*. These acts represent the potential actions that the robot is considering at a moment. We have considered four kinds of acts:

- REQUESTED, to indicate that the action in the argument of the act has been requested by another robot in order to be performed by this one.
- REQUEST, to ask another agent a particular action.
- INFORMED, presenting a piece of information sent by other robot.
- SUPPLY_INFO, to point out that some information has to be sent to another robot.
- DO, that represent potential skills that the robot can perform by itself. In the next section, the fundamentals of these behaviors are presented.

2.6 Heuristics

Heuristics decide at any time what act to select from the agenda. We have used fuzzy rules for the current implementation. The input variables of these rules can be, for instance:

- The priority of the skill associated to an act.
- The time that an act has been in the agenda.
- Number of acts that require an act to be evaluated.
- Information from the environment.
- The type of agent (defender, forward, etc.).

The output will be the weight of each act in the agenda. Once the acts have been weighted, the eligible act to be executed will be the one with the highest weight.

2.7 Agent Model

In summary, from the point of view of our model, the robot can be considered as a knowledge structure defined as a set of statical and dynamical attributes. Among the static ones we can quote the name of the robot (N), the list of its skills (S), the knowledge about its team-mates names and skills, (Y), the language to represent the information about the current state of the world (L), and the set of heuristic rules that governs the behavior of the agent (H). So, an agent (A) can be represented as the tuple: $A = \langle N, S, Y, L, H \rangle$. In the same way, the team of agents can be represented as $\langle N, S, Y, L, H \rangle^+$, given that a team is made up of at least one agent.

Among the dynamic information that defines the current situation of an agent, we can cite the agenda (A_g) that contains the acts currently under consideration, the queues of messages (Q) received or pending to be sent and the information (I) about the current state of the world, defined using the language L . So, an agent in a given moment is defined by $\langle A, A_g, I, Q \rangle$, and the situation of the whole team as $\langle A, A_g, I, Q \rangle^+$.

3 System Execution

The definition of a particular skill (top right box on figure 2) consists first on the design and implementation of the controller that performs the desired action (this is represented as the function *Execute* in the figure). Then, a condition for triggering the controller (named *Ready* in the figure) is established in order to know if the controller can be executed. In the case of the skill being evaluated but not being able to execute its associated controller (the *Ready* function returns a FALSE value), the skill provides a list of skills that can make it “executable”. This list has been named *Needs* in the figure. The remaining slot of the box is the *Priority* assigned to the behavior. This value can be used in the heuristic rules to select acts from the agenda.

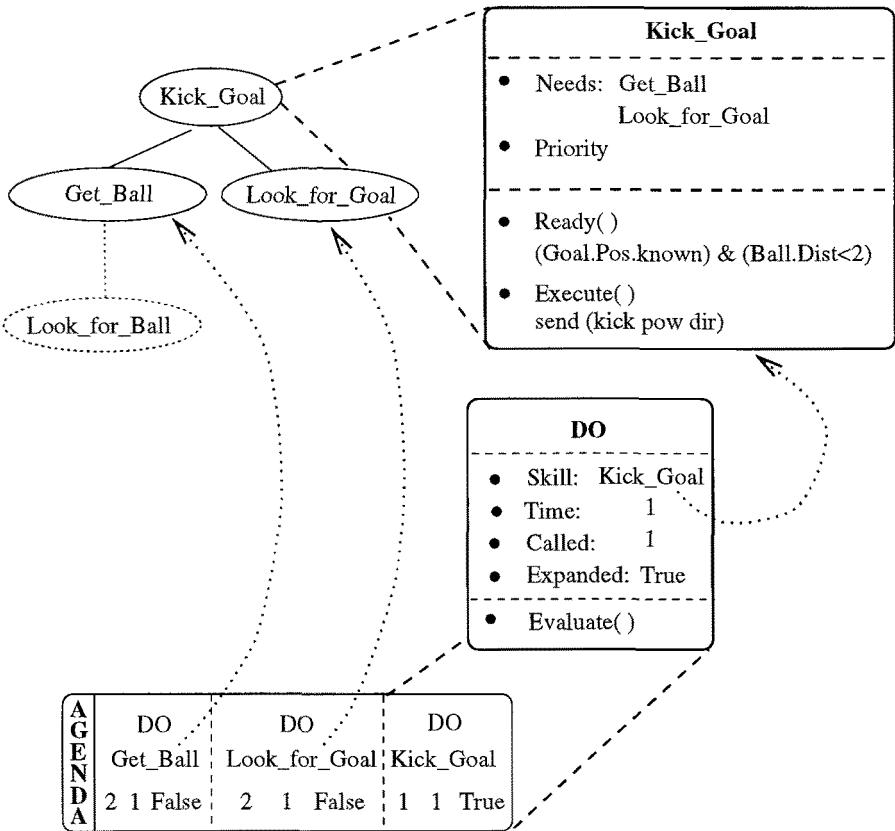


Fig. 2. Relations among the skills and the agenda.

Once the skills of the robot have been designed, (in this example we will consider only the four skills that appear in the top-left tree of figure 2), the heuristics have to be defined. Let us suppose that we settle up a simple heuristic: “Select from the agenda the act whose *Priority* value is the highest from the ones that are *Ready*”. Let us also suppose that the information that the robot has about the world is the raw data that it gets from the RoboCup simulator and that it does not concern other team-mates skills. So, in this environment, we are considering only acts of type DO.

In such a simplified environment, the robot is only able to look for the ball and to kick it towards the opponent goal, according to the Kick_Goal skill. In order to do that, the robot has to be “stimulated” to do it. This means that the robot has to be initialized to pursue the goal Kick_Goal. This initialization is performed by inserting the act [DO: Kick_Goal] into the agenda.

The components of an act (as shown in Figure 2) are: the type of the act (DO, REQUESTED, etc.); the name of the associated skill; the counters Called,

that indicate the number of acts that require it, and Time that keeps the time when the act was inserted into the agenda; the switch Expanded, that indicates if the needs of the associated skill have been added to the agenda or not; and the function Evaluate, that indicates what has to be done when the act is selected (for example, execute its associated skill if the type of the act is DO).

The way the control cycle works is as follows: first, the applicable acts are selected. This is achieved by consulting the *Ready* function of the skill associated to each DO act. If the act is not applicable, then the Expanded switch is checked. If it has not been expanded its needs are inserted into the agenda like [DO: <need>] acts. This addition checks if that act had been previously added to the agenda by other acts. If the act already was in the agenda, the counter Called of the act is increased. Otherwise, a new act is added to the agenda. On the other hand, if the act is applicable and had been expanded, the counter Called is decreased. At the same time that the applicable acts are selected, the acts whose Called counter is equal to zero (no other act requires them) are removed from the agenda. Once the applicable acts have been selected, the domain heuristics are applied to select the one that will be evaluated.

The state of the agenda in Figure 2 shows that the act [DO: Kick_Goal] was inserted in the agenda at time 1 and it has been expanded. As a result of its expansion, the acts [DO: Look_for_Goal] and [DO: Get_Ball] were inserted at time 2. These acts have not been expanded, and are called by the act [DO: Kick_Goal]. This means that the agenda shows both the current state of the agent goals and part of the history of its activity.

The treatment of the other types of acts will be similar. Only the evaluation of these acts will be different. For instance, if an act [REQUESTED: Look_for_Goal] is evaluated, it can result in the evaluation of the skill Look_for_Goal and the insertion of its result as a [SUPPLY_INFO: <result>] act into the agenda.

4 Conclusion and Further Works

At Carlos III University we had been using preliminary versions of *ABC*² on our Khepera mini-robots [6]. As we had experience on designing reactive low level behaviors, we expected the main problem to be the design of the complex behaviors. So we began to work with the RoboCup simulator and the software we had previously built, which implements fuzzy controllers and the main part of the agenda-based control architecture, and we did not find many problems integrating both softwares.

The architecture was tested in the RoboCup'97 simulation track. The team lost its first match against CMUnited (9-1). It beat RMKnights (10-0) and lost against one of the teams of Kinki University.

The first conclusion we can get from these results is that if a team is better than another one in some particular tasks, then the results are really large. This is due to the fact that the competition is held on a simulator. So, we should try to focus on the issues that made a team better and not in the numerical results.

The first one was really a well tuned team. They had been working on this specific domain for a long time [8], and they had well-performing players and a nice global strategy. In summary, they actually have a good team. The only objection from our point of view is that they used a very specific approach. The second one, RMKnights, was also testing a general architecture in this domain. Its main drawback was that its players were too slow. We have less information about the third one, but we consider that the main reason for its victory was their control of the stamina parameter. This made them able to move faster in some periods of the match.

Now we are mainly working in two aspects of our architecture: Refining the basic behaviors and improving the cooperation mechanisms in order to use different attack strategies. In the first aspect we are studying, for instance, how to improve the basic skills using some kind of mathematical prediction about the robot moments (to know its position at any moment), the ball moments (to predict its position), etc.

References

1. Valentino Braatenberg. *Vehicles. Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA (USA), 1984.
2. Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), 1986.
3. Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, RA-2(3):177–212, 1986.
4. Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Life*, pages 19–24, 1995.
5. Vicente Matellán, José Manuel Molina, and Camino Fernández. Learning fuzzy behaviors for autonomous robots. In *Fourth European Workshop on Learning Robots*, Karlsruhe, Germany, December 1995.
6. Vicente Matellán, José Manuel Molina, and Lorenzo Sommaruga. Fuzzy cooperation of autonomous robots. In *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, Lisboa, Portugal, July 1996.
7. Itsuki Noda. Soccer server: A simulator of robocup. In *Proceedings of AI Symposium'95*. Japanese Society for Artificial Intelligence, December 1995.
8. Peter Stone and Manuela Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. Technical report, School of Computer Science. CMU-CS-95-207. Carnegie Mellon University, 1995.

Integrating Learning with Motor Schema-Based Control for a Robot Soccer Team

Tucker Balch

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-208 USA

Abstract. This paper describes a reinforcement learning-based strategy developed for Robocup simulator league competition. In overview: each agent is provided a common set of skills (motor schema-based behavioral assemblages) from which it builds a task-achieving strategy using reinforcement learning. The agents learn individually to activate particular behavioral assemblages given their current situation and a reward signal. The entire team is jointly rewarded or penalized when they score or are scored against (global reinforcement). This approach provides for diversity in individual behavior.

1 Introduction

Motor schemas are an important example of behavior-based robot control. The motor schema paradigm is the central method in use at the Georgia Tech Mobile Robot Laboratory, and is the platform for this research. Motor schemas are the reactive component of Arkin's Autonomous Robot Architecture (AuRA) [1]. AuRA's design integrates deliberative planning at a top level with behavior-based motor control at the bottom. The lower levels, concerned with executing the reactive behaviors are incorporated in this research.

Individual motor schemas, or primitive behaviors, express separate goals or constraints for a task. As an example, important schemas for a navigational task would include **avoid_obstacles** and **move_to_goal**. Since schemas are independent, they can run concurrently, providing parallelism and speed. Sensor input is processed by perceptual schemas embedded in the motor behaviors. Perceptual processing is minimal and provides just the information pertinent to the motor schema. For instance, a **find_obstacles** perceptual schema which provides a list of sensed obstacles is embedded in the **avoid_obstacles** motor schema. Motor schemas may be grouped to form more complex, emergent behaviors. Groups of behaviors are referred to as *behavioral assemblages*. One way behavioral assemblages may be used in solving complex tasks is to develop an assemblage for each sub-task and to execute the assemblages in an appropriate sequence. The resulting task-solving strategy can be represented as a Finite State Automaton (FSA). The technique is referred to as *temporal sequencing* [1].

Even though behavior-based approaches, like the motor schema paradigm are robust for many tasks and environments, they are not necessarily adaptive.

When feedback regarding success in a task is available, *reinforcement learning* can shift the burden of behavior refinement from the designer to the robots operating autonomously in their environment. For some simple tasks, given a sufficiently long trial, agents are even able to develop optimal policies [5]. Rather than attempting to design an optimal system from the start, the designer imbues his robots with adaptability. The robots strive continuously to improve their performance; finding suitable behaviors automatically as they interact with the environment. For these reasons reinforcement learning is becoming pervasive in mobile robot research.

Q-learning is a type of reinforcement-learning in which the value of taking each possible action in each situation is represented as a utility function, $Q(s, a)$. Where s is the state or situation and a is a possible action. If the function is properly computed, an agent can act optimally simply by looking up the best-valued action for any situation. The problem is to find the $Q(s, a)$ that provides an optimal policy. Watkins provides an algorithm for determining $Q(s, a)$ that converges to optimal [8].

The Java-based soccer control system described here (Figure 1) was originally developed for research using a simulator developed at Georgia Tech [4]. The system is being converted for operation with the Soccer Server simulation system developed by Noda [6].

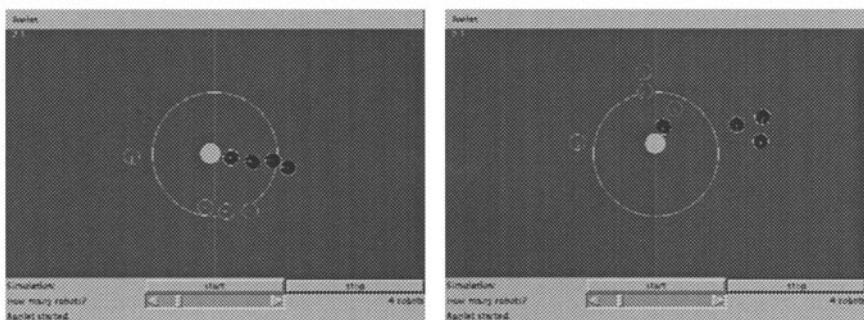


Fig. 1. Examples of homo- and heterogeneous learning soccer teams. In both cases the learning team (dark) defends the goal on the right. The agents try to propel the ball across the opponent's goal by bumping it. A homogeneous team (left image) has converged to four identical behaviors which in this case cause them to group together as they move towards the ball. A heterogeneous team (right) has settled on diverse policies which spread them apart into the forward middle and back of the field.

How can we objectively evaluate a robot soccer team? In a human game the object is to have scored the most points when time runs out. It is only necessary to score one more point than the other team. Here, we take the stance that greater score differentials indicate better performance (it is best to humiliate

the opponent!). Hence, the performance metric for robot teams is

$$P = S_{\text{us}} - S_{\text{them}} \quad (1)$$

where S_{us} and S_{them} are the scores of each team at the end of the game.

The simulation proceeds in discrete steps. In each step the robots process their sensor data, then issue appropriate actuator commands. The simulation models physical interactions (robot, ball and wall collisions), sensors and motor-driven actuators. When the ball is bumped by a robot it immediately accelerates and rolls away. Rolling friction is modeled with constant deceleration after the bump. Each agent is provided the following synthetic sensors:

- **Velocity sensor:** provides present heading and speed of the robot.
- **Bump sensor:** returns a force vector in the direction of any bump.
- **Ball position sensor:** provides an egocentric vector to the soccer ball.
- **Defended goal sensor:** provides an egocentric vector back to the robot's own goal.
- **Team sensor:** returns an array of egocentric vectors pointing to the robot's team members.
- **Enemy sensor:** an array of egocentric vectors pointing to the robot's opponents.
- **Score sensor:** indicates whether the team has just scored or was scored against.
- **Robot ID:** a unique integer from 1 to the size of the team.

The ball position, robot ID and defended goal sensors are used in the experimental robots examined here. At present, the sensors are perfect. Future revisions of the simulator may address real-world issues like noise, sensor occlusion and field-of-view constraints. The following actuator interface is provided to the control system:

- **Set drive speed:** a real value from -1 to 1 is sent to the robot's drive motor, indicating how fast the robot should go.
- **Set heading:** a real value from 0 to 2π is sent to the robot's steering actuator indicating the desired heading for the robot.

The sensor and actuator interface closely parallels those available on commercial robots. An eventual goal is to verify this work by porting the system to four Nomadic Technologies Nomad 150 robots in Georgia Tech's Mobile Robot Laboratory.

2 Behaviors for Soccer

Behavior-based approaches are well suited for robot soccer since they excel in dynamic and uncertain environments. The robot behaviors described here are implemented in Clay, an object-oriented recursive system for configuring robot behavior. Clay integrates primitive behaviors (motor schemas) using cooperative and competitive coordination operators. Both static and learning operators are available. The system is outlined at a high level here. For more detail the reader is referred to [2].

perceptual feature	assemblage
	<i>mtb gbb mtbf f</i>
<i>not behind.ball</i>	0 1 0
<i>behind.ball</i>	1 0 0

Control Team Forward

perceptual feature	assemblage
	<i>mtb gbb mtbf f</i>
<i>not behind.ball</i>	0 1 0
<i>behind.ball</i>	0 0 1

Control Team Goalie

Fig. 2. The control team's strategy viewed as look-up tables. The 1 in each row indicates the behavioral assemblage selected by the robot for the perceived situation indicated on the left. The abbreviations for the assemblages are introduced in the text.

Experiments are conducted by engaging an *experimental* team against a fixed opponent *control* team in soccer contests. We begin by describing the control team's behavioral configuration. Since the experimental team's performance will be significantly impacted by the skill of its opponent, it is important to avoid variability in the control team's strategy to ensure consistent results. The control team will always follow a fixed policy against the teams under evaluation.

The control team's design is based on two observations. First, points are scored by bumping the ball across the opponent's goal. Second, robots must avoid bumping the ball in the wrong direction, lest they score against their own team. A reasonable approach then, is for the robot to first ensure it is behind the ball, then move towards it to bump it towards the opponent's goal. Alternately, a defensive robot may opt to remain in the backfield to block an opponent's scoring attempt.

To implement this design, each robot is provided a set of behavioral assemblages for soccer. Each assemblage can be viewed as a distinct "skill" which, when sequenced with other assemblages forms a complete strategy. This style of behavior-based robot design, referred to as *temporal sequencing*, views an agent's strategy as a Finite State Automaton. The strategies may be equivalently viewed as lookup tables (Figure 2). This paper will focus on the lookup table representation since it is also useful for discussing learned policies. The behavioral assemblages developed for these experiments are:

- *move_to_ball (mtb)*: The robot moves directly to the ball. A collision with the ball will propel it away from the robot.
- *get_behind_ball (gbb)*: The robot moves to a position between the ball and the defended goal while dodging the ball.
- *move_to_back_field (mtbf)*: The robot moves to the back third of the field while being simultaneously attracted to the ball.

The overall system is completed by sequencing the assemblages with a selector which activates an appropriate skill depending on the robot's situation. This is accomplished by combining a boolean perceptual feature, *behind_ball*

(*bb*) with a selection operator. The selector picks one of the three assemblages for activation, depending on the current value of *bb*.

The team includes three “forwards” and one “goalie.” The forwards and goalie are distinguished by the assemblage they activate when they find themselves behind the ball: the forwards move to the ball (*mtb*) while the goalie remains in the backfield (*mtbf*). Both types of player will try to get behind the ball (*gbb*) when they find themselves in front of it.

3 Learning Soccer

To isolate the impact of learning on performance, the learning teams were developed using the same behavioral assemblages and perceptual features as the control team, thus: **the relative performance of a learning team versus the control team is due only to learning.**

Recall that Clay (the system used for configuring the robots) includes both fixed (non-learning) and learning coordination operators. The control team’s configuration uses a fixed selector for coordination. Learning is introduced by replacing the fixed mechanism with a learning selector. A Q-learning [8] module is embedded in the learning selector. It is acknowledged that other types of reinforcement learning approaches are also appropriate for this system. Q-learning was selected arbitrarily for this initial study. Future investigations may be undertaken to evaluate the impact of learning type on robotic systems.

At each step, the learning module is provided the current reward and perceptual state, it returns an integer indicating which assemblage the selector should activate. The Q-learner automatically tracks previous perceptions and rewards to refine its policy.

The policy an agent learns depends directly on the reward function used to train it. One objective of this research is to discover how *local* versus *global* reinforcement impacts the diversity and performance of learning teams. Global reinforcement refers to the case where a single reinforcement signal is simultaneously delivered to all agents, while with local reinforcement each agent is rewarded individually. To that end, we consider two reinforcement functions for learning soccer robots. Assuming the game proceeds in discrete steps, the global reinforcement function at timestep *t* is:

$$R_{\text{global}}(t) = \begin{cases} 1 & \text{if the team scores,} \\ -1 & \text{if the opponent scores,} \\ 0 & \text{otherwise.} \end{cases}$$

This function will reward all team members when any one of them scores. Thus a goalie will be rewarded when a forward scores, and the forward will be punished when the goalie misses a block. Observe that the global reinforcement function and the performance metric (Equation 1) are related by:

$$P = \sum_{t=0}^{t=N} R_{\text{global}}(t)$$

where N is the number of steps in the game. A close correlation between reward function and performance metric is helpful, since reinforcement learning mechanisms seek to maximize their reward. If the reward and the performance measure are similar, the agent stands a better chance of maximizing its performance. Now, consider a local function where each agent is rewarded individually:

$$R_{\text{local}}(t) = \begin{cases} 1 & \text{if the agent was closest to the ball} \\ & \quad \text{when its team scores,} \\ -1 & \text{if the agent was closest to the ball} \\ & \quad \text{when the opposing team scores,} \\ 0 & \text{otherwise.} \end{cases}$$

This function will reward the agent that scores and punish an agent that allows an opponent to score. There may not be much benefit, in terms of reward, for a robot to serve a defensive role in this model since it would receive frequent negative but no positive rewards.

4 Results

A static (non-learning) version of our Java-based soccer system was completed and utilized in the RoboCup-97 competition. Unfortunately, there was not enough time to integrate the learning system. The results reported below were generated in Georgia Tech's Java-based simulator.

Experimental data were gathered by simulating thousands of soccer games and monitoring robot performance. The learning robots are evaluated on three criteria: objective performance (score), policy convergence, and diversity of behavior.

For each trial, the learning robots are initialized with a default policy (all Q-values set to zero). A series of 100 10-point games are played with information on policy convergence and score recorded after each game. The robots retain their learning set between games. An experiment is composed of 10 trials, or a total of 1000 10-point games. Each trial uses the same initial parameters but different random number seeds (the simulations are not stochastic, but Q-learning is).

4.1 Objective Performance

When rewarded using the global reinforcement signal R_{global} , the learning teams out-score the control team by an average of 6 points to 4. The average is for all games, even during the initial phase of training. The winning margin is notable since the losing control team was hand-coded. When trained using the local reward R_{local} , the learning teams lose by an average of 4 points to 6.

4.2 Policy Convergence

Convergence is tracked by monitoring how frequently an agent's policy changes. Consider a robot that may have been following a policy of moving to the ball

	<i>mtb</i>	<i>gb</i>	<i>mtbf</i>	<i>mtb</i>	<i>gb</i>	<i>mtbf</i>	<i>mtb</i>	<i>gb</i>	<i>mtbf</i>
<i>not bb</i>	0	0	1	0	0	1	0	0	1
<i>bb</i>	0	0	1	0	1	0	1	0	0
<i>not bb</i>	0	1	0	0	1	0	0	1	0
<i>bb</i>	0	0	1	0	1	0	1	0	0
<i>not bb</i>	1	0	0	1	0	0	1	0	0
<i>bb</i>	0	0	1	0	1	0	1	0	0

Fig. 3. The nine soccer robot policies possible for the learning agents discussed in the text. Each policy is composed of one row for each of the two possible perceptual states (not behind ball or behind ball - *bb*). The position of the 1 in a row indicates which assemblage is activated for that policy in that situation. The policies of the goalie and forward robots introduced earlier (Figure 2) are in bold.

when behind it, but due to a recent reinforcement it switches to the *getBehindBall* assemblage instead. These switches are tracked as policy changes.

The data for robots rewarded using the local signal shows good convergence. The average number of changes per game drops to 0.05 after 100 games. An individual simulation to 1000 games resulted in convergence to zero. The number of policy changes for robots using R_{global} initially decreases, but does not converge in the first 100 games. The average number of policy changes is 0.25 per game after 100 games. Future simulation studies will include longer simulation runs to investigate whether convergence occurs eventually.

4.3 Behavioral Diversity

After the training phase, robots are evaluated for behavioral diversity by examining their policies. The teams are classified as hetero- or homogeneous depending on whether the robot's policies are the same. Altogether there are 9 possible policies for the learning agents since for each of the two perceptual states, they may select one of three assemblages. Figure 3 summarizes the possible policies. Based on these nine policies there are a total of 6561 possible 4 robot teams.

Two example teams, one homogeneous, the other heterogeneous are illustrated in Figure 1. The team on the left has converged to identical policies. In fact, *all* robots on the 10 locally-reinforced teams converged to the same “forward” policy used by the control team (Figure 2). All 10 teams converged to fully homogeneous behavior.

In contrast, all of the 10 globally-reinforced teams diversify to heterogeneous behavior. In all cases, the agents settle on one of three particular policies. All the teams include one robot that converges to the same “forward” policy used by the control team; they also include at least one agent that follows the same policy as the control team’s “goalie.” The other robots settle on a policy of always selecting the *getBehindBall* assemblage, no matter the situation (for convenience this policy is referred to as a “mid-back”). In cases where the team had not fully converged (zero policy changes per game), investigation reveals that the changes

are due to one agent alternating between the “goalie” and “mid-back” policies. In summary the globally-reinforced teams always converged to one “forward,” one or two “mid-backs” and one or two “goalies.”

To help quantify the varying degree of diversity in these teams, *Social Entropy* [3] is used as a measure of behavioral heterogeneity. Social Entropy, inspired by Shannon’s Information Entropy [7], evaluates the diversity of a robot society based on the number of behavioral castes it includes and the relative size of each. $\text{Het}(\mathcal{R})$, the Social Entropy of the robot society \mathcal{R} , ranges from a minimum of zero, when all agents are identical, to a maximum when each robot forms a different caste. The maximum entropy for a team of four soccer robots is 2.0. $\text{Het}(\mathcal{R}) = 0$ for the homogeneous teams trained using local reinforcement and $\text{Het}(\mathcal{R}) = 1.5$ for the heterogeneous teams. For more detail on Social Entropy, the reader is referred to [3].

5 Discussion and Conclusion

The results reported above show that in this task local reinforcement provides quicker learning, while global reinforcement leads to better performance and greater diversity. The globally-reinforced teams perform significantly better than the human-designed control team.

The locally-reinforced teams converge to “greedy” behaviors that maximize their individual reward, but lead to poor team performance. This is probably because defensive play is important in soccer but there is no incentive for a robot to fill a defensive role. With the local reward strategy a goalie would be “punished” every time the opponent scores and never receive a positive reinforcement. Quick convergence in the locally-reinforced teams is due to the close relationship between an individual agent’s actions and the rewards it receives with local reinforcement strategies.

The globally-reinforced teams perform better but do not converge to stable policies. It may be that longer experimental runs will show convergence with R_{global} reinforcement. It may also be that for complex multi-robot domains, convergence does not always occur. Either way, convergence is not a requirement for good performance: the globally rewarded teams perform significantly better than the locally reinforced teams in spite of a lack of convergence.

To conclude, the relative benefits of local versus global reinforcement in learning robot soccer teams has been evaluated in terms of team performance, learning rate, and social entropy in the resulting team. The teams were evaluated as they engaged a fixed opponent team over thousands of trials. In summary, the primary results are:

- Individual learning robots will, in many cases, automatically diversify to fill different roles on a team.
- Teams of learning robots can better the performance of human-designed teams.
- Global reinforcement leads to better performance and greater diversity, but slow policy convergence for robot teams.

- Local reinforcement leads to poorer performance and fully homogeneous behavior, but fast policy convergence.

The author thanks Ron Arkin, Chris Atkeson, Gary Boone, John Pani and Juan Carlos Santamaria for their comments on this work.

References

1. R.C. Arkin and T.R. Balch. Aura: principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), 1997. to appear.
2. T. Balch. Clay: Integrating motor schemas and reinforcement learning. College of Computing Technical Report GIT-CC-97-11, Georgia Institute of Technology, Atlanta, Georgia, March 1997.
3. T. Balch. Social entropy: a new metric for learning multi-robot teams. In *Proc. 10th International FLAIRS Conference (FLAIRS-97)*, May 1997.
4. Tucker Balch. Learning roles: Behavioral diversity in robot teams. In *AAAI-97 Workshop on Multiagent Learning*, Providence, R.I., 1997. AAAI.
5. L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
6. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proc. Autonomous Agents 97*. ACM, 1997. Marina Del Rey, California.
7. C. E. Shannon. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.
8. Christopher J. C. H. Watkins and Peter Dayan. Technical note: Q learning. *Machine Learning*, 8:279–292, 1992.

Describing Soccer Game in EAMMO

Nobuhiro Ito¹, Takahiro Hotta² and Naohiro Ishii¹

¹ Department of Intelligence and Computer Science, Nagoya Institute of Technology,
Gokiso-cho, Showa-ku, Nagoya 466, JAPAN

² Department of Division of Informatics for Natural Sciences, Graduate School of
Human Infomatics, Nagoya University, Furo-cho, Chikusa-ku, Nagoya-City, 464-01
JAPAN

Abstract. Object-oriented techniques are useful in many fields such as database systems, operating systems, programming, etc.. However, the conventional object-oriented models meet difficult to represent autonomy and flexibility of agents. To solve these problems, we proposed "an Environmental Agent Model for Multi-Objects"(EAMMO). EAMMO consists of three types of agents as follows: (1) an upper-agent describes autonomous and dynamic objects, (2) a lower-agent describes functional and static objects, and (3) an environmental-agent describes the environment including agents. It has a great influence on the agents in the environment. By using our agent model, we successfully describe modeling a soccer game for RoboCup-97. As the result, we confirmed that our agent model is a suitable paradigm to represent multi-objects.

1 Introduction

Object-oriented techniques are useful in many fields such as database systems, operating systems, programming languages, etc.[1]. When we model real-world entities, such as soccer players, as objects, the objects must have many faceted nature and dynamic nature. We call such objects "multi-objects"[2]. Multi-objects are similar to multi-agents. Multi-agents are cooperative agents and components of multi-agent systems. Multi-agents also means interactions of cooperative agents. On the other hand, a multi-object is one object having many faceted nature and dynamic nature.

Many faceted nature is that an object has different aspects(roles) in different cases. Each aspect corresponds to one case or more. Dynamic nature is that the objects must reflect changes of surroundings(other objects and environment) and represent their aspects. However, the conventional object-oriented models are suffered from shortcoming in their flexibility and ability to model both many aspects and dynamic nature of real-world entities[3].

To solve above problems, we proposed "an Environmental Agent Model for Multi-Objects"(EAMMO)[4, 5]. This agent model consists of three types of agents which are upper-agents, lower-agents, and environmental-agents as follows: (1) an upper-agent describes an autonomous and dynamic object, (2) a lower-agent describes a static object, and (3) an environmental-agent describes a passive object such as fields and rooms.These agents interact with each other

to simulate behaviors of multi-objects. By EAMMO, we successfully model a soccer game for RoboCup-97. We believe that EAMMO is a suitable model for describing multi-objects like soccer game(soccer players).

2 Multi-objects

Some real-world entities have more than one aspects, properties, and characters. When we model those entities as objects, the objects must also have more than one aspects, properties, and characters. We call such objects multi-objects[2]. For example, consider an object who is a man (named Ito); Ito is a student in Univ. A, and a teacher in Univ. B. Ito is a student from Monday to Saturday. But, on Sunday, Ito is a baseball player. ITO (multi-object) could be modeled by object-oriented models. But, the conventional object-oriented models have some problems in the description of multi-objects. For example, if the conventional object-oriented models are used to realize object views, object migration, and multiple inheritance, the maintenance of objects becomes complicated.

Multi-objects are similar to multi-agents. Multi-agents are cooperative agents and components of multi-agent systems. Multi-agents also means interactions of cooperative agents. On the other hand, a multi-object is one object having many faceted nature and dynamic nature.

3 EAMMO

3.1 Concepts of EAMMO

EAMMO(An Environmental Agent Model for Multi-Objects) is a multi-agent model that we proposed for representing multi-objects. By applying the agent-oriented paradigm[6] to multi-objects in object-oriented techniques[4, 5]. Using agent-oriented paradigm, EAMMO makes object-oriented models flexible and autonomous.

Flexibility and autonomy are useful to represent real-world entities. In real-world, some entities behave complicatedly such as creatures, and some behave simply such as tools. Besides, there is also an environment around them. Therefore, our agent model consists of three type agents, an upper-agent behaving complicatedly, a lower-agent behaving simply, an environmental-agent including them. Figure.1 shows a relationship among three type agents.

3.2 Upper-Agent

Definition 1 (Upper-Agent). An upper-agent is defined as an object that behaves autonomously and complicatedly. The upper-agent decides both its next status and behaviors with its surroundings (status of environment and target objects) and its current status. An upper-agent is shown in Fig.2

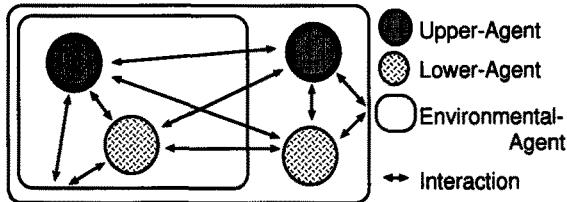


Fig. 1. Relation among three types of agents

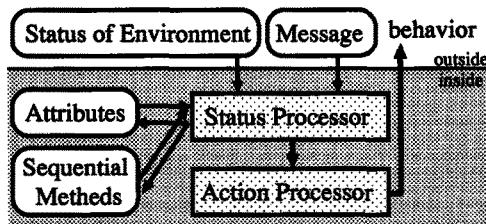


Fig. 2. An Upper-Agent

Figure. 2 shows that both the status and the action processor operate on attributes and sequential methods. Because the upper-agents have autonomous and reflective features, these agents operate on attributes and sequential methods at runtime.

Definition 2 (Status Processor). Status processor decides the next status of the upper-agent by knowing which are the target agents, the current status of the environmental-agent, and its own current status.

Each next status is one aspect of multiple aspects. Consequently, an upper-agent represents multiple aspects by this status processor. An upper-agent has aspects which are initialized to some default values. These aspects are changed by the status processor, when a requirement is satisfied by the target agents, current status of an environmental-agent, and its own current status. With the status processor and attributes, the upper-agent can perform works efficiently according to plans or scenarios such as its daily schedules.

The status decision flow consists of following three phases:

Relation-restricting phase The target agents restrict relations between agents.

Role-deriving phase The relations are restricted by the status of an environmental-agent.

Role-deciding phase The relations and roles are decided with its current status(current status of the source object). In this phase, roles are actually decided. Then next status of the (source) agent is decided.

The flow is shown in Fig.3. By an output of the status processor, the action processor actually decides its behavior.

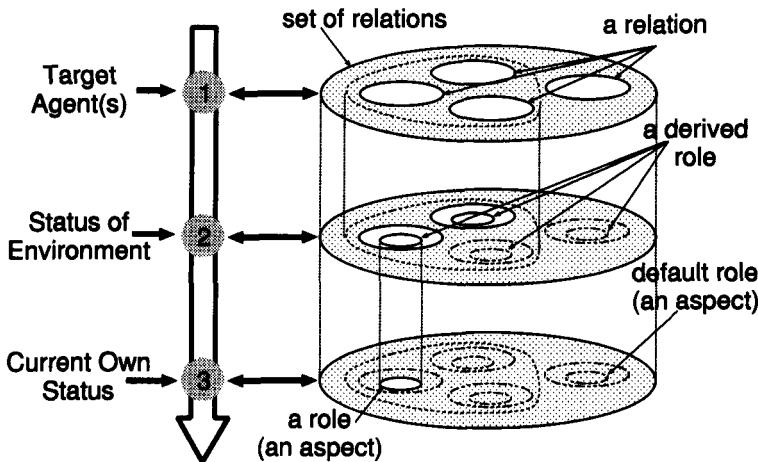


Fig. 3. Status decision flow

Definition 3 (Action Processor). Action processor decides actions(behaviors) of agents. It exactly decides actions with the current status of a source agent, an environment(environment-agent), and the body of a message.

Both upper-agents and lower-agents have an action processor. In the case of upper-agent, the action processor decides actions with the status decided by the status processor.

In upper-agents, the status processor decides one of aspects, and then the action processor invokes one of sequential methods or methods. Through these two steps, the upper-agent represents multiple aspects.

3.3 Lower-Agent

To improve efficiency of agent systems, agents are classified into upper-agents and lower-agent.

Definition 4 (Lower-Agent). A lower-agent is defined as a conventional object. It behaves reactively A lower-agent is shown in Fig.4.

As shown in Fig.4, the lower-agent doesn't have a status processor. The lower-agent only behaves with messages. So, a unique action processor operates the attributes and methods in Fig.4.

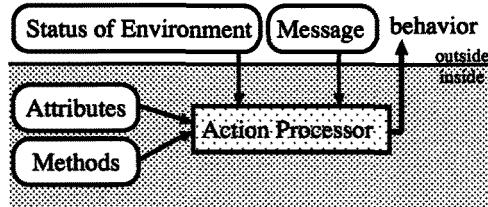


Fig. 4. A Lower-Agent

3.4 Environmental-Agent

Definition 5 (Environmental-Agent). An environmental-agent is defined as the environment where the upper-agents and the lower-agents behave. The environmental-agent is characterized by attributes, sequential methods, an environmental database, a status processor, and an action processor. The environmental database contains attributes which show the status of agents included. These attributes in the environmental database are managed by the status processor. An environmental-agent is shown in Fig.5.

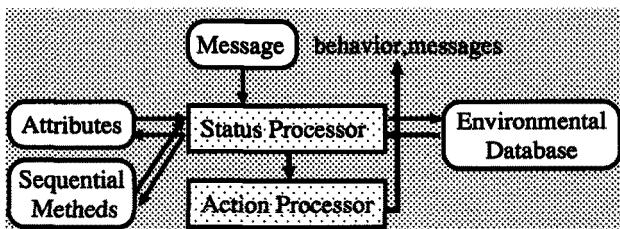


Fig. 5. An Environmental-Agent

The upper-agents and the lower-agents don't have visible sensors. But, the environmental-agent plays the visual role for the other agents, and it provides the other agents with the visual data. The visual data are stored in the environmental database. The environmental-agent can multi-cast and broadcast to the other agents with the same attribute values, by searching the environmental database for these attributes. When agents move from an environmental-agent to another environmental-agent, the result of the movement is reflected in the environmental database.

4 Describing a Soccer Game in EAMMO

4.1 Describing Soccer Game

We describe a soccer game in EAMMO. We use the rules and simulation environments offered by the Robot World Cup(RoboCup) '97 in this paper.

In the soccer field, soccer players may play autonomously and flexibly. On the other hand, when soccer players are requested from other players(goal keeper etc), they may play requested behavior passively.

When a player passes the ball to a teammate, the player who passed the ball need to behave by taking the strategies of his team into consideration. This is a positive aspect of the player. In most of the time, players behaves in this way. This aspects of player can be represented by using the upper-agent, which can behave autonomously and flexibly. By using status processor, upper-agent can decide his own behavior by taking the strategies of his team into consideration.

A player who receives the ball behaves autonomously, too. But, he often obeys the message from player that passes the ball. In the game, players communicate with each other. But when a player receives a message from other players, he should obey the message. Because the cost of communication is huge. So, a player who sends messages must send messages correctly. This is a passive aspect of the player. This aspect of players can be represented by using lower-agent which can behave reactively. When a player just need to obey the message immediately, the status processor is not necessary.

Hence, soccer players have both positive and passive aspects. Therefore, we model a hybrid agent that has both aspects of an upper-agent and a lower-agent to describe a soccer player. This agent is an upper-agent basically. But when an agent receives a message from teammates, the agent can decide his own behavior by using only the action processor.

We use Soccer Server as a simulation environment offered by the RoboCup-97. Instead of the environmental-agent, we use the Soccer Server. The Soccer Server provides each player with their visual and voice information, which is corresponding to the function of the environmental-agents.

Our soccer agents decide a role with their surroundings. Then, they play based on the role through their communications. Our agent doesn't share visible information with the other teammates. He can behave(decide a role) with partial information, completely. Each player can behave through less communication with the other players, when network traffic is heavy.

4.2 Simulation

We modeled soccer players for RoboCup-97 in our agent model. A soccer player is modeled as an upper-agent and the Soccer Server offered by the RoboCup-97 is considered as an environmental-agent. Table 1 shows results that our team competes with the champion team of the Pre-RoboCup-96 which was held at IROS-96 in Osaka, Japan. Our agents decide their role dynamically from their location and surroundings, and the champion team's agents behaves with fixed

positions. As the result of Table 1, we concluded that our agent model is a suitable model for a dynamic environment such as soccer games.

Table 1. Our team vs. Champion Team(Pre-RoboCup-96)

Our team kicks off.				Champion team kicks off.				
ours	champion	result	ours	champion	result	ours	champion	result
8	7	Win	7	2	Win	12	6	Win
7	5	Win	16	6	Win	10	8	Win
8	5	Win	6	4	Win	12	5	Win
4	6	Lose	8	5	Win	6	5	Win
9	6	Win	8	6	Win	12	8	Win
9	8	Win	10	3	Win	7	8	Lose
4	5	Lose	10	4	Win	13	11	Win
10	5	Win	7	4	Win	3	7	Lose
8	6	Win	5	3	Win	9	3	Win
9	4	Win	6	4	Win	6	5	Win
9	6	Win	5	6	Lose	8	4	Win
8	5	Win	9	6	Win	9	4	Win
8	5	Win				8	7	Win

total result(50 games) : 41 wins, 8 loss, and 1 draw(84 % wins)

5 Results of RoboCup-97

Table 2 shows the result of our team in RoboCup-97. In the first day, our team played against Team B and C. And we played against Team A in the second day. We confirmed that our agents behaved as thought. Unfortunately, we could not win these games, to our deep regret. We think of causes as following.

In RoboCup-97, participating teams were classified into two classes: synchronous behavior teams and asynchronous behavior teams. The results showed that the synchronous behaviors were much faster than the asynchronous behaviors. And the synchronous behavior overwhelmed the asynchronous behavior teams. Although EAMMO has enough ability to model a synchronous behavior team, we unfortunately have modeled our team as an asynchronous behavior one in this time, to our deep regret. Because we believed that the description of events, which occur inside an agent or among agents, is more important. As the result, our team could not win the game against Team B in the first day. Team B did not only behave synchronously, but also had a method catching up with the ball faster than other teams by a neural network. In this RoboCup, we prepared only basic methods such as pass, shoot, and so on, but EAMMO can describe more advanced methods.

In EAMMO, we modified our asynchronous team to simulate synchronous team in a day. By this modification, our team became much faster than before.

Our team can be benefited in a crowded network, because it can behave with less communications between agents. It was showed by our game against Team

C. Among three games, only the first half of one game has heavy traffic, and we got one point in this time.

We concluded that EAMMO was a suitable model for a soccer game. We will describe a new soccer model that has more advanced methods and faster behaviors in EAMMO.

Table 2. Result of RoboCup-97

	Team A	Team B	Team C	Our Team
Team A	–	3-4	1-0	7-0
Team B	4-3	–	13-0	20-0
Team C	0-1	0-13	–	8-1
Our Team	0-7	0-20	1-8	–

Team A: Jukka Riekki, University of Oulu, Finland

Team B: Peter Stone & Manuela Veloso, CMU, USA

Team C: Paul Scerri, RMIT, Australia

6 Conclusion

In soccer game, an agent(a player) should behaves autonomously and plays multiple roles in the game. On the other hand, the agent should be able to behave immediately. In this paper, we use a multi-agent model EAMMO to describe the soccer player. The former is modeled by an upper-agent with a status processor and an action processor. The latter is modeled by a lower-agent with an action processor. Furthermore, the environment like the Soccer Server providing informations is modeled by an environmental-agent. The three types of agents provide a flexible and appropriate way to describe multi-role entities such as a soccer player. Some simulations results and an analysis about our results on RoboCup-97 are also included.

References

1. J. Runbaugh, M. Blaha, W. Premerlani, F. eddy, W. Lorenzen: Object-Oriented Modeling and Design, Prentice Hall(1991).
2. S. Tsukada, T. Sugimura: MAC-model: an extended object-oriented data model for multiple classification, Computer Software, Jurnal of JSSST, Vol. 11, No. 5, (1994) 400-413
3. Y. Kambayashi, Z. Peng: Object Deputy Model and Its Applications, Proceedings of Fourth International Conference on Database System for Advanced Applications(DASFAA '95) (1995) 1-15
4. N. Ito, H. Sato, T. Hayashi: An Agent Model for Description of Multiple Objects, Technical Report of IEICE, Vol.95, No. 460, AI95-50 (1996) 63-70.
5. N. Ito, H. Sato, T. Hayashi, N. Ishii: An Agent Model for Objects with Multiple Aspects, Proceedings of IASTED International Conference (1996) 330-333.
6. Michael J. Wooldridge, Nicholas R. Jennings (Eds.): Intelligent Agents, Lecture Notes in Artificial Intelligence 890, Springer-Verlag(1995).

Team GAMMA: Agent Programming on Gaea

NODA, Itsuki¹

noda@etl.go.jp

ETL

Umezono 1-1-4, Tsukuba

Ibaraki 305, JAPAN

1 Introduction

We are developing a new software methodology for building large, complicated systems out of simple units[Nakashima, 1991]. The emphasis is on the architecture used to combine the units, rather than on the intelligence of individual units.

We call the methodology “organic programming”, referring to the characteristics of organic systems, *context dependency* and *partial specification*. Our approaches toward those characteristics are situatedness and reflection. Situatedness corresponds to the former characteristic, and reflection to the latter.

In this paper we describe an application of organic programming language, Gaea, to the programming of players of a multi-agent game, soccer. Our approach to multiagent systems is recursive. Each agent is programmed in a multiagent way.

In programing a complex agent like soccer player, we need various kinds of mechanisms of mode-change, interruption and emergency exit. We propose “dynamic subsumption architecture” as a flexible methodology to realize such mechanisms.

2 Gaea

Gaea[Nakashima *et al.*, 1996] is a logic programming language using organic programming methodology [Nakashima, 1991]. From the viewpoint of logic programing, Gaea is a variation of Prolog with the following new features:

- Multi-process (multi-thread): In Gaea, we can start a new process by “fork(GOAL)” predicate, by which “GOAL” is solved independently from the original process. All process share cells describes below.
- Multi-environment: In Gaea, definitions of predicates are stored in “cells”. “Cell” is similar to “module” in recent prolog systems. The main difference of cell and module is that cell-structure (environment described below) is dynamic, while relation between module is static. Each process has an environment that is a list of cells, and searches for matching definitions in the list. Moreover, the process can manipulate its and other’s environment dynamically.

Figure 1 shows a conceptual image of “processes” and “cells”.

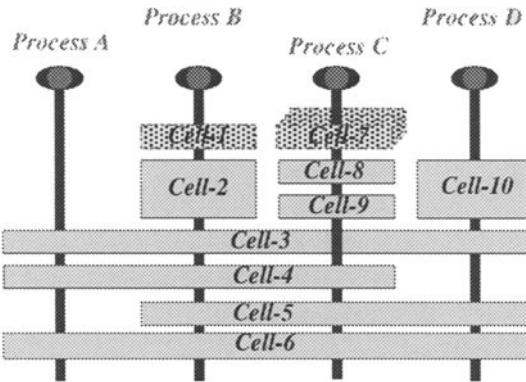


Fig. 1. Multi-Process and Multi-Environment

3 Programming Players in Gaea

3.1 Dynamic Subsumption Architecture

In programming a complex agent like a soccer player, it is essential that the agent accepts many modes. The same agent may respond to the same input differently according to the mode. For example, in soccer, a player may react differently to a ball according to whether his team is on offense or defense.

The mode of the agent must be changeable flexibly. The mode changes as a result of agent's action and plan, and it also changes according to the change of situation. Moreover, modes will be divided into a couple of levels. For example, in soccer, "offensive mode" and "defensive mode" are relatively high-level modes. On the other hands, "chase-ball mode" and "dribble mode" are relatively low-level modes. In addition to it, we can consider a couple of types of changes of modes. For example, an agent may "goto" a new mode, or it may "enter" a new mode and "return" when it exits from the new mode.

In order to realize such flexible mode-change, we use an extension of subsumption architecture[Brooks, 1991], called *dynamic subsumption architecture*. This architecture is the combination of subsumption architecture and dynamic environment change. Since subsumption architecture assumes fixed layers of functions, it is either difficult or inefficient to implement multiple modes on top of it. It is straightforward in organic programming, using context reflection.

We assume the following conditions for dynamic subsumption architecture:

1. Overriding is done by name
2. The same ontology (global name) is used by all cells.

Actually, dynamic subsumption architecture is implemented using dynamic environment manipulation (Figure 2).

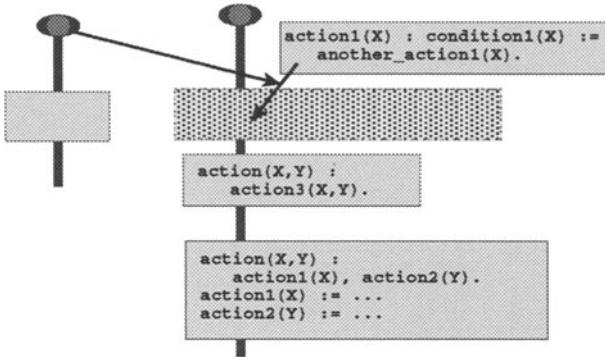


Fig. 2. Dynamic Sybsumption Archtecture in Gaea

3.2 Overview of the Architecture of a Soccer Player

As described in Section 1, we program an agent in multi-agent way. A soccer player consists of the following processes(agents):

- **Sensor Process:** receives sensor informations sent from the Soccer Server, analyzes them, and puts the results into the common cell.
- **Command Process:** sends control commands to the Soccer Server. It is controlled to send one command per 100 milli-seconds, because Soccer Server accepts only one command per 100 milli-seconds. In other words, this process is a resource manager of sending control commands.
- **Action Process:** controls low-level modes of player’s action by manipulating the environment of the command process.
- **Object Detection Process:** checks the way to the target of chasing or kicking, and changes the behavior by modifying the environment of the command process. if there are objects on the way,
- **Communication Process:** controls high-level modes of player’s action according to the message from referee and teammates.

The top level of each process is a loop that repeatedly calls “`cycle(top)`”. It is defined in the “`basic`” cell, that is shared all processes, as follows:

```
/* In "basic" cell */
toplevel() := repeat(cycle(top)).
```

3.3 Basic Action Modes

A player is situated in certain action modes, such as *chase-ball*, *dribble*, *shoot*, *pass*, and so on, during the play. Each mode is defined in one or in a set of cells.

The mode is changed by pushing cells into the environment of the command process of the player. For example, in *chase-ball* mode, the role process pushes the following “*chase*” cell into the environment of the action process. In each cell, “*cycle(top)*” for the command process is defined accordingly.

```
/* In "chase" cell */
cycle(top) ::==
    target(Target),
    /* turn to the target */
    cycle(turn(Target)),
    /* and dash */
    cycle(dash(Target)).
cycle(turn(Target)) ::==
    direction(Target,Dir),
    turn(Dir)./* send turn com. */
cycle(dash(Target)) ::==
    distance(Target,Dist),
    dash(Dist)./* send dash com. */
target(ball).
```

In Gaea, the environment of a process can be manipulated by the process itself and also by other processes. In our program, the action process manipulates the environment of the command process in order to control basic action mode. The followings are a sample of definition of “*cycle(top)*” for the action process.

```
/* In "dribbler" cell */
cycle(top) :
    distance(ball,BDist), BDist < 2,
    /* if ball is near */
    distance(goal,GDist), GDist < 10 := 
    /* and in front of goal */
    change_command_env([shoot]).
    /* then shoot !! */
cycle(top) :
    distance(ball,BDist), BDist < 2 :=
    /* if ball is near */
    change_command_env([dribble]).
    /* then start dribble */
cycle(top) :=
    change_command_env([chase]).
    /* otherwise chase the ball */
    /* Replacing new mode cells */
    /* into the environment of */
    /* the command process */
    change_command_env(NewMode) :
```

```

command_process(PID),
environment(Env,PID),
include(NewMode,Env) := .
change_command_env(NewMode) :
command_process(PID),
environment(Env,PID),
remove_mode_cell(Env, NEnv),
append(NewMode, NEnv, NewEnv),
set_environment(NewEnv, PID).

```

In this program, the action process checks the situation of the field, selects one “shoot”, “dribble” and “chase” cells, and swaps it with the current mode cell in the environment of the command process.

The merit of this architecture, in which the action process manipulate the environment of the command process, is that the command process can send commands constantly even if it take a lot of time for the action process to check the situation.

Control of high-level mode is also realized in the same manner. In this case, communication process manipulate the environment of the action process in order to change higher-level of behavior like “chasing ball” and “supporting” (Figure 3).

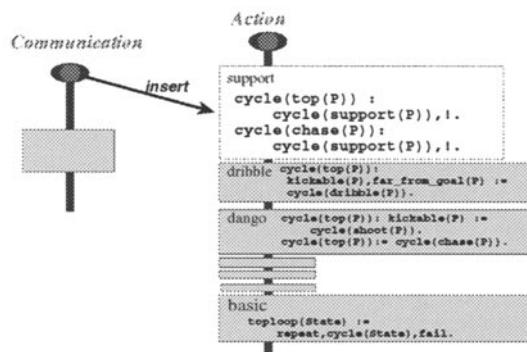


Fig. 3. Changing Behavior by Communication Process

3.4 Modifying Basic Actions

It is easy to realize small modification of behavior by manipulating environments.

In order to avoid objects, the object detection process pushes the following “**avoid**” cell into the environment of the command process.

```
/* In "avoid" cell */
target(Target) :
    remove_cell(avoid),
    /* remove this cell */
    target(T1),
    /* find original target */
    Target = T1 + rel_dir(60).
    /* shift relative angle in 60 */
```

This cell only override the definition of “`target(Target)`” rather than “`cycle(top)`”. Moreover, this definition is used only once, because the “`avoid`” cell is removed when “`target(Target)`” is called. (See `remove_cell`.)

The program for the object detection process is as follows:

```
/* In "check_object" cell */
cycle(top) :
    command_process(PID),
    environment(Env,PID),
    in(Env,target(Target)),
    /* get Target information */
    /* in the command environment */
    not_clear_to(Target),
    /* check if the target direction */
    /* is clear */
    add_action_env([avoid]).
    /* add avoid into the */
    /* command environment */
```

3.5 Interruption

Interruption is a kind of temporal change of modes. The feature of interruption is to suspend current job, execute another job, and return to execute the current job.

For example, consider the case that the player must reply its position immediately when a teammate says “`tell your position`”. This behavior can be realized by pushing the following cell into the environment of the command process.

```
/* In "tell_pos" cell */
cycle(_) :
    estimate_current_pos(X,Y),
    unum(UNum),
    say([UNum,position,X,Y]),
    remove_cell(tell_pos),
    fail.
```

This definition is used whenever “*cycle/1*” is called, and discarded after the execution in failure. Then original definition of “*cycle/1*” is called.

We can control the level of interruption by specifying the argument of “*cycle/1*”. For example, consider the case the player loses site of a ball when it is chasing the ball. In this case, the other process will interrupt the command process to search the ball. In the chase mode, the command process calls “*cycle(turn)*” and “*cycle(dash)*” sequentially. If the interruption is defined as “*cycle(_)*” like the above example, the definition is executed when “*cycle(dash)*” is called. But the interruption of searching ball will change player’s direction, so that the player may dash to the wrong direction. In order to avoid this, we can simply define this interruption as “*cycle(top)*” as follows:

```
/* In "search" cell */
cycle(top) :
    target(Target),
    cycle(look(Target)),
    remove_cell(search).
cycle(look(Target)) :-
    cycle(turn(Target)),
    cycle(wait_visual_sensor),
    new_info(Target), cut.
cycle(look(Target)) :-
    cycle(search(Target)).
cycle(search(Target)) :-
    turn(90),
    cycle(wait_visual_sensor),
    new_info(Target), cut.
cycle(search(Target)) :-
    cycle(search(Target)).
cycle(wait_visual_sensor) :-
    current_time(CTime),
    repeat,
    usleep(100000),
    cycle(v_sensor_newer(CTime)), cut.
cycle(v_sensor_newer(CTime)) :
    visual_sensor_time(VTime),
    VTime > CTime, cut.
```

We can use any kinds of infon on the argument of “*cycle/1*”, so that it is able to realize flexible control of interruption.

3.6 Emergency Exit

It is also possible to realize emergency exit of execution of plays. For example, consider the case that the player must go back to its own goal immediately because of the clutch. This behavior is realized by pushing `emergency`, `guard_goal`

and **chase** cells into the environment of the command process, where these cells have the following definitions.

```

/* In "emergency" cell */
cycle(top) :
  /* when cycle(top) is called, */
  /* exit emergency */
  remove_cell(emergency) :=
    fail.
  /* all other cycle(_) fails */
cycle(_) := fail.

/* In "guard_goal" cell */
target(owngoal).

```

4 Conclusion

Thorough our experience of programming soccer players in Gaea, we are convinced that the language is suitable for complex multiagent programming.

In programing complex agents, we need various kinds of mechanisms of mode-change, interruption and emergency exit. We proposed “dynamic subsumption architecture” as a flexible methodology to realize such mechanisms.

We could also program the system in layer-by-layer manner, from a simple behavior to more complex ones, taking full advantage of subsumption architecture design.

References

- [Brooks, 1991] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–160, 1991.
- [Nakashima *et al.*, 1996] Hideyuki Nakashima, Itsuki Noda, Kenichi Handa, and John Fry. GAEA programming manual. TR-96-11, ETL, 1996. Gaea system is available from <http://cape.etl.go.jp/gaea/>.
- [Nakashima, 1991] Hideyuki Nakashima. New models for software architecture project. *New Generation Computing*, 9(3,4):475–477, 1991.

Using Reactive Deliberation for Real-Time Control of Soccer-Playing Robots

Yu Zhang and Alan K. Mackworth

Department of Computer Science, University of British Columbia, Vancouver B.C.
V6T 1Z4, Canada, yzhang@cs.ubc.ca, mack@cs.ubc.ca

Abstract. Soccer meets the requirements of the Situated Agent approach and as a task domain is sufficiently rich to support research integrating many branches of AI. Reactive deliberation is a robot architecture that combines responsiveness to the environment with intelligent decision making. Under Reactive Deliberation, the robot controller is partitioned into a deliberator and an executor; the distinction is primarily based on the different time scales of interaction. A controller for our team entry in the Robocup97 Simulation League, *UBC Dynamo97*, has been developed using the Reactive Deliberation architecture.

1 Introduction

The Good Old Fashioned Artificial Intelligence and Robotics (GOFAIR) [3] research paradigm has shaped the area of intelligent robotics since the time of the robot Shakey. Some of the typical fundamental assumptions made about the world were that there is only one agent, that the environment is static unless the agent changes it, that actions are discrete and are carried out sequentially and that the world the robot inhabits can be accurately and exhaustively modeled by the robot. These assumptions proved to be overly restrictive and ultimately sterile. In the usual dynamic of the scientific dialectic, a new movement has emerged as a synthesis of GOFAIR and “Nouvelle AI”: the Situated Agent approach. A situated agent is a real physical system grounded and embedded in a real world, here and now, acting and reacting in real-time. Mackworth [3] proposed that playing soccer be a paradigmatic task domain since it breaks with nearly all of the restrictive assumptions on which GOFAIR is based and meets the standards proposed in the Situated Agent approach. The soccer domain has the following characteristics:

1. Neutral, friendly, and hostile agents
2. Inter-agent cooperation and communication
3. Real-time interaction
4. Dynamic environment
5. Real and partially unpredictable world
6. Objective performance criteria
7. Repeatable experiments

Soccer meets the requirements of the Situated Agent approach and as a task domain is sufficiently rich to support research integrating many branches of AI [4].

The Dynamite testbed has been developed in our laboratory for testing theories in the soccer domain using multiple mobile robots [1]. The testbed consists a fleet of radio controlled vehicles that perceive the world through a shared overhead perceptual system. The Vision Engine produces the absolute position of all the objects on the soccer field. Each vehicle is controlled by a distributed user program running on two transputer nodes. The movement of all vehicles is controlled through radio transmitters attached to a single shared transputer node. A physics-based real-time graphics simulator for the Dynamite world is also available for testing and developing reasoning and control programs [2, 5].

Two approaches are taken in our laboratory to robot control in the soccer domain: *Constraint Nets (CN)* and *Reactive Deliberation*. CN is a formal model for robotic systems and behaviours. CN programs are composed of modules with I/O ports. CN provides a theoretical foundation for systems design and analysis. The dynamics and control for each robot, as well as the interaction between robots, can be modeled using CN [3, 9].

Reactive deliberation [7] is a robot architecture that combines responsiveness to the environment with intelligent decision making. Several controllers based on reactive deliberation have been implemented to allow Dynamites (soccer robots) to compete in complete two-on-two games of soccer [4, 7]. A controller for our team entry in the Robocup97 Simulation League, *UBC Dynamo97*, has also been developed using the Reactive Deliberation architecture.

2 The Reactive Deliberation Architecture

Reactive deliberation is a robot architecture that integrates reactive and goal-directed activity. Even deliberation must be, to some extent, reactive to respond to changes in the environment. Under reactive deliberation, the robot controller is partitioned into a deliberator and an executor; the distinction is primarily based on the different time scales of interaction. Informally, the deliberator decides what to do and how to do it, while the executor interacts with the environment in real-time. These components run asynchronously to allow the executor to interact continuously with the world and the deliberator to perform time consuming computations.

2.1 The Executor

The executor is composed of a collection of action schemas. An *action schema* is a robot program that interacts with the environment in real-time to accomplish specific actions. The deliberator enables a single action schema with a set of run-time parameters that fully defines the activity. Only one action schema is enabled at a time and it interacts with the environment through a tight feedback loop. In the world of real-time control there is no room for time consuming planning

algorithms. Computations in action schemas are restricted to those that can keep pace with the environment, so lengthy computations are performed in the deliberator.

There are three action schemas implemented in the controller for UBC Dynamo97, our team in the Robocup97 Simulation League:

1. *Intercept ball* generates a sequence of actions to move to the ball according to the dynamics of the ball and the robot.
2. *Goto position* generates a sequence of actions to move to a certain position on the field.
3. *Kick ball* decides how much force the player should use to kick the ball.

2.2 The Deliberator

The focus of the deliberator is on an effective mechanism for selecting actions or goals in a timely manner. A central feature of reactive deliberation is that the deliberator is composed of concurrently active modules called *behaviours* that represent the goals of the robot.

A *behaviour* is a robot program that determines an action that may, if executed, bring about a specific goal. Behaviours *propose* actions whereas action schemas *perform* actions. Each behaviour must do the following: 1) select an action schema, 2) compute run-time parameters for the schema (plan the action), and 3) generate a bid describing how appropriate the action is. The most appropriate behaviour, and hence action, is determined in a distributed manner through inter-behaviour bidding.

Each bid is an estimate of the expected utility of the proposed action and is based on the current state of the world as well as the results of planning. Currently, the criteria for generating the bids are hand-coded and tuned so that the most appropriate behaviour is active in each situation. This approach requires the designer of a system to explicitly state the conditions under which certain behaviours are suitable or favorable. The bids are simple algebraic formulas that are easily computed. Each behaviour has a basic bid that is modified through the addition and subtraction of weighted factors that depend on the environment and the results of motion planning. At any point in time, the *ruling behaviour* is the behaviour with the currently highest bid.

The principal advantage of behaviour-based bidding is modularity. Since bids are calibrated to an external measure of utility, behaviours can be added, modified or deleted without changing the bidding criteria of the established system. A new behaviour must, of course, be tuned to be compatible with existing ones. Behaviours are independent, so they can have different representations and approaches to generating actions. There is no central decision maker that evaluates the world and decides the best course of action, so behaviours can be run concurrently on different processors (instead of timesharing a single processor), thus improving the speed of the system.

There are five behaviours implemented in the controller for UBC Dynamo97:

1. *Intercept ball.* The player intercepts the ball. It has a base bid that is modified by environmental factors and planning results. Some of the factors are: the speed and heading of the ball, the distance to the ball, the speed and heading of other players and their distances to the ball.
2. *Defend.* The player goes to its assigned defensive position.
3. *Offend.* The player goes to its assigned offensive position. The Defend and Offend behaviours have the same factors affecting their bids as the first behaviour has. These defensive and offensive positions could be set by learning algorithms, though here they are set by hand.
4. *Kick.* When the player can kick the ball, he should choose a direction in which to kick the ball. He should decide whether he should pass the ball to his teammates or shoot it at the goal. (He shouldn't kick the ball to his opponents but sometimes this does happen because of imperfect information coming from the soccer server.) A sub-bid system is used here to choose the appropriate sub-behaviour among passing, shooting and kicking to an open area. Passing is a inter-robot cooperation problem. Due to the restricted communication mechanism provided by soccer server, we don't use any explicit communication to implement inter-robot cooperation. Using communication, each robot would broadcast its intended action and a bid which estimates the appropriateness of that action. Robots whose actions are in conflict would reevaluate their decision to include the bid information of other robots. A robot with a lower bid than another robot will reevaluate its internal bid for that action, since that action may not be the best one in the current situation.
5. *Avoid.* When the player is obstructed by another player, it plans to find a new path.

3 Conclusions and Future Work

Under Reactive Deliberation, the robot controller is partitioned into a deliberator and an executor; the distinction is primarily based on the different time scales of interaction. Informally, the deliberator decides what to do and how to do it, while the executor interacts with the environment in real-time. These components run asynchronously to allow the executor to interact continuously with the world and the deliberator to perform time-consuming computations.

In Reactive Deliberation, there is no central decision maker that evaluates the world and decides on the best course of action, so behaviours can be run concurrently on different processors, thus improving the speed of the system.

Some learning techniques could be used here to estimate the bid for each behaviour and to set the defensive and offensive positions.

Using constraint-based behaviours, the Constraint Net and Reactive Deliberation approaches could be integrated for a more formal and modular approach.

References

1. R. A. Barman, S. J. Kingdon, J. J. Little, A. K. Mackworth, D. K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. DYNAMO: Real-time experiments with multiple mobile robots. In *Intelligent Vehicles Symposium*, pages 261–266, Tokyo, July 1993.
2. R. A. Barman, S. J. Kingdon, A. K. Mackworth, D. K. Pai, M. K. Sahota, H. Wilkinson, and Y. Zhang. Dynamite: A testbed for multiple mobile robots. In *Proc. IJCAI Workshop on Dynamically Interacting Robots*, pages 35–45, Chambéry, France, August 1993.
3. A. K. Mackworth. On seeing robots. In A. Basu and X. Li, editors, *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, Singapore, 1993.
4. M. Sahota and A. K. Mackworth. Can situated robots play soccer? In *Proc. Artificial Intelligence 94*, pages 249 – 254, Banff, Alberta, May 1994.
5. Michael K. Sahota. Dynasim user guide. Available at <http://www.cs.ubc.ca/nest/lci/soccer>, January 1996.
6. Michael K. Sahota. Real-time intelligent behaviour in dynamic environments: Soccer-playing robots. Master's thesis, University of British Columbia, August 1993.
7. Michael K. Sahota. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. In *AAAI94*, pages 1303–1308, 1994.
8. Michael K. Sahota, Alan K. Mackworth, Rod A. Barman, and Stewart J. Kingdon. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3690–3663, 1995.
9. Y. Zhang and A. K. Mackworth. Synthesis of hybrid constraint-based controllers. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 552 – 567. Springer Verlag, 1995.

A Multi-layered Planning Architecture for Soccer Agent

Ransui Iso (*ransui@ina-lab.ise.aoyama.ac.jp*)
Hiroshige Inazumi (*hiro@ina-lab.ise.aoyama.ac.jp*)

Dept. of Industrial and Systems Engineering
Aoyama Gakuin University.

Abstract

Based on manual simulation experiments, we propose a type of agent as a Multi-Layerd Planning(MLP) Architecture to identify the game situation and create action policy. As a team, we will arrange the various type of agent according to game strategy, and realize semi-cooperative Multiagent model with minimum amount of communication.

1 Introduction

Simulated Robotic Soccer is a typical and interesting AI domain, especially Multiagent domain. RoboCup97 will include its tournament using the Soccer Server system. Soccer Server captures enough real-world complexities to be a very challenging domain.

This simulator is realistic in the following way;

1. players' vision is limited,
2. players can communicate by posting to a blackboard that is visible to all players,
3. each player is controlled by a separate process,
4. players have limited stamina,
5. actuators and sensors are noisy,
6. dynamics and kinematics are modelled, and
7. play occurs in real time: the agents must react to their sensory inputs at roughly the same speed as human or robotic soccer players.

The simulator, acting as a server, provides a domain and supports users who wish to build their own agents.

We firstly developed several type of field viewer and manual simulation tools, based on data only from soccer server, like Video Games, that is limited area of

soccer field, we play soccer game with those tools. We have investigated which kind of data are important and useful, how human beings make decisions about action rules, and so on.

We consider, agents must have a following ability.

1. Quick desition to dynamic environment
2. Self learing to low-level skill and hi-level strategy
3. Easy programming to build a agent

We made 2 types of experiment agents before RoboCup97. Each agents are contains following ideas.

1.1 View-Information-Oriented Model

In view-information-oriented model, the main purpose is to reinforce abilities of agent as soccer players. This model consists of two shemes. i.e., analyzing scheme and reasoning scheme. Analyzing scheme works as a filter from view data given by soccer server to useful information of objectsm for the agent. In Reasoning scheme, each agent bridges time-gap of view data from soccer server, estimates position-information, and decides suitable action by using heuristic rules satisfying robustness. Although any agent dosen't always calculate the parameters of estimated position and action exactly.

1.2 Q-Learning Model with Multi-Demensional State-Transition

Daiji Tamagwa who is one of our teammate, makes a agent that uses Q-Learning Model with Multi-Dementional State-Transition(MDST) [1]. MDST provides state-transition that sparated by object status in soccer field that is called “Sub-State-Transition” . This architecture provided quick decition and self learning, but program and data structures are very complexly.

2 Multi-layered Planning Architecture

2.1 Environment Modeler

Environment Modeler, as a preliminaru module for planning, makes ObjectList structure from object information from Soccer Server. ObjectList holds information of objects to which other modules easily made reference.

2.2 Planner

Planning module, as core of MLP, reads the object list and creates command text sequence that can be sent to Soccer-Server directly (see fig1).

We construct our agent under Multi-Layerd Planning (MLP) Architecture. MLP is made up by following function layers.

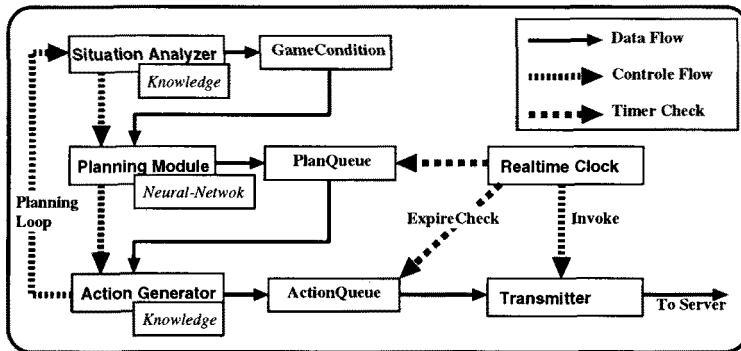


Fig. 1.

Situation Analyzer : Situation Analyzer generates identifier of game condition from ObjectList. It has simple rule database for analyzing game situation. It contains some rule pair of predicate for object condition and identifier of game situation.

Planner : Planner layer is main function of MLP. Planner generating action-policy that made up by action-style, and direction of sending a ball. Action-style describes what's agent must to do. The value could be assigned one of 3 attributes, such as "Attack", "Defense" and "Normal". Those 2 elements of action-policy generated by perceptron type Neural-Network(NN) that implemented in the Planner.

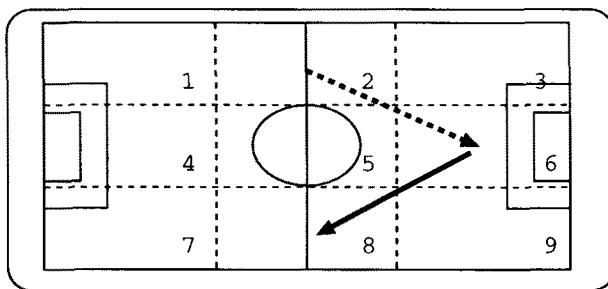


Fig. 2.

We divided soccer field to express ball position (see fig2). NN in planner takes ball transition data that made up a pair of past ball position and present ball position, and outputs pair of action-policy and direction of sending a ball.

For example. Ball moves from field-section 2 to 6, NN takes a pair of ball position, such as (2 6), then NN outputs a pair of action-policy and direction of sending a ball, such as ("Defense" 8).

Now, NN is teached how to decition to make a correct action-policy by our hand, not learning itself.

Action Generator : Action generator creates command-text sequence from a pair of action-policy , derection of sending a ball which generated by planner, and ObjectList that generated by Environment Modeler. The command-text can be send to Soccer-Server directly.

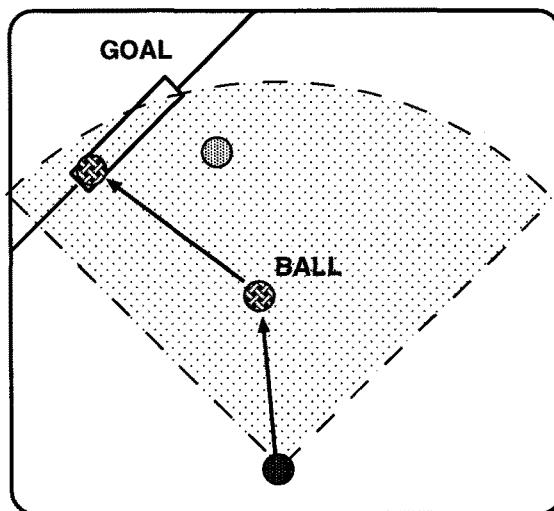


Fig. 3.

Action generator references Object-List and Action-policy to make a command-text sequence. For example, see upper figure. A Agent position is front of a goal, and he can see a ball that is free. If planner made a action-policy that say “Get a goal” such as “(4 G)¹”, and send a action generator. Action generator searches a “Get a goal” pattern in action list that made up by skelton of command sequence. Agent uses following skelton of command sequence for “Get a Goal”.

```

if distance(BALL) ≤ 2.0
  if distance(GOAL) ≥ 50
    power = 100.0
  if distance(GOAL) ≤ 50
    power = distance(GOAL) × 2
  kick(power, derection(GOAL))
  done

```

¹ Symbol 'G' is a Goal symbol"

```

if distance(BALL) > 2.0
  turn(derection(BALL))
  if (distance(BALL) ≤ 3.0
    power = distance(BALL) / 3.0
    dash(power)
    done
  if (distance(BALL) > 3.0)
    n = distance(BALL) / 3.0
    ∑i=1n dash(100)
    done

```

In this case, agent made a following command sequence. (Ball position is (-5.0 15.0), Goal position is (-20.0 30))

1. turn(-5.0)
2. dash(100)
3. dash(100)
4. dash(100)
5. dash(100)
6. dash(100)
7. kick(-10.0, 100)

Real-Time Clock : Real-Time clock module controls all of the layers. When agent wants to send any command-text to Soccer-Server, it must have some interval time to send, because SoccerServer can execute only one command in 0.1 second time slice. Therefore agent must send to one command every 0.1 second time slice. Real-Time Clock also provides expire time for old plan. If agent have old plan that can not apply to new game situation , agent must creates new plan for new game situation.

3 Discussion

The MLP is useful architecture to make agent that lives in complexly environment. We consider MLP has 3 features.

First, MLP is made up some modules. Each modules are independence by others. This mean we can make modules separately, and we can implement different methods each modules. now we implemented rule-based decition in SituationAnalyer, and Neural-Network in Planner.

Second, All functions module in MLP can run separately. For example, If planner-module is too hevy to run in machine that you use, you can run a planner-module in faster machine, and you can use network connection for exchange information between modules.

Finally, MLP containg Real-Time Clock module. That module provides time information to modules. Modules can trace environment transition and use past time information effective. now we use time information to timing of sending command and expire a plan.

References

1. Daiji Tamagawa, "Reinforced Learning in Dymamic Environment with Multi-Dimentional State-Trasition" *Aoyama Gakuin University, Graduation thesis 1996*

Author Index

- Achim, S. 242
Adibi, J. 123, 295
Adobbatı, R. 295
Al-Onaizan, Y. 123
Andou, T. 373
André, E. 200
Asada, M. 1, 20, 42, 305
Asama, H. 42, 333
- Balch, T. 181, 483
Borrajo, D. 475
Burkhard, H-D. 357
- Cheng G. 144
Cho, B. 295
Ch'ng, S. 458
Coradeschi, S. 62, 112
- D'Angelo, A. 434
De la Rosa, J.Ll. 286
Del Acebo, E. 286
Doncker, S. 277
Drogoul, A. 42, 277
Duhaut, D. 42, 277
- Erdem, A. 123, 295
Esteva, S. 286
- Farris, J. 398
Ferrari, C. 434
Fernndez, C. 475
Figueras, A. 286
Frank, I. 216
Fry, J. 450
Fujita, M. 168
Funes, P. 348
- Garcia, R. 286
- Han, K. 242
Hannebauer, M. 357
Herzog, G. 200
Hendler, J. 398
Hohn, C. 398
- Hotta, T. 492
Huang, L. 450
Humet, J. 286
- Idasiak, V. 277
Igarashi, H. 420
Inazumi, H. 513
Inden, T. 443
Ishii, N. 492
Ishizuka, H. 305
Iso, R. 513
Ito, N. 492
- Jackson, G. 398
Jennings, A. 88, 320
- Kaetsu, H. 333
Kageyama, K. 168
Kaminka, G. A. 123
Karlsson, L. 112
Kato, T. 305
Kawabata, K. 333
Kawamoto, Y. 156
Kendall, E. 88
Kitano, H. 1, 20, 42, 62, 168
Kneen, J. 320
Kourogi, M. 156
Kosue, S. 420
Kuniyoshi, Y. 1
- Luke, S. 398
- Mackworth, A. 508
Marsella, S. C. 123
Matellán, V. 475
Matsubara, H. 1, 20, 62
Matsumoto, A. 132, 333
Mishima, C. 305
Miyahara, M. 420
Mizuno, H. 156
Montesello, F. 434
Moradi, H. 295
Muslea, I. 123
Muraoka, Y. 156

- Nagai, H. 132
Nakamura, M. 305
Nakamura, T. 257
Noda, I. 1, 20, 62, 500
Ohta, M. 412
Oller, A. 286
Osawa, E. 1, 62
Ozaki, K. 333
Padgham, L. 458
Pagello, E. 434
Peters, S. 450
Price, A. 88, 320
Ramon, J.A. 286
Riekki, J. 74
Rist, T. 200
Rocher, S. 277
Röning, J. 74
Salemi, B. 295
Sargent, R. 348
Schneider-Fontán, M. 348
Scerri, P. 467
Shen, W-M. 295
Shinjoh, A. 188
Stone, P. 42, 62, 99, 242, 389
Suzuki, S. 20, 305
Takaki, S. 428
Takahashi, T. 443
Takahashi, Y. 305
Tallis, M. 123
Tambe, M. 123
Tejada, S. 295
Uchibe, E. 305
Umaba, T. 420
Veloso, M. 42, 62, 99, 242, 389
Verner, I. M. 231
Wendler, J. 357
Werger, B. 348
Witty, C. 348
Witty, T. 348
Yokota, K. 333
Zelinsky A. 144
Zhang, Y. 508

Lecture Notes in Artificial Intelligence (LNAI)

- Vol. 1228: S.-H. Nienhuys-Cheng, R. de Wolf, Foundations of Inductive Logic Programming. XVII, 404 pages. 1997.
- Vol. 1229: G. Kraetschmar, Distributed Reason Maintenance for Multiagent Systems. XIV, 296 pages. 1997.
- Vol. 1236: E. Maier, M. Mast, S. LuperFoy (Eds.), Dialogue Processing in Spoken Language Systems. Proceedings, 1996. VIII, 220 pages. 1997.
- Vol. 1237: M. Boman, W. Van de Velde (Eds.), Multi-Agent Rationality. Proceedings, 1997. XII, 254 pages. 1997.
- Vol. 1244: D. M. Gabbay, R. Kruse, A. Nonnengart, H.J. Ohlbach (Eds.), Qualitative and Quantitative Practical Reasoning. Proceedings, 1997. X, 621 pages. 1997.
- Vol. 1249: W. McCune (Ed.), Automated Deduction – CADE-14. Proceedings, 1997. XIV, 462 pages. 1997.
- Vol. 1257: D. Lukose, H. Delugach, M. Keeler, L. Searle, J. Sowa (Eds.), Conceptual Structures: Fulfilling Peirce's Dream. Proceedings, 1997. XII, 621 pages. 1997.
- Vol. 1263: J. Komorowski, J. Zytkow (Eds.), Principles of Data Mining and Knowledge Discovery. Proceedings, 1997. IX, 397 pages. 1997.
- Vol. 1266: D.B. Leake, E. Plaza (Eds.), Case-Based Reasoning Research and Development. Proceedings, 1997. XIII, 648 pages. 1997.
- Vol. 1265: J. Dix, U. Furbach, A. Nerode (Eds.), Logic Programming and Nonmonotonic Reasoning. Proceedings, 1997. X, 453 pages. 1997.
- Vol. 1285: X. Jao, J.-H. Kim, T. Furuhashi (Eds.), Simulated Evolution and Learning. Proceedings, 1996. VIII, 231 pages. 1997.
- Vol. 1286: C. Zhang, D. Lukose (Eds.), Multi-Agent Systems. Proceedings, 1996. VII, 195 pages. 1997.
- Vol. 1297: N. Lavrač, S. Džeroski (Eds.), Inductive Logic Programming. Proceedings, 1997. VIII, 309 pages. 1997.
- Vol. 1299: M.T. Pazienza (Ed.), Information Extraction. Proceedings, 1997. IX, 213 pages. 1997.
- Vol. 1303: G. Brewka, C. Habel, B. Nebel (Eds.), KI-97: Advances in Artificial Intelligence. Proceedings, 1997. XI, 413 pages. 1997.
- Vol. 1307: R. Kompe, Prosody in Speech Understanding Systems. XIX, 357 pages. 1997.
- Vol. 1314: S. Muggleton (Ed.), Inductive Logic Programming. Proceedings, 1996. VIII, 397 pages. 1997.
- Vol. 1316: M.Li, A. Maruoka (Eds.), Algorithmic Learning Theory. Proceedings, 1997. XI, 461 pages. 1997.
- Vol. 1317: M. Leman (Ed.), Music, Gestalt, and Computing. IX, 524 pages. 1997.
- Vol. 1319: E. Plaza, R. Benjamins (Eds.), Knowledge Acquisition, Modelling and Management. Proceedings, 1997. XI, 389 pages. 1997.
- Vol. 1321: M. Lenzerini (Ed.), AI*IA 97: Advances in Artificial Intelligence. Proceedings, 1997. XII, 459 pages. 1997.
- Vol. 1323: E. Costa, A. Cardoso (Eds.), Progress in Artificial Intelligence. Proceedings, 1997. XIV, 393 pages. 1997.
- Vol. 1325: Z.W. Raś, A. Skowron (Eds.), Foundations of Intelligent Systems. Proceedings, 1997. XI, 630 pages. 1997.
- Vol. 1328: C. Retoré (Ed.), Logical Aspects of Computational Linguistics. Proceedings, 1996. VIII, 435 pages. 1997.
- Vol. 1342: A. Sattar (Ed.), Advanced Topics in Artificial Intelligence. Proceedings, 1997. XVIII, 516 pages. 1997.
- Vol. 1348: S. Steel, R. Alami (Eds.), Recent Advances in AI Planning. Proceedings, 1997. IX, 454 pages. 1997.
- Vol. 1359: G. Antoniou, A.K. Ghose, M. Truszczyński (Eds.), Learning and Reasoning with Complex Representations. Proceedings, 1996. X, 283 pages. 1998.
- Vol. 1360: D. Wang (Ed.), Automated Deduction in Geometry. Proceedings, 1996. VII, 235 pages. 1998.
- Vol. 1365: M.P. Singh, A. Rao, M.J. Wooldridge (Eds.), Intelligent Agents IV. Proceedings, 1997. XII, 351 pages. 1998.
- Vol. 1371: I. Wachsmuth, M. Fröhlich (Eds.), Gesture and Sign Language in Human-Computer Interaction. Proceedings, 1997. XI, 309 pages. 1998.
- Vol. 1374: H. Bunt, R.-J. Beun, T. Borghuis (Eds.), Multimodal Human-Computer Communication. VIII, 345 pages. 1998.
- Vol. 1387: C. Lee Giles, M. Gori (Eds.), Adaptive Processing of Sequences and Data Structures. Proceedings, 1997. XII, 434 pages. 1998.
- Vol. 1394: X. Wu, R. Kotagiri, K.B. Korb (Eds.), Research and Development in Knowledge Discovery and Data Mining. Proceedings, 1998. XVI, 424 pages. 1998.
- Vol. 1395: H. Kitano (Ed.), RoboCup-97: Robot Soccer World Cup I. XIV, 520 pages. 1998.
- Vol. 1397: H. de Swart (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods. Proceedings, 1998. X, 325 pages. 1998.
- Vol. 1398: C. Nédellec, C. Rouveiro (Eds.), Machine Learning: ECML-98. Proceedings, 1998. XII, 420 pages. 1998.
- Vol. 1409: T. Schaub, The Automation of Reasoning with Incomplete Information. XI, 159 pages. 1998.

Lecture Notes in Computer Science

- Vol. 1367: E.W. Mayr, H.J. Prömel, A. Steger (Eds.), *Lectures on Proof Verification and Approximation Algorithms*. XII, 344 pages. 1998.
- Vol. 1368: Y. Masunaga, T. Katayama, M. Tsukamoto (Eds.), *Worldwide Computing and Its Applications — WWCA'98*. Proceedings, 1998. XIV, 473 pages. 1998.
- Vol. 1370: N.A. Streitz, S. Konomi, H.-J. Burkhardt (Eds.), *Cooperative Buildings*. Proceedings, 1998. XI, 267 pages. 1998.
- Vol. 1371: I. Wachsmuth, M. Fröhlich (Eds.), *Gesture and Sign Language in Human-Computer Interaction*. Proceedings, 1997. XI, 309 pages. 1998. (Subseries LNAI).
- Vol. 1372: S. Vaudenay (Ed.), *Fast Software Encryption*. Proceedings, 1998. VIII, 297 pages. 1998.
- Vol. 1373: M. Morvan, C. Meinel, D. Krob (Eds.), *STACS 98*. Proceedings, 1998. XV, 630 pages. 1998.
- Vol. 1374: H. Bunt, R.-J. Beun, T. Borghuis (Eds.), *Multimodal Human-Computer Communication*. VIII, 345 pages. 1998. (Subseries LNAI).
- Vol. 1375: R. D. Hersch, J. André, H. Brown (Eds.), *Electronic Publishing, Artistic Imaging, and Digital Typography*. Proceedings, 1998. XIII, 575 pages. 1998.
- Vol. 1376: F. Parisi Presicce (Ed.), *Recent Trends in Algebraic Development Techniques*. Proceedings, 1997. VIII, 435 pages. 1998.
- Vol. 1377: H.-J. Schek, F. Saltor, I. Ramos, G. Alonso (Eds.), *Advances in Database Technology — EDBT'98*. Proceedings, 1998. XII, 515 pages. 1998.
- Vol. 1378: M. Nivat (Ed.), *Foundations of Software Science and Computation Structures*. Proceedings, 1998. X, 289 pages. 1998.
- Vol. 1379: T. Nipkow (Ed.), *Rewriting Techniques and Applications*. Proceedings, 1998. X, 343 pages. 1998.
- Vol. 1380: C.L. Lucchesi, A.V. Moura (Eds.), *LATIN'98: Theoretical Informatics*. Proceedings, 1998. XI, 391 pages. 1998.
- Vol. 1381: C. Hankin (Ed.), *Programming Languages and Systems*. Proceedings, 1998. X, 283 pages. 1998.
- Vol. 1382: E. Astesiano (Ed.), *Fundamental Approaches to Software Engineering*. Proceedings, 1998. XII, 331 pages. 1998.
- Vol. 1383: K. Koskimies (Ed.), *Compiler Construction*. Proceedings, 1998. X, 309 pages. 1998.
- Vol. 1384: B. Steffen (Ed.), *Tools and Algorithms for the Construction and Analysis of Systems*. Proceedings, 1998. XIII, 457 pages. 1998.
- Vol. 1385: T. Margaria, B. Steffen, R. Rückert, J. Posegga (Eds.), *Services and Visualization*. Proceedings, 1997/1998. XII, 323 pages. 1998.
- Vol. 1386: T.A. Henzinger, S. Sastry (Eds.), *Hybrid Systems: Computation and Control*. Proceedings, 1998. VIII, 417 pages. 1998.
- Vol. 1387: C. Lee Giles, M. Gori (Eds.), *Adaptive Processing of Sequences and Data Structures*. Proceedings, 1997. XII, 434 pages. 1998. (Subseries LNAI).
- Vol. 1388: J. Rolim (Ed.), *Parallel and Distributed Processing*. Proceedings, 1998. XVII, 1168 pages. 1998.
- Vol. 1389: K. Tombre, A.K. Chhabra (Eds.), *Graphics Recognition*. Proceedings, 1997. XII, 421 pages. 1998.
- Vol. 1390: C. Scheideler, *Universal Routing Strategies for Interconnection Networks*. XVII, 234 pages. 1998.
- Vol. 1391: W. Banzhaf, R. Poli, M. Schoenauer, T.C. Fogarty (Eds.), *Genetic Programming*. Proceedings, 1998. X, 232 pages. 1998.
- Vol. 1392: A. Barth, M. Breu, A. Endres, A. de Kemp (Eds.), *Digital Libraries in Computer Science: The McDoc Approach*. VIII, 239 pages. 1998.
- Vol. 1393: D. Bert (Ed.), *B'98: Recent Advances in the Development and Use of the B Method*. Proceedings, 1998. VIII, 313 pages. 1998.
- Vol. 1394: X. Wu, R. Kotagiri, K.B. Korb (Eds.), *Research and Development in Knowledge Discovery and Data Mining*. Proceedings, 1998. XVI, 424 pages. 1998. (Subseries LNAI).
- Vol. 1395: H. Kitano (Ed.), *RoboCup-97: Robot Soccer World Cup I*. XIV, 520 pages. 1998. (Subseries LNAI).
- Vol. 1396: G. Davida, M. Mambo, E. Okamoto (Eds.), *Information Security*. Proceedings, 1997. XII, 357 pages. 1998.
- Vol. 1397: H. de Swart (Ed.), *Automated Reasoning with Analytic Tableaux and Related Methods*. Proceedings, 1998. X, 325 pages. 1998. (Subseries LNAI).
- Vol. 1398: C. Nédellec, C. Rouveiro (Eds.), *Machine Learning: ECML-98*. Proceedings, 1998. XII, 420 pages. 1998. (Subseries LNAI).
- Vol. 1399: O. Etzion, S. Jajodia, S. Sripada (Eds.), *Temporal Databases: Research and Practice*. X, 429 pages. 1998.
- Vol. 1401: P. Sloot, M. Bubak, B. Hertzberger (Eds.), *High-Performance Computing and Networking*. Proceedings, 1998. XX, 1309 pages. 1998.
- Vol. 1403: K. Nyberg (Ed.), *Advances in Cryptology — EUROCRYPT '98*. Proceedings, 1998. X, 607 pages. 1998.
- Vol. 1409: T. Schaub, *The Automation of Reasoning with Incomplete Information*. XI, 159 pages. 1998. (Subseries LNAI).