Action Selection in Cooperative Robot Soccer using Case-Based Reasoning

Tesi Doctoral

Autora

Raquel Ros Espinoza

Director
Ramon López de Màntaras

Tutor
Josep Puyol Gruart

Universitat Autònoma de Barcelona

Programa de Doctorat en Informàtica
Departament de Ciències de la Computació
Universitat Autònoma de Barcelona

Realitzada a
Institut d'Investigació en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas

Bellaterra, Febrer del 2008.

al Pepe y a la Mazo

Acknowledgments

Como se suele hacer en estos casos, tendría que comenzar agradeciendo a mi director de tesis, a Ramon. Y de hecho, lo haré. Pero no solo por haberme guiado y dado consejo en mi investigación durante estos últimos cuatro años, sino también por consentirme y permitirme el cambio de tema tesis el día que se lo propuse, lo cual además suponía la adquisición de nuevos robots. Todo hay que decirlo, y para suerte mía, a él también le hacían gracia estos juguetes, y no fue difícil convencerlo de este cambio, que finalmente se presenta hoy en esta tesis. Junto a él, y aunque a efectos burocráticos no figura formalmente, Josep Lluís ha sido mi segundo director. Sin duda alguna las discusiones y reuniones con él han sido de gran utilidad a la hora de desarrollar esta tesis, y es por ello que parte de este trabajo se lo debo a él. Continuaré agradeciendo el apoyo de otro "jefe", aunque un poco más lejano a la tesis, que me abrió las puertas del instituto cuando yo empezaba a descubrir el área de la inteligencia artificial. Carles, mi primer director en el mundo de la investigación, con quien tengo garantizado pasar buenos momentos charlando, riendo... y comiendo en la masía!

I sincerely thank Manuela Veloso, without whom this thesis wouldn't have even begun. She not only accepted to work with me proposing new investigation trends in my career, but always enthusiastically received me in her lab in several occasions giving me the oppotunity of living new experiencies out of the IIIA. Also thanks to the CMDash team, specially Sonia, Doug and Colin, with whom I'm happy to say that besides sharing research interests, we also share a close friendship. And before flying back to Spain to continue with the acknowledgments, I would like to thank Cindy Marling, for reviewing this dissertation, as well as for her helpful comments and suggestions.

Agradezco a la gente del IIIA en general, por los buenos momentos compartidos a la hora del café, las comidas, los viajes y congresos. Especialmente a amigos más cercanos, Eva, Maarten, Dani y Jordi, quienes me han ayudado de una manera u otra y gracias a quienes incluso los días más duros han sido más fáciles de superar. A mis amigos de la uni, Enric, Miquel y Pep, a "les nenes", Cris, Meri i Glor, a mi familia, Saray, Dídac (petit), Beto y Asun, y a todos los amigos que a pesar de no entender muy bien mi trabajo, han estado apoyándome en todo momento e intentando comprender lo que hago con unos perritos futboleros!

Evidentemente dejo al final a los "más importantes"! A Dídac, por acompañarme desde el primer día (nunca mejor dicho!) en este proyecto, no sólo a nivel profesional discutiendo ideas, programando y escribiendo papers, pero sobretodo apoyándome a nivel personal, en los buenos momentos y en los de

frustración, siempre con optimismo y confiando en mi capacidad para salir adelante. Finalmente, a los protagonistas de esta tesis, a mis robots: Nata, Boira, Fang y Terra, sin quienes, definitivamente, esta tesis no hubiera podido ser completada!

Raquel Ros holds a scholarship from the Generalitat de Catalunya Government. This work has been partially funded by the Spanish Ministry of Education and Science project MID-CBR (TIN2006-15140-C03-01) and by the Generalitat de Catalunya under the grant 2005-SGR-00093.

Abstract

Designing the decision-making engine of a team of robots is a challenging task, not only due to the complexity of the environment where the robots usually perform their task, which include uncertainty, dynamism and imprecision, but also because the coordination of the team must be included in this design. The robots must be aware of other robots' actions to cooperate and to successfully achieve their common goal. Besides, decisions must be made in real-time and with limited computational resources.

This thesis contributes a novel case-based approach for action selection and coordination in joint multi-robot tasks in real environments. This approach has been applied and evaluated in the representative domain of robot soccer, although the ideas presented are applicable to domains such as disaster rescue operations, exploration of unknown environments and underwater surveillance, among others.

The retrieval process proposes a case to reuse, evaluating the candidate cases through different measures to overcome the real world characteristics, including the adversarial component which is a key ingredient in the robot soccer domain. Unlike classical case-based reasoning engines, the case reuse consists in the execution of a set of actions through a team of robots. Therefore, from the multi-robot perspective, the system has to include a mechanism for deciding who does what and how. In this thesis, we propose a multi-robot architecture along with a coordination mechanism to address these issues.

We have validated the approach experimentally both in a simulated environment and with real robots. The results showed that our approach achieves the expected goals of the thesis, i.e. designing the behavior of a cooperative team of robots. Moreover, the experimentation also showed the advantages of using collaborative strategies in contrast to individualistic ones, where the adversarial component plays an important role.

Contents

Contents **ix**

1 Introduction **1**
1.1 Motivation and Overview . . . 3
1.2 Problem Domain . . . 7
1.3 Contributions . . . 10
1.4 Publications . . . 11
1.5 Outline of the Thesis . . . 11

2 CBR Preliminaries and Related Work **13**
2.1 Case Based Reasoning . . . 13
2.2 CBR Applied to RoboCup . . . 16
2.2.1 Karol et al. . . . 16
2.2.2 Lin, Chen and Liu . . . 16
2.2.3 Berger and Lämmel . . . 17
2.2.4 Wendler et. al . . . 17
2.2.5 Marling et al. . . . 18
2.2.6 Ahmadi et al. . . . 19
2.2.7 Steffens . . . 19
2.3 Other Models Applied to RoboCup . . . 20
2.3.1 Learning from Observation or Imitation . . . 20
2.3.2 Reinforcement Learning . . . 20
2.3.3 Pattern Recognition . . . 22
2.3.4 Fuzzy Theory . . . 22
2.3.5 Planning . . . 23
2.3.6 Neural Networks . . . 24
2.3.7 Evolutionary Algorithms . . . 25
2.3.8 Other Approaches . . . 26
2.4 CBR Applied to Other Robotic-Related Domains . . . 27
2.5 Summary . . . 29

3 The Retrieval Step **33**
3.1 Case Definition . . . 33
3.1.1 Problem Description . . . 33
3.1.2 Solution Description . . . 35
3.1.3 Case Scope Representation . . . 37
3.1.4 Case Example . . . 38
3.2 Case Base Description . . . 39

3.3 Case Retrieval 41
3.3.1 Similarity Measure 41
3.3.2 Cost Measure 44
3.3.3 Case Applicability Measure 47
3.3.4 Case Filtering 50
3.3.5 Experiments 53
3.4 Conclusions and Future Work 59

4 Case Reuse through a Multi-Robot System 61
4.1 Robot Architecture 61
4.1.1 Deliberative System 63
4.1.2 Executive System 64
4.2 Multi-Robot System and Case Reuse 65
4.3 Conclusions and Future Work 69

5 Learning the Scopes of Cases 71
5.1 Scope Adaptation Algorithm 72
5.1.1 When to Adjust the Values 73
5.1.2 How to Adjust the Values 74
5.1.3 Example 76
5.2 Acquiring New Cases 78
5.3 Experiments 78
5.3.1 Simulation Experiments 78
5.3.2 Real World Experiments 83
5.4 Conclusions and Future Work 88

6 Experimentation 89
6.1 CBR System Settings 90
6.2 Experiments Setup 95
6.2.1 Robot's Behaviors 95
6.2.2 The Scenarios 96
6.2.3 Evaluation Measures 98
6.3 Simulation Experiments 98
6.3.1 The Simulator 98
6.3.2 Simulation Results 99
6.4 Real Robot Experiments 104
6.4.1 The Robots 104
6.4.2 Results 106
6.5 A Trial Example 112
6.6 Discussion and Future Work 124

7 Conclusions and Future Work 127
7.1 Summary of the Contributions 127
7.2 Future Directions 129

Bibliography 133

List of Figures

1.1 Snapshot of the Four-Legged League field 8

2.1 The Case-Based Reasoning cycle 15

3.1 Example of a problem description 35
3.2 Example of the scope of a case 37
3.3 Example of a case 39
3.4 Example of symmetric cases 40
3.5 2D Gaussian function 42
3.6 Strategy function combining time and score difference 43
3.7 Adapted positions example 45
3.8 Trapezoid layout of two matching pairs 46
3.9 Correspondence examples. 47
3.10 Ball's trajectory representation 49
3.11 Example of the ball's path and the opponents similarity computation 50
3.12 Sorting experimentation scenarios 54
3.13 Trials sorted by frequency of cases retrieved 55
3.14 Case frequency for each scenario 57

4.1 Robot architecture. 62
4.2 Multi-robot system 65
4.3 FSM for retrievers and executors 66
4.4 FSM for the case execution 67
4.5 Kick adaptation during the case reuse 68

5.1 Scope of a case: security region and risk region 73
5.2 Increasing policies for the scope learning process 75
5.3 Case scope evolution. 77
5.4 Learning the scopes parameters in simulation 80
5.5 Learning evolution of the scope in simulation. 81
5.6 Convergence of the average value of τ 82
5.7 Comparing strategies with real robots. 84
5.8 Training a case base with real robots 87

6.1 Case Base: single cases. 91
6.2 Case Base: multiple cases. 92
6.3 Case examples 94

6.4 Action region for the defender and the goalie 96
6.5 Scenarios used during the experimentation 97
6.6 Snapshot of the robot soccer simulator PuppySim 2 99
6.7 Ball classification outcome: scenario 1 and 2 101
6.8 Ball classification outcome: scenario 3 and 4 102
6.9 Sony AIBO ERS-7(M2-M3) robot description. 104
6.10 The robots 105
6.11 Images extracted from the robot vision system 106
6.12 CBR approach sketch performance in scenario 3 108
6.13 Reactive approach sketch performance in scenario 3 109
6.14 CBR approach sketch performance in scenario 4 111
6.15 Reactive approach sketch performance in scenario 4 112
6.16 Trial example: first case 114
6.17 Trial example: first case (sequence 1 and 2) 115
6.18 Trial example: first case (sequence 3 and 4) 116
6.19 Trial example: second case 118
6.20 Trial example: second case (sequence 1 and 2) 119
6.21 Trial example: second case (sequence 3 and 4) 120
6.22 Trial example: third case 121
6.23 Trial example: third case (sequence 1 and 2) 122
6.24 Trial example: third case (sequence 3 and 4) 123

List of Tables

2.1 Table of abbreviations. 29
2.2 Related work classification. 30

3.1 List of available actions and their parameters. 36
3.2 List of spatial transformations. 40
3.3 Number of cases and average time per experiment 56

5.1 Strategies for the real world experiments. 85

6.1 Ball outcome classification (simulation). 100
6.2 Defender's ball possesion (simulation). 100
6.3 Backing up results 103
6.4 Single vs. multiple cases results 103
6.5 Ball outcome classification (real robots). 110
6.6 Defender's ball possesion (real robots). 110

Chapter 1

Introduction

He turned once more to the robot. "Get up!"
The robot towered upward slowly and Donovan's head craned and his puckered lips whistled.
Powell said: "Can you go out upon the surface? In the light?"
There was consideration while the robot's slow brain worked. Then, "Yes, Master."
"Good. Do you know what a mile is?"
Another consideration, and another slow answer. "Yes, Master."
"We will take you up to the surface then, and indicate a direction. You will go about seventeen miles, and somewhere in that general region you will meet another robot, smaller than yourself. You understand so far?"
"Yes, Master."
"You will find this robot and order him to return. If he does not wish to, you are to bring him back by force."

Extracted from the short story Runaround, *in Asimov's* I, Robot *[7].*

"Now on four occasions recently," Powell said, "your boss deviated from brain-scheme. Do you remember those occasions?"[...]
Powell turned back to the robot, "What were you doing each time... I mean the whole group."[...]
He said [the robot], "The first time we were at work on a difficult outcropping in Tunnel 17, Level B. The second time we were buttressing the roof against a possible cave-in. The third time we were preparing accurate blasts in order to tunnel farther without breaking into a subterranean fissure. The fourth time was just after a minor cave-in."
"What happened at these times?"
"It is difficult to describe. An order would be issued, but before we could receive and interpret it, a new order came to march in queer formation."

Extracted from the short story Catch that Rabbit, *in Asimov's* I, Robot *[7].*

Perhaps the excerpts shown previously describe simple and irrelevant scenes that usually would not attract much of our attention. Moreover, probably the only curious event that would even draw a slight smile in the reader's face is the fact that a human communicates with a robot through natural language, i.e. talking, while the robot not only understands the conversation, but also replies in the same way. However, this is not our focus of interest, although it is in indeed a very challenging task that researchers in artificial intelligence (AI) are still working on.

The short story from where the first text is extracted (Runaround) takes place in Mercury and is about a robot, SPD-13 ("Speedy"), that is sent to bring selenium from the nearest selenium pool, 17 miles away from the base station. However, after five hours of having departed, Speedy has not returned yet. Therefore, Donovan and Powell decide to send another robot to get him and to analyze what happened. At this point, the conversation shown in the text takes place. Powell orders the robot what seems a simple task. Although the story actually continues without the robot having to execute the task, we are interested in analyzing the consequences of this "simple" task. Let us first assume that somehow the robot understands what the task commanded by Powell is about. Some of the abilities the robot must have to perform this "simple" task are: ability to **perceive** its environment in order to create its internal world model; ability to build a map (if the robot does not have it a priori) of the environment and **localize** itself and the place it has to go within the map; ability to **plan** a route to the goal location and then to come back; ability to **navigate** through the environment, probably avoiding obstacles, conflicting paths, etc.; ability to **recognize** another robot; ability to **decide** how to perform the task, i.e. which actions to execute; ability to **react** and **recover** upon possible conflicts it could encounter during the execution of the task. Hence, what seemed a simple task turned out to be a more complex one, requiring a set of abilities where each of them leads to a broad range of challenging problems that different fields in AI have addressed since their origins in the 1950's.

Besides the above mentioned abilities we expect a robot to be programmed with, we can find a last interesting component within the second text. In this occasion (Catch that Rabbit), Powell and Donovan have to discover why the robot DV-5 (Dave) fails executing the task it is commanded to perform. The robot's peculiarity is that it is a robot with six subsidiary robots which are controlled through positronic fields. In other words, Dave can be seen as a coordinator robot with six "worker" robots under its responsability which perform the tasks commanded by the coordinator. Back to the story, Powell and Donovan decide to spy while the robots work to discover why the task is not being correctly fulfilled. They discover that in general the robots are working the right way until something unexpected happens and they start marching and dancing leaving the task aside. To understand why they behave that way, Powell asks one of the subsidiary robots what is going on (extracted text). At this point the robot relates the tasks they were assigned to do. Two ingredients in this story draw our attention. First, the story is related to a **team of robots**, and second, the tasks to perform cannot be accomplished individually, but through **teamwork**. In order to fulfill the tasks, a coordinator is in charge of the team, sending the commands to the team and supervising the task execution. When dealing with **cooperative tasks** where a group of robots have to achieve a joint task, some of the challenges, besides the ones mentioned before

of course, consist in answering the following questions: who decides what to do?, i.e. a single robot decides which actions to perform (centralized system) or all robots discuss the selected actions (distributed system); who does what?, i.e. one robot is selected to perform the complete task (single task execution) or each robot may perform part of the task or subtasks (distributed task execution); who monitors the task execution?, i.e. one robot receives all the information from the rest of the robots and decides by itself (single monitoring) or each robot has its own beliefs of the world and reacts accordingly (distributed monitoring); and finally, which **coordination mechanism** to employ in order to synchronize the robots' actions?

Thus, from what seemed to be two independent excerpts of not much interest, we have remarked a set of problems that probably, from our human point of view, have obvious solutions (we face them in our daily life without even noticing their difficulty). However, from an AI researcher perspective, when designing the robot behavior, these problems are not trivial at all, and in fact, result in big challenges for AI nowadays.

1.1 Motivation and Overview

The dissertation presented in this work is addressed to two of the presented challenges. First, the decision-making for the action selection problem and second, the incorporation of a coordination mechanism to achieve cooperative tasks within a multi-robot system. We next overview the main problems which we have to deal with and how we propose to solve them.

An important aspect to consider when designing the reasoning engine is the type of environment where the robot performs its task. The difficulties that arise within deterministic environments (controlled environments) are far much easier to deal with that when dealing with stochastic environments (uncontrolled). Clearly the latter is much more interesting, and is the one this dissertation addresses. In such environments, where the world continuously changes out of our control, the reasoning engine must include mechanisms to overcome imprecision and dynamism of the environment. More precisely, it has to be able to react and recover from unexpected situations that may occur during the performance of the task where a real-time response is fundamental.

The behavior of a robot results from the execution of actions for different states, if we define acting as the execution of a policy $\pi : s \rightarrow a$ (where s is the current state and a, the action to execute in the given state) [44]. Defining each possible state and the actions to perform at each state, i.e. defining the policy, is challenging and tedious to be done completely manually. This policy is one of the fundamental parts of the robot's reasoning engine. Therefore, it is crucial to find a way for automatically and efficiently acquiring it. As we review further on, several machine learning techniques have been proposed during the past years.

Besides the difficulties emerged due to the nature of the environment, we must also take into account the limitations of the robot performing the task. Thus, the uncertainty of the robot's internal beliefs of the world depends on the accuracy of the robot's sensors. The reasoning engine must be able to handle uncertainty so the behavior of the robot does not result degraded. A last important aspect to consider are the robot's computational resources. The processor

determines the type of algorithms (in terms of complexity) that the reasoning engine may use.

From a multi-agent perspective, the problem we address in this work is related to cooperation or collaboration[1] among agents. Collaboration is desired in several domains where a group of robots (also seen as agents) work together to achieve a common goal. It is not only important to have the agents collaborate, but also to do it in a coordinated manner so the task can be organized to obtain effective results. Providing the agents with capacities to collaborate and to coordinate is complex, as it is not just a matter of dividing the tasks and assigning roles to each agent. Instead, it is also a matter of beliefs and commitments of all robots to fulfill a common task. Drogoul and Collinot [14] distinguish three levels of behaviors when designing a multi-agent system:

- elementary behaviors, actions or functions that the agents individually perform (what to do);
- relational behaviors, how agents interact with other agents and the influences of their elementary actions on the other agents (what to do in presence of other agents); and,
- organizational behaviors, how the agents can manage their interactions to stay organized (what to do with these agents).

Similarly, Grosz and Kraus [19] argue that collaborating agents must

- establish mutual beliefs on what actions they will perform to complete the task (relational level);
- agree on who does what (organizational level); and,
- establish mutual beliefs of their individual intentions to act (relational level).

Communication among agents is essential to achieve these requirements.

The robot soccer domain is a very challenging test-bed that incorporates most of the problems enumerated so far. Hence, we deal with a highly dynamic environment that requires real-time response. Robots' sensors are not very accurate and therefore, we must model uncertainty within the reasoning engine to act accordingly. Robots' actions performances are imprecise and recovery mechanisms should be considered. Finally, computational resources are very limited, and thus, simple processes have to be taken into account. In this dissertation we contribute with an approach for action selection and coordination for joint multi-robot tasks. More precisely, we apply Case-Based Reasoning (CBR) techniques to model the reasoning engine and its application in the robot soccer domain. Case-based reasoning is an approach to problem solving that emphasizes the role of prior experience during future problems solving [39]. It has been inspired by the way humans reason and use their memory of previous experiences to solve new problems. An example directly related with the work presented in this dissertation can be found in team sports. During training the coach studies with the players different game situations and the according

[1] Through the dissertation we will refer to both concepts, cooperation or collaboration, as synonyms, although the latter can also be related to "working with the enemy", including a traitorous sense.

movements (gameplays) that the players should perform. The *playbook* corresponds to the case base in the CBR system. During a game, when the players detect a similar configuration between the current situation in the field and the ones in the playbook (CBR's retrieval step), they automatically reproduce the gameplays reviewed during the training, performing certain adaptations if necessary (CBR's reuse step).

The approach models the state of the game at a given time as a problem description, which is compared to a set of predefined situations, called cases. A case describes the state of the environment (problem description) and the actions to perform in that state (solution description). The retrieval process consists in obtaining the most similar case. Next, the solution of the retrieved case is reused after some adaptation process, as required. We model the case solution as a set of sequences of actions, *gameplays*, which indicate what actions should each robot perform. Finally, we specify a multi-robot architecture and a coordination mechanism based on messages exchanged among the robots in order to achieve a cooperative behavior.

Why Case-Based Reasoning?

The first question that can arise when reading this work is *why CBR?* As we review in Chapter 2, different approaches to solve the action selection problem have been presented through the past years (Reinforcement Learning, Fuzzy Theory, Decision Trees, Neural Networks, etc.) obtaining successful results. However, we believe that Case-Based Reasoning integrates fundamental properties that not only help the designer in building a reasoning engine, but also result very intuitive for humans since it is tightly related to the way humans reason.

From the implementation point of view, we can classify the design of a robot behavior from a procedural implementation, where the behavior is composed of a sequence of subroutines and evaluating conditions (low level approach), to a model-based implementation, where the knowledge representation is done through state-action models and the task to learn is the mapping function between states and actions (high level approach). Although the high level approaches have several advantages over procedural implementations, low level approaches are still being widely used for designing robot behaviors, specially in very specific scenarios or in competition in the case of robot soccer (RoboCup). A common approach used within RoboCup to describe behaviors are hierarchical finite state machines (FSMs) [60, 67, 53] to provide a certain degree of abstraction level. The advantage of using this approach is probably due to its high reactivity. The robot is able to rapidly switch from one behavior to another when required. However, programming individual or complex behaviors is still tedious and slow. As argued in [46, 57] changes are complicated due to interdependencies and large amount of parameters to consider when programming the behaviors. A minor modification in the code of a behavior can have a big impact on other behaviors. Another important drawback within reactive approaches is that from a strategic point of view we can classify them as "short-sighted", meaning that their decision-making is usually driven by a partial state of the environment where the actions take place, without having a broader view of the world state. Thus, an action can be suitable for a given moment in time, but probably another action would have been a better choice

if the whole world state or possible future states could have been considered or predicted.

Regarding model-based approaches, we can classify them according to the policy *readability*, i.e. how understandable for a human reader the learned policy is. Techniques as reinforcement learning (RL), neural networks (NN) or evolutionary algorithms (EA) have proved to be useful in many domains, including action selection in robotics. However, their main drawback is that the learned policy cannot be manually followed by an expert, and thus, analyzing why a certain action has been selected is not feasible. As other researchers have previously remarked [18, 46, 31, 10, 38, 15], we believe that this is an important property to consider when choosing among the available approaches, specially within complex domains, where some kind of justification is necessary for evaluating the appropriateness of the selected actions. On the other hand, the advantage of other approaches such as decision trees, expert systems, fuzzy rules (rule-based approaches in general) or case-base reasoning (instance-based approaches) is that their knowledge representation is readable from the expert perspective, not only facilitating the comprehension of the policy, but also providing easy access to modify the current knowledge of the reasoning engine.

Another important component to consider is the time required for learning the policy and the amount of training data to achieve an acceptable accuracy level. From the above mentioned approaches, RL, NN, EA and decision trees either require a large amount of training data or time or both, which are usually not available within the robotics field.

Finally, and not less important, a on-line learning ability is desired for this kind of domains where the robot may encounter unexpected situations that where not considered during the design stage. With this last component, the adaptability of the robot's behavior is guaranteed, allowing it to acquire new knowledge as it performs the tasks. Rule-based approaches lack of this last component. The only way for introducing new knowledge is manually modifying the rule set. Other approaches, such as NN, EA or decision trees need to repeat the training process before using the new learned policy. Modifying the current knowledge of the system is time consuming and requires new training data.

After reviewing the desired properties of the approach used for the action selection problem, we conclude that Case-Based Reasoning (an instance-based approach) fulfills the requirements described. The case base contains the knowledge representation of the reasoning engine, which in fact, corresponds to a set of situations (cases) the robot encounters through the task execution. Each case may represent a complete or partial description of the state of the environment and the corresponding solution to that state, i.e. the actions to execute. Cases can either be generalized or specialized allowing the expert to gradually introduce knowledge as needed. The knowledge of the system is "transparent" and the expert can easily modify or insert new knowledge without spending time training the reasoning engine again.

Regarding cooperation and teamwork, several works have been presented so far, either using more formal methodologies as the *joint intention* theory introduced by Cohen and Levesque [11] in Tambe's *flexible team work* [62], or simpler mechanisms such as role assignment where cooperation usually results as an emerging property [57, 65, 74, 35, 15], or including explicit coordination mechanisms through communication to enforce commitment among

the involved agents (request-acknowledge type) [66, 3, 18]. Interestingly, in our work the use of cases also allows us to easily model cooperative tasks. As mentioned before, in order to have agents performing joint tasks it is fundamental that: first, all agents agree on the task to perform; second, the implied agents commit to execute the task as planned; and third, these agents must be aware of the actions each of them performs to synchronize. In this work we specify a coordination mechanism that takes place during both the retrieval and the reuse steps based on messages exchanged among the robots about their internal states (beliefs and intentions). Hence, reviewing the requirements by Grosz and Kraus mentioned before, the combination of the case structure and the coordination mechanism we propose ensures that: the solution description indicates the actions the robots should perform (requirement *i*); the retrieval process allocates robots to actions (requirement *ii*); and finally, with the coordination mechanism, the robots share their individual intentions to act (requirement *iii*).

In conclusion, we believe that using CBR techniques is appropriate, not only due to the close relation with the way humans reason, but also because it provides a high level abstraction of the problem to solve through a modular methodology. This latter allows the expert to easily modify the robots' behavior as required, either introducing or replacing cases in the case base (knowledge of the system), defining new similarity functions, altering the retrieval process, etc. CBR is a very flexible and intuitive framework and thus, is suitable for the problem domain this dissertation is focused on.

1.2 Problem Domain

The problem domain where this thesis is applied to is robot soccer. RoboCup is a well known competition [69] whose ultimate goal is to develop a team of humanoid robots to play soccer against the human world champion team by 2050. Of course this is long term goal, but in fact, the main objective of designing this test-bed is to foster AI and robotics within a very complex domain and to motivate researchers of different fields to work together in order to achieve a common goal.

To this end RoboCup offers several leagues where, altough the goal is the same, the challenges differ. Currently we can find the following leagues: Simulation, Small size, Middle size, Standard Platform and Humanoid. The Standard Platform League is a new league that will start next year (2008) replacing the Four-Legged League[2]. Within this league all teams use the same robot, so the challenge is focused on developing the software for the robots, and not the physical robot contrarily to other leagues.

Within the The Four-Legged League teams consist of four Sony AIBO robots which operate fully autonomously, i.e. there is no external control, neither by humans nor by computers. Communication among robots of the same team is allowed through wireless or speakers and microphones (although the last ones are not usually used). There are two teams in a game: a red team and a blue team. The field dimensions are 6m long and 4m wide and represents a Cartesian plane as shown in Figure 1.1. There are two goals (cyan and yellow) and

[2]The robots for this league were the AIBO robots from Sony. Since Sony stopped manufacturing the robots, the RoboCup organizers had to switch to another model, the humanoid Aldebaran Nao.

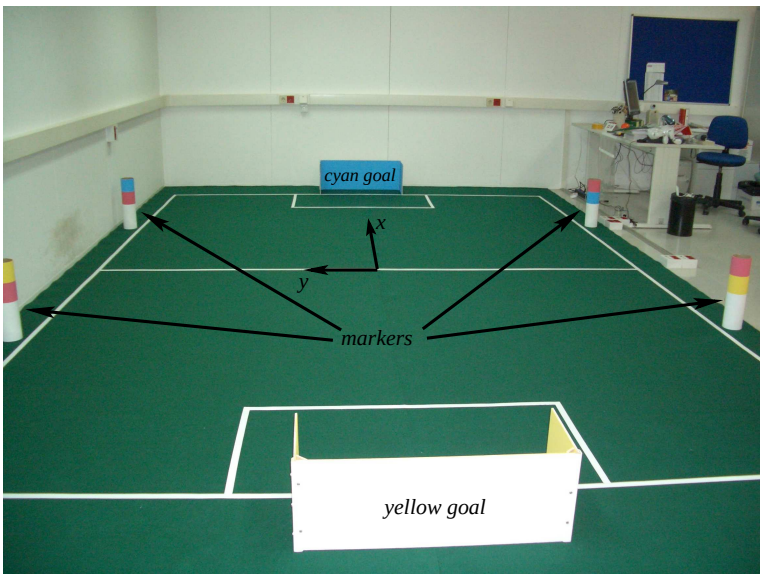


Figure 1.1: Snapshot of the Four-Legged League field (image extracted from the IIIA lab).

four colored markers the robots use to localize themselves on the field. A game consists of three parts, i. e. the first half, a half-time break, and the second half. Each half is 10 minutes. The teams change the goal defended and color of the team markers during the half-time break. At any point of the game, if the score difference is greater than 10 points the game ends. There is also an external controller, the *GameController*, which sends messages to the robots in order to stop or resume the game after a goal, to notify penalized robots, to start or end the game, etc. For more details on the official rules of the game refer the RoboCup Four-Legged League Rule Book [12].

A Brief History of RoboCup

Extracted from the RoboCup Official website (Overview) [1].

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.

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 investigations 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, they 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, they received overwhelming reactions from researchers outside of Japan, requesting that the initiative be extended as an international joint project. Accordingly, they 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 web. 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 at a large scale. The decision was made to provide two years of preparation and development time, so that initial groups 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 the International Conference on Intelligence Robotics and Systems (IROS-96), Osaka, from November 4–8, 1996, with eight teams competing in a simulation league and demonstration of real robot for the middle size league. While limited in scale, this competition was the first competition using soccer games for promotion of research and education.

The first official RoboCup games and conference was held in 1997 with

great success. Over 40 teams participated (real and simulation combined), and over 5,000 spectators attended.

1.3 Contributions

This thesis contributes a novel case-based approach for action selection and coordination in joint multi-robot tasks. This approach is applied and evaluated in the representative domain of robot soccer.

The main characteristics of the approach can be summarized as follows:

- The case definition corresponds to a complete description of the environment, including the actions to perform by a team of robots and general domain knowledge to handle uncertainty in the incoming information from perception.

- Two types of features are introduced: controllable and non-controlable features. The former ones are related to those features whose values can be directly modified in order to increase the similarity between the evaluated case and the current problem; while the latter ones, correspond to those features that the system cannot modify.

- The retrieval step is composed of three measures: the aggregation of domain-dependent similarity measures; the cost of adapting the current problem to a case; and the applicability evaluation of a case combining domain knowledge rules and similarity measures. The retrieval step applies a filtering mechanism to reduce the search space as fast as possible due to the real-time response requirements.

- The internal robot architecture is defined as a three-layer hybrid architecture: the deliberative system, i.e. the case-based reasoning engine; the reactive system, i.e. a set of behaviors corresponding to skills the robot performs; and the low level, which includes the sensors and executors of the robot.

- The multi-robot architecture includes a set of robots called *retrievers* that incorporate the reasoning engine and therefore are in charge of deciding the cases to reuse, and the *executors*, who only perform the actions indicated by the retrievers (or default actions). However, any robot has the ability to abort the execution of a task when required.

- A coordination mechanism that enables the case reuse not through a single user, but through a team of users (in this case, the robots).

- A supervised learning process to acquire the scope of a case automatically.

Finally, in this dissertation we present empirical evaluation both in a simulated environment and in a real one with robots to prove the effectiveness of the proposed approach. Moreover, we argue that a collaborative behavior is advantageous to achieve the goal of the task, specially because of the adversarial component. It is well known that a good strategy to avoid an opponent

during a game is to have passes between teammates. In contrast, using an individual strategy, where only one robot moves with the ball without taking into account its teammates, increases the chances for the opponent to block the attack, unless the robot is much faster than the opponent. Therefore, we have successfully included the pass action in our approach, which is not common, as far as we know, in this domain (Four-Legged League).

1.4 Publications

The following publications have been derived from this thesis:

- R. Ros, M. Veloso, R. López de Màntaras, C. Sierra and J.L. Arcos (2006), Retrieving and Reusing Game Plays for Robot Soccer. 8th European Conference on Case-Based Reasoning. *Advances in Case-Based Reasoning* of Lecture Notes in Computer Science, Volume 4106, pp. 47–61. Springer. ***Best paper award***.
- R. Ros, J.L. Arcos (2007). Acquiring a Robust Case Base for the Robot Soccer Domain. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1029–1034. AAAI Press.
- R. Ros, M. Veloso (2007). Executing Multi-Robot Cases through a Single Coordinator. In *Proceedings of the 6th International Conference on Autonomous Agents and Multiagent Systems*, E. H. Durfee, M. Yokoo eds., pp. 1264–1266.
- R. Ros, R. López de Màntaras, J.L. Arcos and M. Veloso (2007). Team Playing Behavior in Robot Soccer: A Case-Based Approach. In *Proceedings of the 7th International Conference on Case-Based Reasoning*. Case-Based Reasoning Research and Development of Lecture Notes in Computer Science, Volume 4626, pp. 46–60, Springer.
- R. Ros, M. Veloso, R. López de Màntaras, C. Sierra and J.L. Arcos (2007). Beyond Individualism: Modeling Team Playing Behavior in Robot Soccer through Case-Based Reasoning. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pp. 1671–1674. AAAI Press.

1.5 Outline of the Thesis

Next, we summarize the contents of Chapters 2 to 7. The core of the research work is described in Chapters 3 to 6.

Chapter 2: CBR Preliminaries and Related Work.

In this chapter we first review basic ideas of Case-Based Reasoning to familiarize the reader with the concepts used through the dissertation. Next, we present related work that describes the different techniques (including CBR) used by researchers in the past years within the robot soccer domain. A brief section is addressed to other robotic domains where CBR has been successfully applied. Finally, the chapter concludes with a summary of the related work through a comparative table and where our work is located with respect to previous work.

Chapter 3: Modeling the CBR Approach: The Retrieval Step

This chapter corresponds to the first step of the CBR cycle, i.e. the Retrieval Step. Thus, we present the different components of the proposed CBR system, including: the case description, the case base structure, the similarity measures and the retrieval process itself. We also present experimental results in simulation to test the introduced process.

Chapter 4: Case Reuse through a Multi-Robot System

In this work the case reuse is fulfilled through a team of robots, instead of an individual robot. Hence, we not only have to define the internal robot architecture, but also the multi-robot architecture. In this chapter, we describe how the robots interact to perform the task, i.e. how to reuse the case in a coordinated way.

Chapter 5: Learning the Scopes of Cases

A first attempt towards the learning stage of the CBR cycle is presented in this chapter. More precisely, it is focused on automatically acquiring the scope of a case through a supervised learning algorithm. Different functions used in the algorithm are proposed to this end. The learning mechanism is evaluated both in simulation and with real robots.

Chapter 6: Experimentation

This chapter is devoted to the experimentation stage. To evaluate the overall system, we have performed experiments in simulation and with the real robots. The scenarios consist of two vs. two games, where two attackers play against a defender and a goalie. The CBR approach is compared with respect to a region-based approach. While the attackers use both approaches for evaluation, the opponents use a fixed behavior. Results are discussed and a trial example with real robots is described in detail.

Chapter 7: Conclusions and Future Work

In this last chapter, we summarize the conclusions addressed in each separate chapter. We also discuss future research lines and open challenges to improve the proposed approach.

Chapter 2

CBR Preliminaries and Related Work

In this chapter we review related work to the one presented in this thesis. First we briefly overview the Case-Based Reasoning methodology so the reader is familiar with the concepts referred to afterwards. Next we describe applications of CBR systems within the robot soccer domain and other machine learning approaches that address the action-selection problem as well. Finally, a short review to other CBR systems applied to related domains and a summary of the reviewed work are presented.

2.1 Case Based Reasoning

Inspired by the cognitive science research in human reasoning and the use of memory [56], Case-Based Reasoning is the process of problem solving based on the exploitation of past experiences, called *cases*, to propose solutions for present problems [39]. The essence of Case-Based Reasoning is based on the assumption that "similar problems have similar solutions".

This lazy learning technique consists in comparing the new problem to solve with respect to past cases in the case library through a similarity measure. The most similar case (or set of similar cases) is retrieved in order to reproduce the solution proposed in the past, probably adapting it to the current problem to solve. The outcome of the solution is then evaluated and the new solved problem may be stored as a new case. Many applications have been proposed since the birth of this methodology, ranging from classical systems such as CHEF, CASEY, JULIA, HYPO, etc. [30] to contemporary systems dealing with more complex domains as we review in the following sections. Although initially case-based reasoning could probably be seen as a supervised learning technique for classification, through the past years it has shown its evolution towards new paradigms and directions increasing the utility of CBR systems [16].

The knowledge representation of a CBR system is the case library (or case base). In contrast to general knowledge (such as rule-based methods), cases represent specific knowledge related to specific situations. Hence, a case is

usually represented by the pair problem-solution, where the problem corresponds to the description of the task to solve, and the solution describes how this task was carried out. A third component can be attached to a case: the outcome. It corresponds to the resulting state of the world once the solution has been applied. This latter component is usually used to guide the reasoner system the next time the case is retrieved.

The simplest and most commonly used problem representation is a set of attribute-value pairs, although more complex representations can be used. The solution description may include the solution itself (values of features), a set of reasoning steps, justifications for decisions made during problem solving, etc. Finally, the outcome may be whether the applied solution was a success or a failure when solving the problem, whether it fulfilled the expectations or not, explanations of the failure, pointer to next attempt solution, etc.

Although different modifications of the CBR methodology can be found in the literature, their differences are basically based on the names or labels and possible extensions of the different steps of the process. However, the main concepts remain unaltered. One of the most accepted problem solving process is the one introduced by Aamodt and Plaza [2], the well know "4 RE's" cycle. The four main steps of the cycle are (Figure 2.1):

- Retrieve: search the case library for cases that are similar to the current problem, based on a similarity measure and obtain candidate solutions.
- Reuse: construct a solution for the current problem based on the solutions proposed by the retrieved cases (usually adapting or merging solutions).
- Revise: evaluate the outcome of applying the reused solution and repair the solution constructed above, if necessary.
- Retain: decide whether the reused case should be incorporated into the case library or not.

Given a new problem to solve, the system applies a similarity measure to obtain the most similar cases. The similarity measure depends on the problem description. Thus, the simplest metric usually corresponds to the distance between two features. In this case, the retrieval corresponds to a *k-nearest neighbor* algorithm. More complex measures can be defined, depending on the domain and on design preferences. Filtering mechanisms can be used, as well as aggregations of different measures, static or dynamic procedures, comparing complete cases or partial descriptions, and so on.

The reuse step consists in building the solution of the current problem to solve using the solution description(s) of the retrieved case(s). Based on the domain requirements, the solution can be straightforward, i.e. reusing the same solution without previous processing, or through some adaptation process. Typical adaptation methods are parameter adjustment, local search, substitution and merging processes, among others [30]. As within the retrieval step, the adaptation can be addressed to the whole solution description, or part of it.

After the solution is carried out, the next step is to evaluate the effectiveness of the proposed solution. In most systems this step is usually driven by the user of the system (analogous to supervised learning technique) who indicates the outcome of the task execution. As mentioned before, not only success

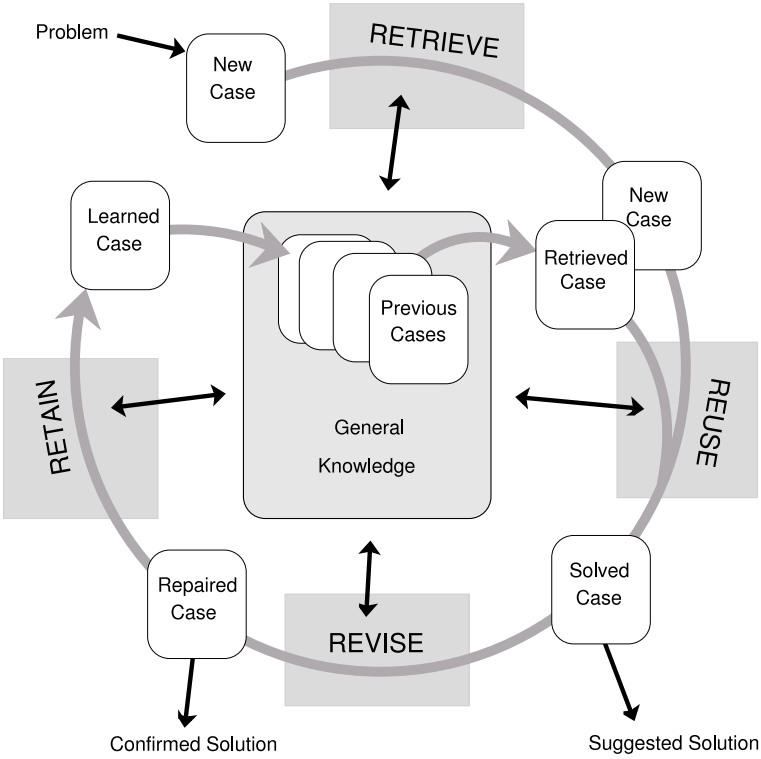


Figure 2.1: The Case-Based Reasoning cycle (Aamodt and Plaza [2]).

or failure can be indicated, but also the reasons of failure or whatever additional useful information for improving the quality of the proposed solution in future situations. Clearly, it would be much more desirable that the system could automatically generate the evaluation of the solution. However, in most domains, this is still a pending task that researchers have to address and which remains as an open challenge so far.

Finally, the new solved problem can be stored in the case library. This latter step is responsible for the learning aspect of the CBR methodology. Thus, a new case is created including the initial problem to solve, the solution proposed and the outcome (if available). Deciding whether a case should be stored or not basically depends on how useful the new case will be in the future. Two aspects must be considered when opting for case retention: case indexing and case base maintenance. If new cases are introduced, the size of the case base will increase through time. As a consequence, the search space during retrieval is also increased. Therefore, case indexing techniques must be considered to speed up the retrieve step. The second aspect, case maintenance, is related to analyzing the case base in order to determine which cases can be removed, due to case redundancy or inconsistency.

The minimal components of a case-based reasoning system are the retrieve and the reuse steps, i.e. generating a solution for a given problem. The two remaining ones are related to the learning process of the system, and therefore, are the most difficult to introduce.

2.2 CBR Applied to RoboCup

In this section we review the research done so far using Case-Based Reasoning techniques within the RoboCup domain. For each author we briefly detail the purpose of the CBR approach, the league in which it is applied to and a general description of the features of the system.

2.2.1 Karol et al.

Very close to our work, Karol et al. [25] present an initial attempt for including a CBR system in the action selection of a team of robots in the Four-Legged League. The problem description includes the robots' positions and the degree of ball possession (qualitative measure). They also propose the use of meta-level features to guide the retrieval process, such as score, time and opponent strategy (if known). As within our work, the solution corresponds to the game-play. They propose three possible similarity measures, all based on comparing the robots positions on the field. Two of them are quantitative (based on the distances the robots would have to travel to be positioned as in the case) and one qualitative. This latter divides the field in uniform rectangular regions, and the measure counts the number of steps the robots would have to move towards the positions indicated in the case. The remaining features of the problem description are not taken into account yet. Since the work is only a first introduction of the model, no experiments have been reported.

2.2.2 Lin, Chen and Liu

The work presented by Lin et al. [38] and Chen and Liu [10] is applied in the Simulation League, where they present a hybrid architecture for soccer players (as in our work). The deliberative layer corresponds to the CBR system and the reactive layer corresponds to fuzzy behaviors (motor schemas introduced by Arkin [6]). The knowledge acquisition is done through first order predicate logic (FOPL), which they claim is easy for an expert to transmit knowledge.

The problem description of a case consists of fuzzy features describing distances and directions between robots and objects on the field (such as ball, goal, etc.). The solution description corresponds to a set of gain values of the motor schemas. Similar to our work, they introduce the concept of *escape conditions*: a new case is retrieved only if the escape conditions are satisfied. This way, the deliberative system monitors the current execution and the retrieval process only takes place when necessary. The similarity measures for the features are trapezoid membership functions initially given by an expert and modified afterwards according to the robot's performance. The overall similarity is based on the Max-Min composition of individual similarities. They also introduce an adaptation process through case merging, and the revision step based on the number of states executed.

They compare their system with other teams obtaining successful results. Their main argument for the success of their approach is the flexibility of the system since modifying cases results in modifying the performance of the soccer players. The claim is that knowledge representation based on first order logic is readable by humans, and therefore, incorporating expert's knowledge is fast and easy.

2.2.3 Berger and Lämmel

Recent work has been initiated by Berger and Lämmel [8] where they propose the use of a CBR system to decide whether a "wall-pass" should be performed or not. A "wall-pass" consists in passing the ball to a teammate, to immediately receive a pass again from the latter. The idea is to distract the opponent so the first player can move to a better position. Their work is applied to the Simulation League. A case represents the positions of the most relevant players on both teams in a given situation. They introduce a procedure to extract these relevant features and show that mostly three or four players are sufficient to describe the case. The solution of the case indicates if a "wall-pass" is possible or not. The similarity is based on Euclidean distances between players positions. Case Retrieval Nets (CRN, introduced in [36]) are used for the retrieval process in order to speed up the search. In order to build the case base, they analyze log files of previous games to extract all potential "wall-pass" situations automatically and manually classify them afterwards.

2.2.4 Wendler et. al

Since the initiation of RoboCup, Wendler et al. have addressed different problems within this domain. The first one, and more related to our work is presented in [72]. In this work, they propose to learn about the opponents and based on the observations adapt the actions of the players within the Simulation League. More precisely, the systems indicates the positions where the players should move according to the opponents actions. Thus, the features of the problem description are: state of the pitch by means of segments of all players (in our work we propose the use of ellipses of different sizes for each opponent player), time steps until a player controls the ball, preference directions of a player, available power resources and distance to ball and players. Two similarity measures are defined based on the domain of the features: for the state of the pitch, two regions are considered to be similar if they are neighbors; and for the remaining features, since they are represented by numerical values, distance functions are proposed. In contrast to our work, the overall similarity is computed as the average of individual similarities (we propose the harmonic mean instead). Similar to the work presented by Berger, the retrieval process is done through Case Retrieval Nets. They propose an off-line learning which would correspond to a training stage to build up the case base, and an on-line learning, to adapt cases to the opponents in the game.

Continuing with the ideas of studying the opponent team to improve the performance of the team players, in [70] they address the behavior recognition and prediction problem based on external observation. The CBR system models the function that maps the situations of a game (represented by trigger patterns) and the behaviors of the players (behavior patterns). Triggers and behaviors consist of different attributes, such as player initialization for pass, vector from passer to receiver, distances between players, ball speed, direction of ball movement, etc. The similarity measures for triggers and behaviors are defined as weighted sums of local similarities (attributes similarities). The weights are determined either manually by the expert, or automatically. During a game, when a trigger is identified, the case base is searched for similar triggers. The retrieved case is then adapted and the resulting behavior is compared with

the actual behavior observed. If the similarity between the predicted behavior and the observed one is below a given threshold (not similar enough), a new case case is retained having the triggers as the problem description and the observed behaviors as the solution description. The results obtained through experimentation show that although the system performs quite well, the prediction model is team-dependent, i.e. it is specific for each team. Therefore, when switching opponent teams, behavior predictions are degraded.

Finally, in [71] a fault-tolerant self localization approach is proposed by means of CBR techniques. In this occasion the work is applied to the Four-Legged League. A case is defined as follows: the problem description represents an omnidirectional picture from the robot point of view, where the features correspond to size and positions of the landmarks in the image and the angles between pairs of landmarks; the solution description corresponds to the position of the robot on the field. Cases are gathered through a two-step semiautomatic process. First, a table relating the landmark distances and their perceived sizes is manually created. Next, the distances and angles between markers and goals are automatically derived. The similarity measure results from the composition of the individual similarities through a weighted sum. Once again, CRN are used for the retrieval process, where a set of neighboring cases from the most similar case are retrieved. A case is considered as a neighbor of another case if the distance between the landmarks of both cases is no more than 50cm. The solution proposed by the system corresponds to the weighted sum of all solutions, i.e. the position of the robot.

2.2.5 Marling et al.

Three CBR reasoner prototypes are presented in [41]: the first one focused on positioning the goalie, the second one on selecting team formations, and the last one on recognizing game states. The prototypes are applied to the Small-Size League, although the experiments are validated in simulation only. For all prototypes, the case description represents a snapshot of the field. The features in the problem description and the solution of the case differ from one prototype to another based on the task to learn. Next, we briefly describe each prototype.

The case structure of the first prototype, positioning the goalie, consists of: a snapshot of the goalie's half field, where the features correspond to the positions and orientations of the players, as well as the ball's position. The solution indicates the success or failure of the goalie's move and the position of the ball after the attempted block. Cases in the case library are organized based on the ball's position to speed up the search. Hence, the library is portioned in three categories: near, middle and far. The similarity measure corresponds to the distance between the current ball's position and the one indicated in the case. After experimentation, they found out that the new prototype was not improving the current reactive one, and therefore, decided to try with high level decision, as team formations and game state recognition.

The second prototype, team formations selection, is aimed at being used by a meta-agent, and not the players themselves. The problem description is composed of a snapshot of the field (the players' and ball's positions); derived features from the previous features and state description such as offensive, defensive, transitional situation; and short-term goals and subgoals. The solution

description indicates the team formation, i.e. a set of roles for each player. The role indicates where the robot should move and the task to perform. Similar to the above prototype, cases are grouped based on the state description (defensive, offensive and transitional).

Finally, in the third prototype, game state recognition, cases represent a whole snapshot of the field, indicating the positions of the robots and the ball including symbolic features derived from the spatial features such as number of defenders in defensive zone, attackers in offensive zone, boolean variable indicating whether the ball is near the goal or not, etc. The solution of the case corresponds to a characterization of the state of a game. Some examples are *Man On* or *Two on One*. The retrieval process consists in a standard nearest neighbor algorithm. They plan to extend the case description to more vision frames, instead of only considering a single snapshot to capture robot and ball motion as well.

As mentioned in the beginning, the work presented was not yet finished and therefore, experiments with real robots are not presented. Future improvements as well as new open trends for CBR in this domain are broadly discussed.

2.2.6 Ahmadi et al.

A common drawback of CBR systems usually discussed among researchers is the difficulties for fast retrieval in large case bases. Focusing on this issue, Ahmadi et al. [4] present a two-layered CBR system for prediction in the Simulation League. A case is evaluated based on low level features, i.e. ball's velocity, players' positions and velocities and some predefined recent events. However, the importance of a player position varies based on its relation with the ball location. Thus, a player close to the ball has more importance compared to another one that is far away. This importance is modeled through weights that are assigned to the players based on the different situations. The upper CBR layer is in charge of assigning these weights. Thus, every lower layer case must be adapted to propose different solutions based on the areas of the field where the situation is taking place. The solution of a case indicates the next ball's and players' positions. The similarity measure compares the positions of the ball and the players through a weighted sum. The initial case base is manually created, and afterwards, new cases are introduced during the system's performance. Positive and negative cases are retained.

2.2.7 Steffens

The last work reviewed in this section corresponds to the work presented by Steffens [58] addressed to opponent modeling in the Simulation League. Similarly to the work presented by Ahmadi above, he argues that the similarity measure should be adapted to the situation and role of the agent whose action is to be predicted. While Ahmadi modifies the weights of the positions of players taken into account, Steffens proposes a similarity measure that considers more or less features when comparing the current problem to solve with the cases. The relevance of the attributes is based on the positions and roles of the agents and it is obtained from a goal dependency network (GDN [59]) which

represents general knowledge. The high-level actions to be predicted are *hold ball*, *dribble*, *intercept ball*, *pass* and *shoot on goal*.

2.3 Other Models Applied to RoboCup

Besides CBR techniques, other approaches have been studied to solve the action selection problem. In this section we review first those fields where most researchers have focused their efforts on, such as Reinforcement Learning, to finally review a set of less common approaches used within the RoboCup domain.

2.3.1 Learning from Observation or Imitation

Although not much work related to RoboCup has been done, we are interested in remarking it due to its similarity with Case-Based Reasoning techniques. The aim of this technique is to model agents that learn from observing other agents and imitating their behavior. As in CBR, the learning agent selects the most similar past observed situation with respect to the current problem and then reproduces the solution performed at that time. The main difference between these approaches is that the learning agent is not able to improve the observed agent since there is no feedback in the model.

Lam et al. [33] focus their research on action selection based on scene recognition in the Simulation League. A scene is described by the positions of objects on the field, i.e. robots and ball, with respect to a single player. Two representations are proposed: continuous (distance and angle from the player to the objects) or discrete (fixed regions on the field). Similar to our work, a matching process is defined in order to map the objects in the current scene with the ones indicated in previous scenes. Finally, the distance between two scenes is computed as a weighted sum of the individual similarities. A k-nearest neighbor algorithm is used to obtain the most similar scenes. Each scene is associated to an action. If more than one scene is retrieved, the most common action (majority voted) is selected.

This work is closely related to ours. However, the main differences are: the number of agents implied in the scenes (we include teammates which interact among them, while they only include one teamate); the objects locations (robots and ball are within fixed regions of field in [33], whereas we deal with variable regions); modeling uncertainty (in our work we include fuzzy functions to this end); and the solution of the problem (we deal with a sequence of actions for each teammate instead of a single action in [33]).

2.3.2 Reinforcement Learning

Reinforcement Learning (RL) [61] is a classical machine learning technique that has been frequently used in the RoboCup domain. Although the obtained results are usually successful, the main drawback of this technique is the large state space that most problems present. As a consequence, a large amount of learning steps are required to find the policy that matches states and actions. Hence, most of the times this technique is not feasible when dealing with real

robots. Nevertheless, researchers have tried different approaches to overcome these drawbacks as we review next.

Riedmiller et al. [52] focus their work on learning two different skill levels: moving level (low level) and tactical level (high level). The former refers to learning a specific move, for example, learning to kick, while the latter refers to learning which move should be applied at a certain point, as *pass the ball*. The work is restricted to the Simulation League, and they only used the moving level during a competition. With respect to the tactical level, they experimented with two attackers against one or two defenders. The attackers used the approach presented, while the defenders used a fixed policy.

Similarly, Kleiner et al. [28] apply a hierarchical RL in a Semi Markov Decision Process (SMDP) framework. In their approach they show that learning skills and the selection of these simultaneously (not separately as in [52]) is advantageous with respect to focusing only on one level at a time. They apply their work to the Middle-Size League, but the learned policy is obtained through simulation. Results with the real robots show that more than an hour would be necessary to improve the hand-coded action selection mechanism.

Ahmadi and Stone [5] introduce a Markov Decision Process (MDP) for action selection between two types of kicks in the Four-Legged League. In their approach they compute off-line the value function V for the MDP without considering opponents and assuming a static environment. During the robot's performance, they distinguish two phases: planning, where the robot selects a kick based on the off-line learned policy, and replanning, when opponents appear in scene. The replanning stage consists in recalculating the Q values of the MDP for those states where an opponent is likely to be located. Thus, they reduce the computational complexity of managing the MDP updating the policy the minimum number of times (only when an opponent appears). Their experiments with real robots in controlled scenarios show that the replanning algorithm improves a policy without on-line update.

Modular Q-learning is proposed by Park et al. [48] within the Small Size League. In their work, the default behavior of each robot is to move around the field using a navigation mechanism (uni-vector field navigation, similar to a potential field). When a robot is within a boundary of the ball, the action selection layer switches the robot behavior to the shoot action. Each robot has its own learning module which is in charge of determining when a shoot action should be taken. Robots' roles are fixed covering different regions of the field. Conflicts regarding which robot should move to shoot the ball in overlapping regions are solved through a mediator. Hence, when two robots select the *shoot the ball* action, the mediator intervenes to indicate which of the robots is the one to perform the action, probably switching roles for short periods. Examples with real robots are presented.

We can also find in the literature combined approaches such as the one presented by Duan et al. [15]. In their work they propose a hierarchical learning module that combines fuzzy neural networks (FNN) and reinforcement learning (RL). The learning task includes dynamic role assignment, action selection and action implementation (low level skills). The hierarchical system allows flexibility and independence to modify a certain layer without modifying the remaining ones. Based on the assigned role, the robot may select among a set of actions: the attacker may shoot, pass, dribble and swing, while the defender may intercept, mark, clear and defend (go back to its home position). For the

offensive player action selection, besides considering the distances between the robot and opponent goal and the robot and the ball, two parameters that indicate the chances of the opponent to intercept the ball after a shoot or a pass are computed. Regarding the defensive robot, the postures of the opponent attacker and the ball are relevant variables to take into account when selecting the action to perform. Experiments in a simulated environment are performed to evaluated the proposed approach.

2.3.3 Pattern Recognition

Recognizing and learning from other teams is a desired ability in order to improve the strategy of a team, and thus, the action selection strategy of the players. Therefore, some researchers address the opponent modeling problem through pattern recognition of sequences. The work reviewed in this section is all applied in the Simulation League, which is the league that can provide most reliable data for the problem tackled here.

Huang et al. [23] present a mechanism to recognize and retrieve teams' plans. A plan includes the agents that take part of the plan, the starting conditions, the goal state and the agents' behaviors. In order to recognize plans, the first step is to translate observations into agents' sequential behaviors. These sequences are gathered following a set of rules and transformed into trie structures (a multi-way tree structure [29]). They define several events that activate the recognition functions (for instance, when a robot gets the ball, the algorithm starts recording). Plans are obtained retrieving the most significant subsequences of behaviors within a trie structure through statistical dependency test.

Lattner, Miene et al. [34, 43] present an approach that applies unsupervised symbolic off-line learning to a qualitative abstraction in order to create frequent patterns in dynamic scenes. The quantitative data is represented by time series. In a first abstraction step, each time series is segmented into time intervals which satisfy certain monotonicity or threshold conditions. In the second step the attribute values describing the intervals are mapped into qualitative classes for direction, speed or distance. The idea then is to create patterns based on the qualitative information of the environment (input). The result of learning is a set of prediction rules that give information about what (future) actions or situations might occur with some probability if certain preconditions satisfy. Patterns can be generalized, as well as specialized.

2.3.4 Fuzzy Theory

Fuzzy Theory is another selected approach [55] since it naturally handles uncertainty, which is highly present in the real world. The advantages of fuzzy theory is that it models imprecise or vague concepts, usually used by humans in their daily reasoning. Using fuzzy rules is more intuitive for an expert, rather than trying to find functions that model the problem to solve. However, the difficulty here is to determine the fuzzy membership functions that represent the variables to use in the problem to solve.

Within this context, a fuzzy logic based strategy for dynamic role assignment is introduced by Sng et al. [57]. Role assignment is related to action selection, since each role determines the actions the robots should perform. Hence,

the selection of a role indirectly implies the actions selection for a player. Their work is applied in the Small-Size League (with only 3 robots), where a centralized coordinator (the fuzzy role selector) assigns the roles to the players. The algorithm first selects the "attack ball" role (the robot moving after the ball), and then assigns the remaining roles to support the main role. Four fuzzy variables describe each robot situation (distance to ball, orientation, shoot angle, and path obstacle). The fuzzy rules determine the output fuzzy membership value for every robot. The robot with the highest value is assigned as the robot going after the ball. The formation of the other two robots is then derived similarly (using fuzzy rules), positioning themselves near to the robot going after the ball.

Lee et al. [35] present similar work where a fuzzy logic system is used as a mediator to handle situations where more than one robot may be responsible for an area. Each robot has a role based on its home area. However, when the ball is positioned in overlapping areas the robots should switch their roles to achieve a cooperative strategy, i.e. one robot should go for the ball, while the other one should assist it. To this end, fuzzy rules for each overlapping region (three predefined regions) are introduced where the input variables are the distance between the robot and the ball, angle between the robot and the ball, and the angle between the robot and the goal. The output indicates the role of the evaluated robot, i.e. support, shoot, change role with the other robot, etc. Their work is applied in the Middle-Size League, although experiments are only shown in a simulated environment.

Related to the action selection problem, Wu and Lee [73] focus their research on the selection of five action categories: *intercept*, *shoot*, *block*, *sweep* and *stand by* within the vision-based soccer system (as in the Small-Size League). The input variables are the defense factor (distance of the robot to the home and opponent goal), the competition factor (distances and angles of all robots to the ball) and the angle factor (ball accessibility from the robot point of view). The output of the rules indicate the action to perform by the robot. The experiments show one-to-one games. Although this work is more related to ours, in the sense that explicit actions are chosen, the approach only considers a single player and therefore, no cooperation can be considered.

2.3.5 Planning

Although a traditional technique within Artificial Intelligence, planning is not generally applied in complex domains where uncertainty, incomplete knowledge, real time response, etc. are present. However, we can find some work within the robot soccer domain, since planning can also be seen as a decision making layer for selecting the appropriate actions to perform.

Fraser and Wotawa [18] propose a framework based on STRIPS [17] where optimizing the plan is not the main purpose, but monitoring the plan execution instead. Knowledge representation is done using first order logic and uncertainty is not considered on purpose. Given the initial state I observed by an agent, a plan p is calculated to achieve the goal G using the domain theory. They extend the classical planning problem definition by plan invariants. A plan invariant is a manually defined logical sentence that has to remain satisfied in the initial and all subsequent states (similar to our work and the escape conditions proposed by Lin and Chen). Possible reasons to invalidate a plan

are: inexecutable actions (a precondition is not fulfilled), failed actions (due to imprecision in the robots actions), unreachable goal (due to external events) and unfeasible goal (changes in the environment). Hence, plan invariants are monitored at all times during plan execution. They also introduce a mechanism for achieving cooperative planning through role allocation. Once again preconditions and invariants assign roles to players. Since the framework is applied in the Middle-Size League, all robots may not share the same world state at a given point. In order to avoid conflicts in the role assignment, once a robot selects its role, the robot broadcasts it with an associated utility value. Hence, the robot with higher utility value keeps its role, while the conflicting robots must pick some other role. Explicit communication is used among robots to achieve interaction. For instance, a robot may request a pass to another robot. Invariants are used to monitor the interaction and to make sure that both robots agree on the cooperative task.

Hierarchical Task Network (HTN) planners have been proposed by Obst and Boedecher [46] to achieve coordinated behavior while agents follow the strategy suggested by a human expert. HTN planning makes use of domain knowledge to speed up the planning process. Tasks may be complex or primitive. The HTN planners use methods to expand complex tasks into primitive tasks that can be then executed using planning operators. Their planner generates what they call *plan stub*, a task network with a primitive task as the first task. As soon as a plan stub is found, the agent can start executing the task. To handle non-determinism, a plan is treated as a stack. The tasks in the stack are marked as pending or expanded. The former ones are tasks waiting for execution if they are primitive tasks, or waiting for expansion, if they are complex tasks. When a subtask fails, all remaining subtasks of the complex task are removed from the stack and it is checked if the complex task can be tried again. Once a task is successfully finished, it is removed from the stack. The lazy evaluation in the subtask expansion ensures that the planning process is fast enough for the requirements of the working domain (Simulation League 3D) and at the same time, maintains its reactiveness property to handle changes in the environment.

2.3.6 Neural Networks

Neural Networks (NN) [54] have been proved to efficiently perform in many domains, including robot control. However one of their main drawbacks, as in decision trees algorithms, is the large amount of data needed for the training, which is not always feasible to provide. Another drawback is that the knowledge of the system cannot be evaluated or directly modified by a human expert.

Initial work is presented by Kim et al. [26]. They propose an action selection mechanism (ASM) based on the role of the player. The ASM is composed of four modules: the action set module computes run-time parameters and feasibility for executing the available actions; the internal module selects an action given the current situation without considering opponents; the supervisor may alter the actions attributes or enforce certain actions; and finally, the intervention module calculates the level of disturbance of opponents, i.e. how the existence of an opponent interferes in the current situation. This latter contribution is similar to the work presented by Ahmadi and Stone, where the Q

values of the states where opponents might possibly be located are modified on-line. In order to compute the level of disturbance a multi-layer perceptron is proposed. The training set is manually obtained: an expert observes a game and labels those situations where the opponents should be taken into account, and therefore, their disturbance level is high. The MLP is a two layer feed-forward neural network. The approach is tested with real robots in a one-to-one scenario.

Jolly et al. [24] present a more complete work, where a two-stage approach using neural networks for action selection in the Small-Size League is proposed. The first attempt is focused on deciding which of the two robots near the ball must go after it while the other remains as a supporter. Hence, the input variables to the NN correspond to the distances and angles to the ball. They also introduce a *forward* boolean variable, which indicates if there is a teammate ahead of the ball. This way they enhance the accuracy of the decision-making with the global strategy of moving forwards, i.e. moving towards the attacking goal. The actions of the robots are based on the region where the ball is located. Thus, if the ball is within the attack zone, the robot going after the ball should kick to goal; within the defense zone, the action is to intercept the ball and pass it to the teammate; and finally, within the pass zone, the robot should pass the ball to a teammate if this is a forward teammate. Otherwise, a kick to goal is performed. The NN is a three layer feed-forward network, with five inputs and two outputs. The first stage of the learning process consists in using an evolutionary approach to roughly acquire the neural network weight matrices. Next, the NN is used to fine-tune the weights. The training data is obtained randomly generating field configurations and the corresponding robots' actions are obtained through rules. They also extend their NN to compound networks in order to handle larger teams (5 vs. 5). They show results for the learning curves of experiments in simulation for 3 vs. 3, 4 vs. 4 and 5 vs. 5 scenarios.

2.3.7 Evolutionary Algorithms

Evolutionary computation is based on the mechanics of natural selection and the process of evolution [22]. Chromosomes encode the potential solutions of the problem to solve. During the search, chromosomes are combined and mutated in order to find the best solution (although it is not guaranteed to find the optimal one).

Nakashime et al. [45] propose an evolutionary method for acquiring team strategies in the Simulation League. The algorithm evolves rules (called action rules) for determining the actions of the players. The actions are of the type: if $agent_i$ in area A and nearest opponent is B_i then action is C. The chromosome corresponds to a concatenation of the ten players' possible actions in the different predefined regions of the field (48 regions). There are ten possible actions, such as dribble toward the opponent side, pass the ball to nearest teammate, clear the ball towards opponent side, etc. Once the rules are evolved, during a game the players follow the rules except for those situations where a player is in possession of the ball within the penalty area. In these cases, the player first evaluates a shoot to the goal. If the evaluation results in success, the player kicks the ball directly. Otherwise, it follows the rule. This way they ensure reactiveness in the player's behavior.

Related to evolving rules, but in this occasion fuzzy rules, Park et al. [47] propose the use of evolutionary algorithms to determine the appropriate fuzzy control rules for the path planning problem in robot soccer. More precisely, they propose some modifications in the classical evolutionary algorithm in order to automatically detect the sensitivity of various genes in contributing to the fitness solution. This way they ensure that the evolved chromosomes are goal-oriented, in the sense that their performance is tested against a specific goal for which they are good at, and not for general purposes. The proposed modifications for the parent selection process assist in the evolution of optimal solutions for multi-objective path planning problems. Experiments with a real robot prove that the proposed approach generates paths that have shorter elapsed times with significantly reduced variation.

Luke et al. [40] propose to learn the robots behaviors through Genetic Programming instead of hand-coding them. Their main goal is not to obtain finely-tuned players, but to study the feasibility of evolving a fair team in the Simulation League. The individuals are represented as program-trees and crossover and mutation are the operators used to evolve them. Two trees were to be learned: one for making kicks, and the other, for moving the player. They also propose two types of teams: homogeneous, where each player would follow the same learned trees, or heterogeneous, where each player would develop its own trees. Teams' fitness were assessed based on the game score of competitions (each team evolving its own behaviors). They expected that the heterogeneous team would outperform the homogeneous team. However, the evaluation showed the contrary. They believe that more training time would permit the heterogeneous team to improve the players strategy, and therefore, outperform the homogeneous team.

2.3.8 Other Approaches

Konur et al. [31] focus their work on learning decision trees for action selection for a whole team (defenders, attackers and midfields) in the Simulated League. They restrict the learning to the players in ball possession. They define a set of meta-level actions which are to be selected by the decision tree (C4.5 introduced by Quinlan [49]). Some of the 35 features used for the attribute set are type of player, playing region, closest teammate to ball, distance and angle to ball, etc. The advantages of decision trees are that the encoded rules are "readable" for humans in the sense that they are easy to understand and inspect by an expert. However, in Konur's approach because of the large number of features used during the learning, the resulting rules are not so understandable nor easy to follow. Instead, we believe that the use of cases is closer to how humans reason. Moreover, the drawback for using decision trees is the need of a large set of training examples, which in the case of real robots, is not feasible. Instead, the Simulation League provides enough data to gather the required information.

Bayesian classification methods are very common to use when dealing with uncertainty because they are based on probability theory [13]. Hence, within the Simulation League 3D, Bustamante et al. [9] present their work addressed to the action selection using Naive Bayesian Networks. The input variables of a Naive Bayesian classifier should be discrete, and therefore, they use a fuzzy extension: a Fuzzy Naive Bayes Classifier. The task of the agent is to evaluate

the success probability of a pass. To this end, the features of the classifier are fuzzy distances and angles to ball and players (teammates and opponents). In their experiment, the player in possession of the ball has to select the most adequate teammate to pass the ball, given their positions on the field. Hence, for each teammate and each opponent, the agent computes the probability of success. The teammate with higher probability is chosen to perform the pass.

2.4 CBR Applied to Other Robotic-Related Domains

Next we review some work done in the past years within the robotics field where CBR techniques have proved to successfully perform. Similar to the RoboCup domain, real-time response (although probably not as restricted as within robot soccer), uncertainty, imprecision and dynamism are some of the features that describe the environments where the work we review next is addressed to.

Ram and Santamaría [51] and Likhachev and Arkin [37] propose the use of CBR in the robot navigation domain: the SINS system and the MissionLab system respectively. The goal of the CBR system is to set the gain parameters for the motor schemas of the robot's navigational layer. In both approaches the case description is represented by feature vectors. During retrieval, while SINS evaluates the current problem with respect to the case base in one round, MissionLab first compares the spatial feature vectors of the cases filtering those with low similarity. The selected cases are then compared using the temporal feature vectors. Once a case is retrieved, SINS reuses the new retrieved case. MissionLab instead includes a case switching tree to decide whether the currently applied case should still be applied or should be switched to the new retrieved case. Next, the gain parameters of the case solution are adapted and reused. Another difference between both approaches is the learning process. SINS receives feedback to evaluate whether the retrieved case should be modified based on the adapted solution currently reused, or a new case should be created instead. Regarding MissionLab, Kira and Arkin present in [27] an extension of the system where a learning module decides which cases to remove if a new case has to be created and the case library is full.

A global navigation strategy in large-scale dynamic environments by means of CBR techniques is presented by Kruusmaa [32]. The problem is to find the path between a starting point and a given goal. The system uses a grid-based path planning and therefore, at least the shape and size of the environment must be indicated. However, the presence and location of obstacles is unknown and changes over time. Hence, the main goal of the CBR system is to minimize collision risk, globally choosing routes where few obstacles are encountered. Given a new task (going from point A to point B), the system chooses between planing a new path through a probabilistic planner or using the description of an old path from the case base (either opting for exploration or exploitation). The solution description corresponds to the path. The outcome of the solution is the cost, which reflects how easy the path was to follow. The cost is updated every time the path is reused indicating the average characteristics of that path. Choosing the path with lower cost leads the robot to chose safer paths and minimize time travel. Similar to the approach presented by Kiran [27], a case forgetting process is included to prevent the case base from growing to much.

Thus, an old case is forgotten and the new one is stored if the cost of the new case is lower. If no similar case is found for the current problem solved, it is incorporated in the case library as well.

Similar work addressed to the path-planning problem is proposed by Haigh and Veloso [21]. They present a path planner for the city of Pittsburgh using the PRODIGY planner and its analogical reasoning [68]. The general knowledge corresponds to the map of the city represented as a planar graph whose edges indicate street segments and the nodes, intersections. A case corresponds to a trace of the plan generated by PRODIGY including search decisions to be avoided at reuse, situations encountered at execution time as explanation of errors and the replanning done to overcome those errors. A case is also approximated by straight line segments in a two-dimensional graph, which acts as an indexing feature for facilitating the retrieval process. As in [32] a compromise between finding new routes or using old ones is taken into account. Therefore each case is assigned an efficiency value that indicates the "quality" of a case considering traffic conditions, road quality and time of the day. This factor is also considered during retrieval. The retrieval process returns a set of cases ordered according to the sequence in which the metric (geometric similarity metric introduced in [20]) believes they should be retrieved. The retrieved cases are then merged to obtain the solution route. During reuse, extra planning can be performed if needed (e.g. a closed road). In this situations, the failed retrieved case is not altered, and instead, the efficiency value is modified.

The problem of indoor navigation using sonar maps is addressed by Micarelli et al. in [42]. The goal of the system is to classify the sonar-based maps into a set of predefined categories that represent structured environments within a building, such as corridor, corner, crossing, end corridor and open space. A case is defined as a tuple where the problem description corresponds to a digital sonar map (reprented by an array of 360 real numbers between 0 and 1), and the solution description corresponds to a topological feature, i.e. one of the categories. There is a first training stage to build the case base where a human expert indicates the solution of the map detected. As the system acquires more cases, the number of queries to the expert decreases. The similarity measure corresponds to a cross-correlation factor metric.

Urdiales et al. [64] present a sonar-based reactive navigation approach. The work is addressed to local obstacle-avoidance of an autonomous robot. They include a CBR system to determine the direction the robot should follow to safely avoid colliding with close obstacles. The deliberative layer determines the robot path and takes into account static or know obstacles, but does not deal with unexpected obstacles found during navigation. Thus, once the goal is set, the robot changes its heading to reach it in a straight way. The reactive layer is only triggered when the robot detects obstacles through the sonar. The sonar readings are part of the problem description of a case, as well as the goal of the task, i.e. the point that the robot should reach. Since no global model of the environment is used, the goal is represented by the direction vector from the current position of the robot to the goal. A nearest neighbor algorithm is used for the retrieval step. A first training stage is performed to acquire the case base. The robot is manually guided through different paths in order to create new cases. A new case is included each time a significant sensor configuration is detected. This process is also used during the autonomous robot performance after the training stage.

Abbreviation	Technique
CBR	Case-Based Reasoning
RL	Reinforcement Learning
LO	Learning from Observation
EA	Evolutionary Algorithms
DT	Decision Trees
FT	Fuzzy Theory
PF	Potential Fields
PR	Pattern Recognition
NN	Neural Networks
A	Auctions
P	Planning
BN	Bayesian Networks

Abbreviation	League
SM	Simulation League
SSL	Small Size League
MSL	Middle Size League
4LL	Four-Legged League

Table 2.1: Table of abbreviations.

2.5 Summary

We summarize the work reviewed within the RoboCup domain in Table 2.2. The classification is based on the problem the work is addressed to and the league where it is applied to, indicating for each work the used technique. The nomenclature used corresponds to the abbreviations presented in Table 2.1. The problems we have focused our attention to are: *action selection*, *opponent modeling* or *state recognition* (team formation recognition), *role assignment*, *positioning* (locations on the field where the robots should move), *localization* (computing the position of the robot on the field by itself) and *skills* (learning low level actions such as kick the ball, walk, etc.).

We must remark that we only list a summary based on the work presented in this thesis and therefore, with direct relation with the thesis topic. We do not intend to summarize a complete survey of the work done so far within RoboCup.

We can observe that in general most of the work is applied to the Simulation League. Besides being the earliest league, the advantage of this league is that researchers do not have to deal with much of the problems that arise with real robots. Hence, they can easily focus their efforts on developing skilled soccer players using different techniques addressed to high level decision-making without considering lower level problems. Moreover, and as mentioned before, this league also provides much more information about the state of the world during a game as well as afterwards (log files produced during a game), which can be later on widely explored using machine learning techniques.

The Middle-Size League has also produced an important amount of work. Probably the appeal of this league with respect to the other leagues is that on

	SL	SSL	MSL	4LL
Action Selection	CBR: [10], [38], [8][1] RL: [52] LO: [33] EA: [45] DT: [31] P: [46] BN: [9][2]	RL: [48], [15][4] FT: [73][3] NN: [26][3], [24][4]	RL: [28][5] P: [18][4]	CBR: [25][6] RL: [5][7,10]
Opp Mod/ State Recog.	CBR: [70], [4], [58] PR: [23], [43]	CBR: [41][4]		
Role Assignment		CBR: [41][4] RL: [15][4] FT: [57], [35][4,8]		A: [65][8]
Positioning	CBR: [72]	CBR: [41][4] EA: [47][9] PF: [48], [63][5]		PF: [65][8]
Localization				CBR: [71][9]
Skills	RL: [52]	RL: [15][4]	RL: [28][5]	

Table 2.2: Related work classification.

(1): The decision making mechanism only indicates whether a pass is feasible or not.

(2): Selects which teammate is the most appropriate to receive a pass.

(3): 1 vs 1 experiments.

(4): Experiments in simulated environment only.

(5): Experiments with a static opponent(s).

(6): No experimentation reported.

(7): The decision making mechanism chooses the type of kick to perform between two types of kicks.

(8): Experimentation without opponents.

(9): Experimentation with one robot.

(10): Random movements for opponents.

the one hand it deals with real robots (robots are rather small and quite precise) but on the other hand, it maintains a centralized system where most of the processing routines, as well as the robot control, take place. Hence, although having to deal with both hardware and software problems, the decision-making is done on an off-field PC. The state of the environment is usually obtained through an overhead camera and the image processing is done by the off-field PC. Thus, robust computer vision algorithms can be applied, minimizing the uncertainty in the incoming information. As we can observe from the summary table, several different approaches have been applied to this league, attempting to solve most of the problems remarked here.

The two latter leagues correspond to the most challenging ones, since robots are completely autonomous and all the decision-making and computational processes have to be done on-board. The advantage of the Four-Legged League with respect to the Middle-Size League is that researchers do not have to deal with hardware design. The robots used in this league are commercial robots and cannot be modified. However, the drawback is that the robot processor is limited and therefore, simple algorithms must be designed. On the contrary, the Middle-Size robots can be adjusted as required, and more powerful PC's can be used. The work presented so far in these leagues is much more limited compared to the other two leagues.

Regarding the experimentation within the leagues with real robots (SSL, MSL and 4LL) we can observe that, in general, simplifications of a real game are considered. Thus, simulated environments are used to test the proposed approaches, or if experimenting with the real robots, opponents are either omitted, static or performing random movements. These assumptions clearly show the difficulties that researchers have to face when dealing with real robots performing tasks in a real environment.

Our work can be classified within the action selection, role assignment and positioning problem domains applied to the Four-Legged League. As we will explain through this dissertation, the CBR system retrieves and adapts a case specifying the actions the robots must perform (action selection problem). Moreover, it indicates which actions should perform each robot and their initial positions. Hence, the role assignment and positioning problems are solved through the case reuse. Although two works have been presented in the action selection domain within this league [25, 5], in this dissertation we present a complete framework addressed to the **decision-making** of a multi-robot system, where a set of actions for each robot is selected (not only two possible actions as in [5]), and the subsequent **execution** of these actions (the work presented in [25] is preliminary and only refers to the first stage, the decision-making). Furthermore, we have included the **cooperative** aspect in the task execution through explicit passes between robots. To this end, a multi-robot architecture and a coordination mechanism are introduced. To evaluate our approach, we have performed experiments consisting of two vs. two scenarios both in simulation and with real robots, where the two attackers play against a defender and a goalie (non-random opponents).

Chapter 3

Modeling the CBR Approach: The Retrieval Step

In this chapter we introduce the first step of the CBR cycle: the retrieval step. Although most of the formulation is domain-oriented, the ideas presented can be easily translated to other domains where a team of robots must perform a joint task, such as moving objects in dynamic environment. First we describe the case definition and the case base structure. Next we present the retrieval process itself, defining the similarity measures and a filtering mechanism to select a set of candidate cases. Finally, experiments in simulation are presented to evaluate the retrieval process.

3.1 Case Definition

A case represents a snapshot of the environment at a given time from a single robot point of view. We call this robot the *reference* robot, since the information in the case is based on its perception and internal state (its beliefs). The case definition is composed of three parts: the problem description, which corresponds to the state of the game; the solution description, which indicates the sequence of actions the robots should perform to solve the problem; and finally, the case scope representation, which contains additional information used to retrieve the case. We formally define a case as a 3-tuple:

$$case = (P, A, K)$$

where P is the problem description, A, the solution description, and K, the case scope representation.

3.1.1 Problem Description

The problem description corresponds to a set of features that describe the current state of the game from the *reference* robot perspective. In the robot soccer domain we consider the following features as the most relevant for describing the state of the game:

$$P = (R_G, R, B, G, Tm_G, Tm, Opp_G, Opp, t, S)$$

where:

1. R_G: reference robot's global position (x_R, y_R)

$$x_R \in [-2700..2700]\text{mm},\ x_R \in \mathbb{Z} \qquad y_R \in [-1800..1800]\text{mm},\ y_R \in \mathbb{Z}$$

2. R: reference robot's position (x_R, y_R) with respect to the ball.

3. B: ball's global position (x_B, y_B)

$$x_B \in [-2700..2700]\text{mm},\ x_B \in \mathbb{Z} \quad y_B \in [-1800..1800]\text{mm},\ y_B \in \mathbb{Z}$$

4. G: defending goal

$$G \in \{\text{cyan}, \text{yellow}\}$$

5. Tm_G: teammates' global positions

$$Tm_G = \{tm_1 : (x_{R_1}, y_{R_1}), ..., tm_n : (x_{R_n}, y_{R_n})\}$$

where tm_i is the robot identifier and $n = 1..3$ for teams of 4 robots. This set could be empty for cases where no teammates are implied in the case solution.

6. Tm: teammates' relative positions with respect to the ball

$$Tm = \{tm_1 : (x_{R_1}, y_{R_1}), ..., tm_n : (x_{R_n}, y_{R_n})\}$$

7. Opp_G: opponents' global positions

$$Opp_G = \{opp_1 : (x_{R_1}, y_{R_1}), ..., opp_m : (x_{R_m}, y_{R_m})\}$$

where opp_i is the opponent identifier and $m = 1..4$ for teams of 4 robots. This set could be empty for cases where no opponents are described in the case.

8. Opp: opponents' relative positions with respect to the ball

$$Opp = \{opp_1 : (x_{R_1}, y_{R_1}), ..., opp_m : (x_{R_m}, y_{R_m})\}$$

9. t: timing of the match. Two halves parts of 10 min

$$t \in [0..20]\text{min}, \quad t \in \mathbb{N}$$

10. S: difference between the goals scored by our team and the opponent's team. The maximum difference allowed is 10. The sign indicates if the team is losing (negative) or winning (positive).

$$S \in [-10..10], \quad S \in \mathbb{Z}$$

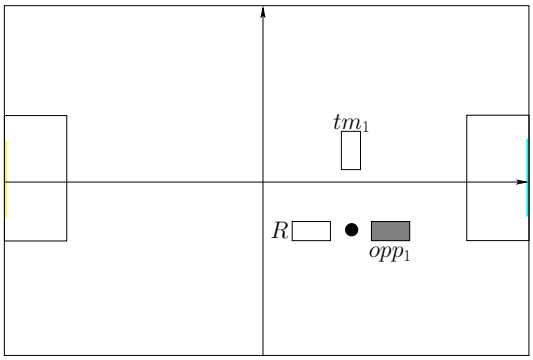


Figure 3.1: Graphical example of the description of a problem. The black circle represents the ball, while the rectangles correspond to the teammates and an opponent (gray).

We include both global and relative coordinates for the robots positions in the problem description because, as we will see through the chapter, for some calculation processes it is advantageous to use one representation or the other. Due to the limited computational resource of the robots it is not feasible to compute the relative representation every time it is needed. Thus, cases are actually stored using the global representation only, while the relative coordinates are derived automatically when loading the case base. Figure 3.1 illustrates a graphical example of the following problem description:

$$P = \begin{pmatrix} R_G & = (517, -506), \\ R & = (-402, 0), \\ B & = (919, -506), \\ G & = \text{yellow}, \\ Tm_G & = \{tm_1 : (919, 337)\}, \\ Tm & = \{tm_1 : (0, 843)\}, \\ Opp_G & = \{opp_1 : (1350, -506)\}, \\ Opp & = \{opp_1 : (431, 0)\}, \\ t & = 5, \\ S & = 1 \end{pmatrix}$$

3.1.2 Solution Description

The solution of a case corresponds to the sequences of actions each robot performs. We call them *gameplays*. In this work, a gameplay must also satisfy two conditions: (*i*) at least one robot has as its first action to get the ball; and (*ii*) only one robot can control the ball at a time. Formally, we define a gameplay as:

$$A = \begin{pmatrix} tm_0 : [a_{01}, a_{02}, \ldots, a_{0p_0}], \\ \ldots \\ tm_n : [a_{n1}, a_{n2}, \ldots, a_{np_n}] \end{pmatrix}$$

where $n = 0..3$ is the robot identifier, and p_i the number of actions teammate tm_i performs (tm_0 corresponds to the reference robot). Actions are either indi-

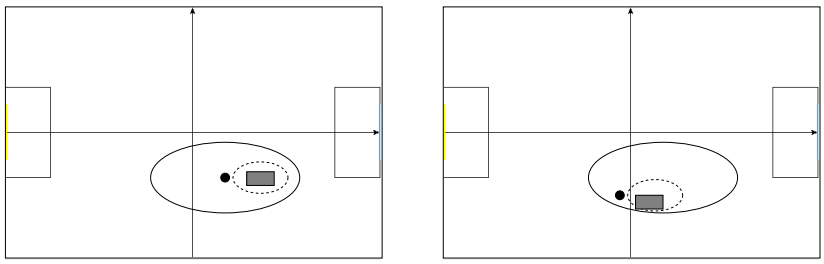
Action	Parameters	Description
NONE	-	no action
GET_NEAR_BALL	-	move close to the ball
APPROACH_GRAB	-	approach the ball and grab it
GO_TO_POINT	(x, y)	move to point (x, y)
TURN	$[\theta \vert (x, y) \vert tm_i]$	turn until the robot's global heading is θ or until facing either point (x, y) or robot tm_i
TURN_WITH_BALL	$[\theta \vert (x, y) \vert tm_i]$	turn while grabbing the ball until the robot's global heading is θ or until facing either point (x, y) or robot tm_i
WAIT_BALL	tm_i	wait until robot tm_i kicks the ball
DODGE	$[side \vert (x, y)]$	dodge to the left or right side, or towards point (x, y)
PICK_BALL	-	get ball
SUPPORT	(x, y)	move towards point (x, y) facing the ball
KICK_TO_POINT	(x, y)	kick the ball towards point (x, y)
KICK	$kick$	kick the ball with $kick$ type (e.g. forward, side, bump, etc.)

Table 3.1: List of available actions and their parameters.

vidual actions, such as "get the ball" or "kick", or joint actions, such as "pass the ball to robot tm_i". The actions may have parameters that indicate additional information to execute them. For instance, in the turn action we can either specify the global heading the robot should have or a point to face; in the kick action we indicate which type of kick to perform (forward, left, ...), etc. Table 3.1 details the list of available actions and their parameters.

During the execution of the solution, all robots on the team start performing their sequences of actions at the same time. The duration of each action is implicitly given by the action type and its initiation depends on the action preconditions. Consider the following situation: robot r_A must pass the ball to robot r_B, and robot r_C has to move to a point p. Without explicitly indicating the timestep of each action, the timing of the overall performance will be: robot r_A starts moving towards the ball to get it, while robot r_B waits for robot r_A to make the pass. Once robot r_A has done the pass, r_B receives the ball and kicks it forwards. In the meantime, since robot r_C has no preconditions, it starts moving to point p independently from the state in which the other robots are. For this example, the solution would be:

$$A = \begin{pmatrix} r_A : [\, get\_ball,\ pass\_ball(r_B)], \\ r_B : [\, wait,\ receive\_ball(r_A),\ kick(\text{forward})], \\ r_C : [\, go\_to\_point(p)] \end{pmatrix}$$



(a) (b)

Figure 3.2: (a) Example of the scope of a case. The black circle represents the ball and the gray rectangle represents the opponent. The ellipses correspond to the ball's scope (solid ellipse) and the opponent's scope (dashed ellipse). (b) Example of a simplified problem description. The opponent's scope is translated with respect to the ball.

3.1.3 Case Scope Representation

Because of the high degree of uncertainty in the incoming information about the state of the world, the reasoning engine cannot rely on precise values of the positions of the objects (robots and ball) on the field to make decisions. Therefore, we model these positions as regions of the field called *scopes*. The scopes are elliptic regions centered in the object's position with radius τ_x and τ_y. The case scope is defined as:

$$K = (ball : (\tau_x^B, \tau_y^B), opp_1 : (\tau_x^1, \tau_y^1), \ldots, opp_m : (\tau_x^m, \tau_y^m))$$

where τ_x^B and τ_y^B correspond to the x and y radius of the ball's scope, opp_i is the opponent identifier, and τ_x^i and τ_y^i, to the radius of opponent opp_i's scope ($i = 1..m$). If there are no opponents in the case, then we do not include any opponent pair $opp = (\tau_x, \tau_y)$. Notice that we only consider the robots that are opponents, and not the ones belonging to the team, i.e. reference robot and teammates (R, tm_i). As we will explain during the retrieval process, we define two different measures for each type of robots. While one requires the use of scopes, the other does not. Therefore, we do not need to include this information for all robots in the case description.

We must also anticipate that the ball's scope is fundamental for the retrieval process as we will explain in Section 3.3. A case might be considered a potential solution only if the position of the ball described in the problem to solve is within the ball's scope of the case. Otherwise, the case is dissmissed.

Regarding the opponents features, Opp, as defined in the problem description it corresponds to the relative positions of the opponents with respect to the ball. The advantage of representing the opponents combining their relative coordinates and their scopes is that we can easily define qualitative locations of the opponents on the field with respect to the ball. Reasoning with qualitative information is advantageous in this kind of domains, specially, as we have said, because of the high uncertainty in the incoming information, and moreover, because it facilitates the generalization of similar situations. For instance,

it is more general to reason about an opponent being in front of the ball, rather than the opponent being in position (x, y).

Figure 3.2a shows a simple example of this situation. The interpretation of this case is that if we want to consider it as a potential solution for a given problem, then the ball should be located within the ball's scope and an opponent should be positioned in front of it. Figure 3.2b depicts a problem example where the opponent is considered to be in front of the ball because it is located within the opponent's scope. Note that the opponent's scope has been translated with respect to the current position of the ball in the problem. This is due to the relative representation of the opponents with respect to the ball. Since the ball is also situated within the ball's scope of the case, we can state that the case in Figure 3.2a is a potential solution to the problem in Figure 3.2b.

3.1.4 Case Example

Following the example shown before for the problem description, Figure 3.3 completes the case representation including the solution description and the case scope. We represent the scopes of the ball and the opponent with solid and dashed ellipses respectively. The arrows show the sequence of actions the robots should perform to solve the problem. Robot R takes the ball first and passes it to robot tm_1 with a left head kick. Next tm_1 receives the pass, turns until facing point $(2700, 200)$ (the pointed location indicated by the arrow in the figure) and kicks with a forward kick. The formal description of this case example is notated as:

$$
case = \left(\begin{array}{c}
\left(\begin{array}{ll}
R_G & = (517, -506), \\
R & = (-402, 0), \\
B & = (919, -506), \\
G & = \text{yellow}, \\
Tm_G & = \{tm_1 : (919, 337)\}, \\
Tm & = \{tm_1 : (0, 843)\}, \\
Opp_G & = \{opp_1 : (1350, -506)\}, \\
Opp & = \{opp_1 : (431, 0)\}, \\
t & = 5, \\
S & = 1
\end{array} \right), \\
\left(\begin{array}{l}
tm_0 : \left[\begin{array}{l} get\_ball, \\ pass\_ball(tm_1, kick(\text{head\_left}) \end{array} \right] \\
\\
tm_1 : \left[\begin{array}{l} wait,\ receive\_ball(tm_0), \\ turn(2700, 200),\ kick(\text{forward}) \end{array} \right]
\end{array} \right), \\
\left(\begin{array}{l} ball : (860, 480), \\ opp_1 : (315, 200) \end{array} \right)
\end{array} \right)
$$

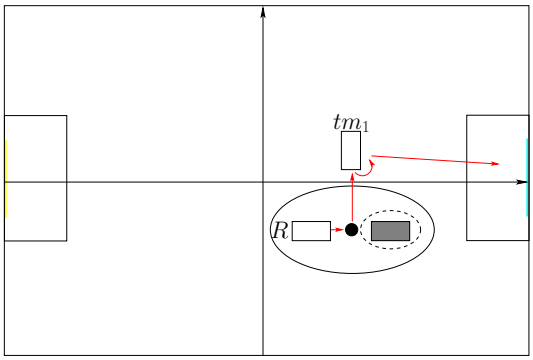


Figure 3.3: Graphical example of the description of a case.

3.2 Case Base Description

Because of the spatial nature of the features in the case description, interestingly a particular case can be mapped into multiple ones through spatial transformations. Thus, from a small set of cases, we can automatically generate a larger set reducing the human effort when building the initial case base.

From the set of features of the problem description, the ball's and robots' global positions and the defending goal have three symmetric properties:

1. with respect to the x axis,
2. with respect to the y axis and the defending goal, and
3. with respect to both axis, x and y, and the defending goal.

Hence, given a description of a problem, we can easily generate three more problems applying spatial transformations based on the symmetric properties shown above. We must point out that the spatial transformations have to be done using the global coordinates of the features instead of using the relative ones. Thus, when loading the case base, we first generate the symmetric cases and then we derive the relative positions of the features for each case.

Similarly, we also compute the symmetric description of the case solution. More precisely, we must transform only the parameters of some of the actions that are related to spatial features, such as right, target point, turn angle, etc. Regarding the case scope, no spatial transformations are needed since they only represent the radius of the scopes. Table 3.2 summarizes the spatial transformations defined above for the different types of features or parameters used in the case description. Figure 3.4 illustrates an example of a simplified case description and its three symmetric descriptions (we omit the scopes and actions for clarity purposes).

Since we are working in a real time domain and because of computational limitations in the robots, it is essential to minimize the time invested during the retrieval process. To speed up the search we use an indexed list to store the cases in memory once they have been loaded. Thus, we separate the cases based on the defending goal feature (yellow or cyan). When a new problem has to be solved, we only look for similar cases in one of the subsets. Searching

	symmetry with respect to		
	x	y & defending goal	xy & defending goal
(x, y)	$(x, -y)$	$(-x, y)$	$(-x, -y)$
g	g	$\{\text{cyan}, \text{yellow}\} \setminus \{g\}$	$\{\text{cyan}, \text{yellow}\} \setminus \{g\}$
α	$-\alpha$	$\pi - \alpha$	$\pi + \alpha$
$side$	$\{\text{right}, \text{left}\} \setminus \{side\}$	$\{\text{right}, \text{left}\} \setminus \{side\}$	$side$

Table 3.2: Spatial transformations of the different features and parameters used in a case, where (x, y) corresponds to a point on the field, g is the defending goal, α is an angle indicating the global heading of the robot, and $side$ is a direction parameter of a given action.

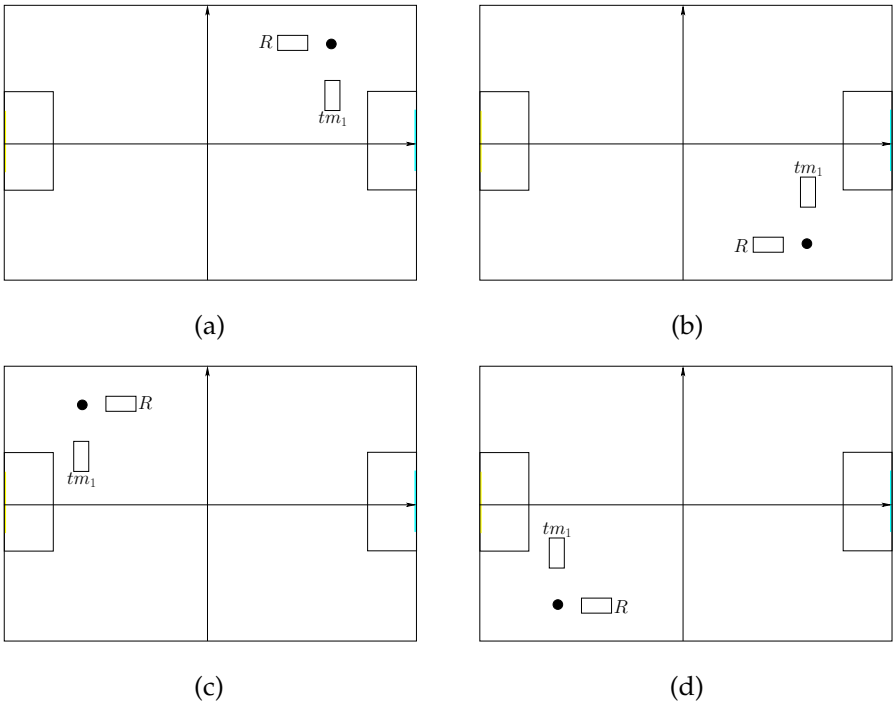


Figure 3.4: Example of the spatial transformations of a case: (a) original case; symmetric cases with respect to (b) the x axis, (c) the y axis and defending goal, and (d) the x and y axis and the defending goal.

in the rest of the case base is useless since those cases will not match the current problem at all.

Summarizing, the case base is composed of a set of cases manually created, where only the basic information is stored (global positions). When the case base is loaded, for each case we first compute its symmetric cases and then the relative coordinates of the features in each case. Therefore, we enlarge the original case base four times its original size, covering the whole field. We use an indexed list to classify the cases based on the defending goal feature to speed up the search, i.e. we only have to explore half of the case base instead of the complete one.

3.3 Case Retrieval

After having described the case definition and the description of the case base used in this work, in the remaining of this chapter we focus our attention in the retrieval step of the Case-Based Reasoning approach proposed.

Case retrieval is in general driven by a similarity measure between the new problem and the saved cases. We introduce a novel method to base the selection of the case to retrieve. We evaluate similarity along three important aspects: the similarity between the problem and the case, the cost of adapting the problem to the case, and the applicability of the solution of the case. Before explaining in more detail the similarity computation we first define two types of features:

- *controllable* features, i.e. position of the reference robot ant the teammates. (the robots can move to more appropriate positions if needed).
- *non-controllable* features, i.e. the ball's and opponents' positions, the defending goal, time and score (which we cannot directly modify).

The idea of separating the features into controllable and non-controllable is that a case can be retrieved if we can modify part of the current problem description in order to adapt it to the description of that case. Given the domain we are working on, the modification of the controllable features leads to a planning process where the system has to define how to reach the positions of the robots indicated in the retrieved case in order to reuse its solution.

3.3.1 Similarity Measure

Since the nature of the features' domain differs from one to another, we introduce different similarity functions to compare the features of a problem p and a case c. We first compute the similarities along each feature (assuming feature independence) and then we use an aggregation function to compute the overall similarity between the problem and the case. More precisely, for this measure we compare a subset of the non-controllable features (ball's position, time, score difference) leaving the opponents' positions for the applicability measure. The defending goal is not taken into account here since, as already mentioned in the case base description, we have pruned the search of cases from the case base in advance by only considering those with defending goal equal to the one described in problem p.

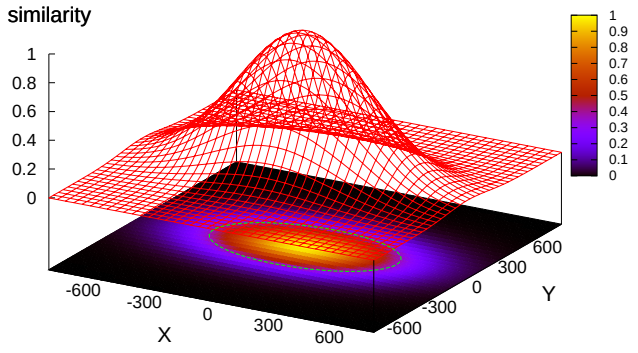


Figure 3.5: 2D Gaussian centered in the origin with $\tau_x = 450$ and $\tau_y = 250$. The solid ellipse on the plane XY corresponds to $G(x, y) = 0.367$.

Similarity of Spatial Features

We are interested in defining a continuous function that given two points in a Cartesian Plane indicates the degree of similarity based on the distance between the points. As larger the distance between two points is, the lower the similarity degree between them. We propose to use a Gaussian function, which besides fullfilling these properties, it is parametrized by its variance. We can use this parameter to model the maximum distance allowed to consider two points to have some degree of similarity. Since we are working in a two-dimensional space, we use a 2D Gaussian function, $G(x, y)$, to compute the degree of similarity between two points.

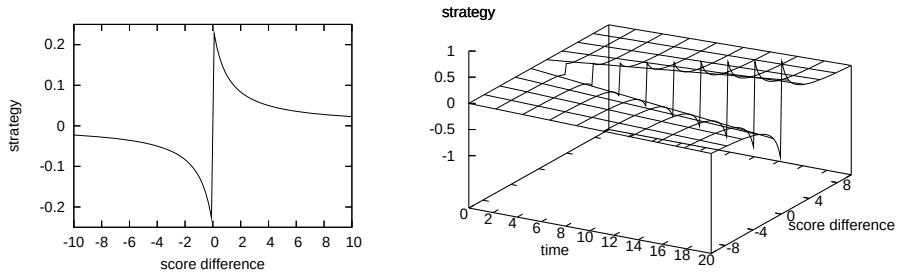
Hence, in the robot soccer domain, we define the similarity function for the ball feature as:

$$sim_B(x_p, y_p, x_c, y_c) = G(x_p - x_c, y_p - y_c) = exp\left(-\left[\left(\frac{x_p - x_c}{\tau_x^B}\right)^2 + \left(\frac{y_p - y_c}{\tau_y^B}\right)^2\right]\right)$$

where (x_p, y_p) corresponds to the ball's position in problem p, (x_c, y_c), to the ball's position in case c, and τ_x^B and τ_y^B the ball's scope indicated in the case as defined in Section 3.1.3. Figure 3.5 draws a 2D Gaussian function and its projection on the XY plane (sequence of ellipses with increasing radius as the similarity decreases). As we can observe, the Gaussian's projection with radius τ_x^B and τ_y^B represents the scope of the ball, i.e. the region within which we consider two points to be similar enough, and corresponds to $G(x, y) = 0.367$.

Similarity of Game Strategic Features

Defining a function that combines time and score is essential since they are closely related. As time passes, depending on the score of the game, we expect a more offensive or defensive behavior. We consider as critical situations those where the score difference S is minimum, i.e. when the chances for any of the



(a) (b)

Figure 3.6: (a) Strategy function for time $t = 5$. (b) Strategy function over time.

two teams of winning or losing the game are still high, and thus the strategy (or behavior) of the team might be decisive. We model the strategy for a 20 minutes game as:

$$strat(t,S) = \begin{cases} \frac{t}{20(S-1)} & \text{if } S < 0 \quad \text{(losing the game)} \\ \frac{t}{20} & \text{if } S = 0 \quad \text{(tie game)} \\ \frac{t}{20(S+1)} & \text{if } S > 0 \quad \text{(winning the game)} \end{cases}$$

where $strat(t,S) \in [-1..1]$, with -1 meaning a very offensive strategy and 1 meaning a very defensive strategy.

Figure 3.6a depicts the behavior of the team at time t. Positive and negative scoring differences mean that the team is winning or losing respectively. The higher the absolute value of S is, the lower the opportunity of changing the current score and the behavior of the team. For extreme values of S (in the interval $[-10..10]$, close to -10 or 10) the outcome of the function approaches zero. Otherwise, the function value indicates the degree of intensity, either for a defensive or an offensive behavior. Figure 3.6b shows the behavior of the function combining both variables. As time passes, the intensity of the strategy increases until reaching maximum values of 1 and -1, (defensive and offensive, respectively). These features are beyond robot soccer and are applicable to other games.

We define the similarity function for time and score difference as:

$$sim_{tS}(t_p, S_p, t_c, S_c) = 1 - |strat(t_p, S_p) - strat(t_c, S_c)|$$

where t_p and S_p corresponds to the time and score difference features in problem p and t_c and S_c, the features in case c.

Aggregation Function

After describing the similarity functions for the different features, we must define an aggregation function in order to compute the overall similarity between

the problem and the case. To this end, we tested four different functions: the mean, the weighted mean, the minimum and the harmonic mean. After evaluating their behavior, we concluded that:

- The minimum function results in a very restrictive aggregation function since the overall outcome is based only on the lowest value. Hence, low values penalize high values rapidly.
- Regarding the harmonic mean, for similar values, its behavior is closer to the mean function. While for disparate values, the lower values are highly considered and the outcome decreases (although not as much as with the minimum function) as more lower values are taken into account. On the contrary, the mean function rapidly increases the outcome for high values, and does not give enough importance to low values.
- Finally, the weighted mean does not differentiate between low and high values either, since the importance of each value is given by their weights. If a low value has a low weight and the rest of the values are all high, the outcome is slightly affected and results high anyway.

We are interested in obtaining an aggregation function that considers all values as much as possible but highlighting the lower ones. This is an important property as the values we are considering are similarities. Hence, if one of the features has a low similarity, the overall similarity has to reflect this fact decreasing its value. Based on the conclusions detailed above, we finally opt for the harmonic mean as the aggregation function:

$$h(x_1, ..., x_n) = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

where x_i corresponds to the individual similarity values of the features.

Therefore, within the domain we are working on, we define the similarity function between problem p and case c as:

$$sim(p, c) = \frac{2}{\frac{1}{sim_B} + \frac{1}{sim_{tS}}} = \frac{2 sim_B sim_{t_S}}{sim_B + sim_{tS}}$$

where sim_B and sim_{tS} are the similarity functions for the ball and time-score difference features respectively.

3.3.2 Cost Measure

This measure computes the cost of modifying the controllable features, i.e. the cost of adapting the current problem to the case. It is computed as a function of the distances between the positions of the robots in the problem and the adapted positions specified in the case after obtaining their correspondences. Next we separately present the new concepts introduced with this measure.

Adapted Positions

We refer to the adapted positions as those global locations where the robots should position in order to execute the solution of the case. In general, to compute them we transform the robots' relative coordinates to global coordinates,

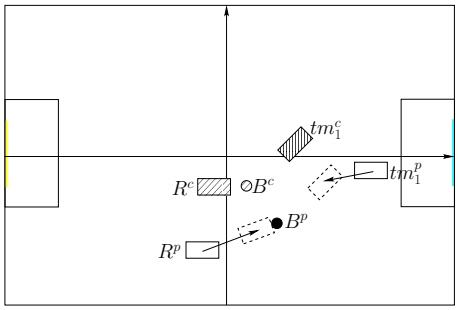


Figure 3.7: Case description (R^c, B^c, tm_1^c), and current problem description (R^p, B^p, tm_1^p). The dashed rectangles represent the adapted positions of the robots with respect to the ball's position described in the problem.

having the position of the ball in the problem as the reference point. But in fact, the adapted position of the first robot taking the ball in the gameplay is computed differently. It actually corresponds to the closest point to the ball within the straight line between the current robot's and ball's position. As we defined in the solution description, there is exclusively one robot whose first action is going after the ball. Hence, its first position on the field will be next to the ball. Figure 3.7 illustrates a simple adaptation example with two robots. Robot R is the one that controls the ball first, while tm_1 waits to receive the pass.

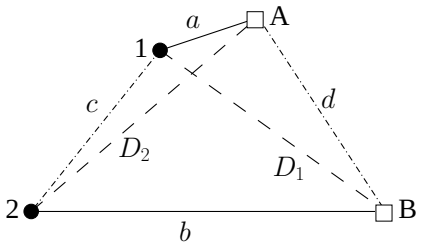
Robots' Correspondence

In order to compute the cost of adapting a problem to a case we must first determine the correspondence between the robots described in the problem and the ones described in the case, i.e. which robot r_i from the problem description corresponds to which robot r_j in the case description. Moreover, we must find the best match, i.e. the one that minimizes the cost, including one restriction: the distance between two points must be shorter than a given threshold, thr_c. Because of the domain's dynamism, the distances the robots have to travel must be limited since we cannot consent the robots to move from one point to another for long periods of time. In the meantime, the state of the world may have significantly changed and thus, the case may not be useful anymore.

In this work, since the maximum number of robots is small and fixed ($n = 3$, the goalie is always the same robot, so we only have to find the best match for the remaining three players of the team) we can easily compute all possible matches ($3! = 6$) without the need of an efficient search algorithm. However, as the number of robots becomes larger, the number of combinations increases exponentially. Thus, we require a search algorithm to optimize the search as we present later.

Cost Computation

We have studied two alternative functions to compute the cost: the sum of distances the robots have to travel and the maximum distance. The sum of distances aggregates all available distances in order to compute the outcome,



$$max(D_1, D_2) < max(a, b)$$
$$a + b < D_1 + D_2$$

Figure 3.8: Trapezoid layout of the matching between pairs $\{1, 2\}$ and $\{A, B\}$. The correspondence based on the sum function is represented by solid lines, while the max function is represented by the dashed ones.

while the max function is based only on one distance (the maximum), without considering the remaining ones. Therefore, we could define the sum as a more informed measure, where all values take part of the outcome. Moreover, interestingly the maximum distance function has a drawback when considering trapezoid (not necessarily having two parallel sides) configurations. Consider the layout depicted in Figure 3.8, where we have to find the optimal match between points $\{1, 2\}$ and $\{A, B\}$. We have depicted in solid lines the distances the robots would have to travel using the sum function, and in dashed lines, the distances using the max function. As we can observe, using the latter function the robots' paths intersect. This situation will happen whenever both trapezoid diagonals, D_1 and D_2, are shorter than the trapezoid larger side, b, and the matching points correspond to the end points of the middle sides, c and d. Figure 3.9 illustrates two more examples comparing the correspondence outcome when using both functions. It is clearly shown that we prefer to use the sum function instead of the max function.

Hence, in this domain we define the adaptation cost as the sum of distances the robots have to travel from their current locations to their adapted positions:

$$cost(p, c) = \sum_{i=1}^{n} dist(r_i, adaptPos_i)$$

where n is the number of robots that take part of the case solution, $dist$ is the Euclidian distance, r_i is the current position of robot i and $adaptPos_i$, the adapted position for robot i.

Optimizing the search

As mentioned previously, finding the robots' correspondence for a large number of robots requires an optimization algorithm to reduce the search complexity. Therefore, we propose a *Branch&Bound* (B&B) search algorithm in a binary tree for finding the best match between two robot configurations (the robots' layout in the problem and the layout in the case). Each node of the tree represents either the fact of considering a match between the pair (r_i, r_j), or the fact of not considering the match between this pair. In order to apply the algorithm we need to define a heuristic function to estimate the lower bound and to set the constraints that restrict the configurations in the nodes of the B&B algorithm (in our case just one constraint):

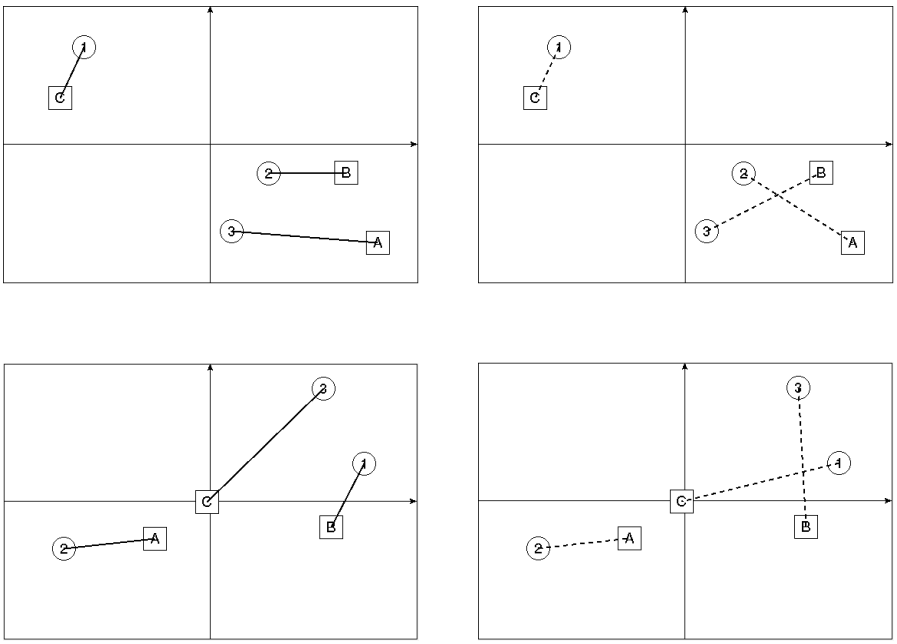


Figure 3.9: Two different layouts showing the advantage of the sum function (solid lines) compared to the max function (dashed lines).

- heuristic: the cost of all possible matches will always be larger or equal to the actual cost of a concrete mapping.

$$h(R_p, R_c) = \sum_j \min_i (\ dist(r_i, r_j))$$

  where R_p and R_c correspond to the set of robots positions in the problem (r_i) and the case (r_j) respectively, $i = 1..n$ for n robots in the problem description, $j = 1..m$ for m robots in the case description ($m \leq n$), and $dist$ is a function that returns the Euclidean distance between two points.

- constraint: the distance between two points must be shorter than a given threshold, thr_c.

Hence, we reduce the complexity of the search for n robots from $O(n!)$, all possible combinations, to $O(2^n)$, the complexity of the search in a binary tree.

3.3.3 Case Applicability Measure

From the set of features included in the problem description of a case, there is one that we have not yet included in any of the metrics described so far: the opponents feature. This last feature is precisely the one we focus on next, which is used to compute the applicability of a case.

Defining all possible configurations of opponents during a game, i.e. opponents' positions on the field, is impossible. Hence, achieving a complete case base composed of all possible situations would be not be feasible at all. For this reason we believe that a certain degree of generalization must be included

in the reasoning engine when dealing with this feature. Thus, we propose to combine rules and case similarity as follows:

- *free path rule*: the trajectory of the ball indicated in the case must be free of opponents to consider the evaluated case to be applicable.
- *opponent similarity*: the more opponents locations described in the problem coincide with the opponents locations described in the case, the higher the similarity between the problem and the case.

Free Path

Given a case, the *free path* corresponds to a function that indicates whether the trajectories the ball follows during the execution of the case solution is free of opponents or not.

Because of the ball's movement imprecision after a kick (either due to the robot's motion or the field's unevenness), the ball could end in different locations. Hence, we represent a trajectory by means of a fuzzy set whose membership function μ is a function that indicates the degree of membership of a point to the trajectory such that the closer the point to the center of the trajectory, the higher the membership. More precisely, this function is defined as a sequence of unidimensional Gaussians along a central axis, where the width of each Gaussian increases from a minimum radius to a maximum one defined in the trajectory. The projection of the μ function on the XY plane results in a trapezoid (Figure 3.10a). This trapezoid covers the area of the field where the ball could most likely go through according to the experimentation we have performed. We formally define the membership function for a trajectory t_j as:

$$\mu_{t_j}(x,y) = exp\left(- \left[\frac{y}{\rho(x, r_{min}, r_{max}, l)} \right]^2 \right)$$

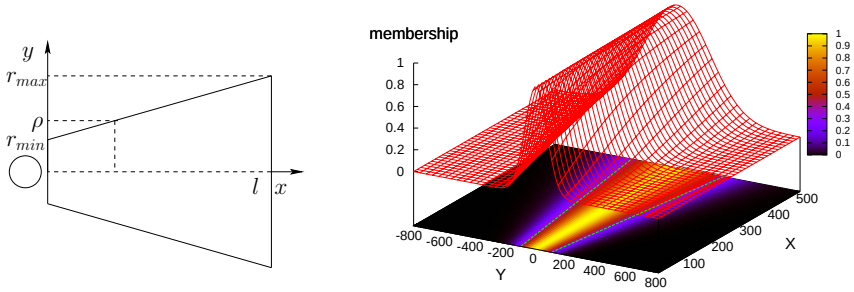
where r_{min}, r_{max} and l correspond to the minimum and maximum radius respectively, and the length of trajectory t_j. Finally, ρ is a linear function that indicates the radius of the Gaussian as a function of x. Figure 3.10b draws the membership function described.

We call ball path the sequence of trajectories the ball travels through in the solution of case c. Hence, we must verify that there are no opponents in the current state of the game (problem p to solve) located within any of the trajectories of the ball path. Figure 3.11a depicts an example. The initial position of the ball corresponds to B_1. After the first trajectory, t_1, the ball stops at B_2 and continues the second trajectory, t_2. Each trajectory results from a robot's kick. Formally, we define the free path function as:

$$free\_path(p,c) = 1 - \max_{t_j \in T}(\phi_{t_j}(Opp))$$

$$\phi_{t_j}(Opp) = \begin{cases} 1, & \exists opp_i \in Opp \ (\mu_{t_j}(opp_i) > thr_t) \\ 0, & \text{otherwise} \end{cases}$$

where T is the sequence of fuzzy trajectories (t_1 and t_2 in Figure 3.11a) described in case c, Opp is the set of opponents (each opponent opp_i is represented by the coordinates (x, y) of its position) in problem p, and $\mu_{t_j} \in [0..1]$ is



(a) (b)

Figure 3.10: (a) Ball's trajectory represented by an isosceles trapezoid defined by the minimum and maximum radius, and the trajectory length. (b) Membership function μ corresponding to the fuzzy trajectory with $r_{min} = 100$, $r_{max} = 300$ and $l = 500$. The solid lines on the plane XY correspond to $\mu(x, y) = 0.367$

the membership function. We consider that a point (x, y) is within a trajectory t_j if $\mu_{t_j}(x, y) > thr_t$, where $thr_t = 0.367$. The free path function could indicate the degree of path freedom using μ directly, instead of ϕ. In other words, we could define it as a fuzzy function as well.

Opponent's Similarity

Opponents on the field are modeled by means of elliptical regions as defined in Section 3.1. The opponent's similarity measure indicates the number of these regions that are occupied by at least one opponent described in the problem to solve. We call them restrictions. As more restrictions are satisfied, the more similar the state of the game and the case description are. Figure 3.11a shows an example where only one restriction is fulfilled, since only one region (reg_1) is occupied by at least one opponent. We define the opponent similarity function between a problem p and a case c as:

$$sim_{opp}(p, c) = |\{reg_j \mid reg_j \in Reg,\ \exists opp_i \in Opp\ (\Omega_{reg_j}(opp_i) > thr_{opp})\}|$$

$$\Omega_{reg_j}(opp_i) = G(x_i - x_j, y_i - y_j) = exp\left(-\left[\left(\frac{x_i - x_j}{\tau_x^j}\right)^2 + \left(\frac{y_i - y_j}{\tau_y^j}\right)^2\right]\right)$$

where Reg is the set of elliptic regions in case c (reg_1 and reg_2 in Figure 3.11a) and Opp is the set of opponents (opp_i is represented by the coordinates (x_i, y_i) of its position) described in problem p. Each region reg_j is defined by an ellipse with radius τ_x^j and τ_y^j centered in (x_j, y_j) (the opponent's scope indicated in the case as defined in Section 3.1.3). We define Ω as a 2D Gaussian function, where the projection on the XY plane for $\Omega(x, y) = 0.367$ corresponds to an elliptical region on the field with radius τ_x^j and τ_y^j. Thus, to consider that an opponent is within a given region we set the threshold thr_{opp} to 0.367. Once again, in this

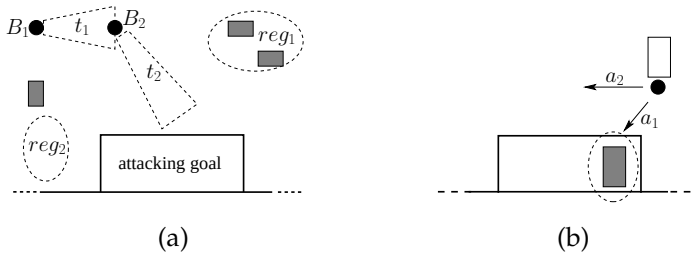


Figure 3.11: (a) Example of the ball's path performed by a pass between two players (robots are not depicted for simplicity). The dashed ellipses represent the opponents regions described in the case, and the gray rectangles, the opponents described in the problem to solve. (b) Opponent similarity as a justification of the action to perform. Action a_1 represents kicking towards the goal, while action a_2, kicking towards the robot's right side.

work we only consider if an opponent is within a region or not, but we could use the degree of occupation of a given region instead.

We must notice that although this measure is not crucial for the selection of a case as a candidate (as we describe in the next section), its importance lies in the candidates sorting process in order to select the retrieved one. While the free path function is fundamental when deciding whether a solution can be applicable or not, the opponent similarity measure can be seen as a justification of the actions defined in the case solution. Consider the example shown in Figure 3.11b. The robot in front of the ball can either kick towards the goal (action a_1), or kick towards its right (action a_2). The selection of one action or the other is basically given by the existence of an opponent in between the ball and the goal. Hence, if there is no opponent, it is clear that the most appropriate action to achieve the robot's objective is to kick towards the goal. But if an opponent (a goalie) is right in front, it makes more sense to try to move to a better position where the robot can then try some other action. Therefore, we can view the existence of an opponent as a justification for the selected action, in this example, kick towards the right.

3.3.4 Case Filtering

After describing the different measures, we now have to combine them to retrieve a case to solve the current state of the game (the new problem p). Because of the real time response requirements and the limited computational resources of the robots, we need to reduce as much as possible the search space. Therefore, for the retrieval process, we use a filtering mechanism. Each case c is evaluated using the measures explained in the previous sections. A case is rejected as soon as one of the conditions is not fulfilled, and we proceed with the next case. If a case fulfills all the conditions, then it becomes a candidate case. The filtering mechanism is shown in Algorithm 1. We first verify the ball similarity between the problem and the evaluated case (line 1), i.e. whether the current ball position is within the ball's scope indicated in the case ($thr_b = 0.367$). Next,

from lines 2 to 6 we check that every distance between the current robots' positions and their adapted positions is below the cost threshold ($thr_c = 1500$mm). Finally, if the ball's path is free of opponents (line 7) then we consider the evaluated case as a valid candidate (line 8).

Algorithm 1 IsCandidate(p, c)

```
1: if sim_B(B_p, B_c) > thr_b then
2:    for all (robot_p, robot_c) ∈ match(p, c) do
3:       if dist(robot_p, robot_c) > thr_c then
4:          return False
5:       end if
6:    end for
7:    if free_path(p, c) then
8:       return True
9:    else
10:      return False
11:   end if
12: else
13:   return False
14: end if
```

After evaluating all possible cases, we obtain a set of candidates. From this set we select only one using a sorting mechanism. The mechanism orders the candidate cases based on a set of criterion. Thus, given a set of candidates and the problem to solve p, each criterion orders the cases as follows:

$$ordered\_list = [c_i, c_j, ..., c_k]$$

where c_i, c_j, c_k are candidate cases, and the criteria are:

1. *number of fulfilled restrictions* (according to the opponent similarity): the more restrictions satisfied, the better.

$$sim_{opp}(p, c_i) \geq sim_{opp}(p, c_j) \geq \ldots \geq sim_{opp}(p, c_k)$$

2. *number of teammates that take part in the solution of the case*: we are interested in using cases with multiple robots implied in the solution so we can obtain a cooperative team behavior instead of an individualistic team, where only one robot takes part in the execution of actions. Therefore, the more teammates implied in the gameplay, the better.

$$num_{tm}(c_i) \geq num_{tm}(c_j) \geq \ldots \geq num_{tm}(c_k)$$

   where num_{tm} returns the number of teammates that take part in the case.

3. *adaptation cost*: the lower the cost, the better.

$$cost(p, c_i) \leq cost(p, c_j) \leq \ldots \leq cost(p, c_k)$$

4. *similarity*: the higher the similarity, the better.

$$sim(p, c_i) \geq sim(p, c_j) \geq \ldots \geq sim(p, c_k)$$

5. *similarity intervals*: case classification in different subsets based on their similarity. In this work we classify the cases in four similarity intervals (the intervals can be easily modified based on the requirements of the domain where the approach is applied):

   - very high similarity: $H = [0.8, 1.0]$,
   - high similarity: $h = [0.6, 0.8)$,
   - low similarity: $l = [0.4, 0.6)$, and
   - very low similarity: $L = (0.0, 0.4)$.

   A further sorting process within each interval based on some other criterion must be then performed. For this work we chose the adaptation cost criterion. The goal of the similarity intervals is to have a trade-off between the similarity and the adaptation cost. Having a case with high similarity is as important as having cases with low cost. Therefore, even if the similarity is very high, if its cost is also too high, it is more interesting to select a less similar case within the same interval, but with lower cost.

   Finally we obtain a flat[1] list:

   $$ordered\_list = flat([int_H, int_h, int_l, int_L]) = [c_i, c_j, ..., c_k]$$

   where $int_s = [c_i, c_j, ...]$ is an ordered list of cases based on the cost criterion, $s \in \{H, h, l, L\}$ stands for the similarity interval, and flat is a function that returns a flat list.

Although we have presented five criteria to sort the candidates, the designer may freely create any other alternative criterion that fits better to the domain the approach is focused on. The next decision point is whether the candidates ranking is based on a single criterion or based on a set of criteria. In the latter case, the designer not only has to select which criteria to use, but also the order in which each criterion will be applied.

Finally, after sorting the candidates, either based on a single criterion or using multiple criteria, the most adequate case to retrieve corresponds to the first element of the ordered list:

$$ret\_case = first(ordered\_list)$$

The overall retrieval process is presented in Algorithm 2. Its inputs are the problem to solve, p, and the case base, CB.

In this work we opted to employ the combination of the five criteria previously described in the sorting process. Hence, we must decide which combination is the most appropriate to use in this domain, i.e. in which order to apply each individual criterion. In the next section we study different candidate sorting functions varying the order of the criteria used to rank the cases.

[1] We define a flat list as a list with one single level, i.e. no nested lists.

Algorithm 2 Retrieve(p, CB)

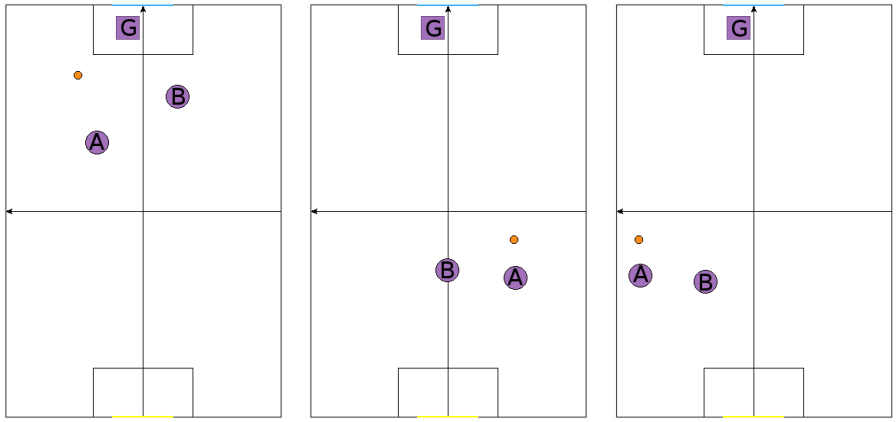
```
1: for c in CB do
2:   if IsCandidate(p, c) then
3:     candidates ← append(c, candidates)
4:   end if
5: end for
6: ordered_list ← sort(candidates)
7: ret_case ← first(ordered_list)
8: return ret_case
```

3.3.5 Experiments

The goal of the experimentation is to determine the most suitable criteria sequence to employ when sorting the candidate cases obtained after the filtering process. To this end, we defined three sequences that apply the different criteria described in the previous section in the following order:

- sorting function 1:
  1. number of teammates
  2. similarity intervals
  3. number of fulfilled restrictions
  4. adaptation cost
  5. similarity
- sorting function 2:
  1. number of fulfilled restrictions
  2. number of teammates
  3. similarity intervals
  4. adaptation cost
  5. similarity
- sorting function 3:
  1. number of teammates
  2. number of fulfilled restrictions
  3. similarity intervals
  4. adaptation cost
  5. similarity

The experiments are performed in simulation only. The case base is composed of 33 hand coded cases (hence, 132 cases in total after the generation of their symmetric cases). The cases can be classified as single or multiple. The former refers to those cases where only one robot takes part of the case, while the latter, cases where two robots take part of the case. Furthermore, cases can also be grouped based on the regions of the field they cover and the purpose of the case. Hence, we organize the cases as follows:



(a) (b) (c)

Figure 3.12: From left to right scenarios 1, 2 and 3 used during the experimentation.

- back of the field (with or without opponents)
- middle of the field (with or without opponents)
- side of the field (with or without opponents)
- corner of the field (with or without opponents)
- in front of the goal (with or without goalie)
- in diagonal of the goal (with or without goalie)

A trial starts positioning the robots (two robots from one team, and a goalie as an opponent) and the ball in a fixed location. Next, the robots from the same team start playing using the CBR approach, i.e. retrieving cases and executing their solutions (we detail the reuse step in Chapter 4), while the goalie randomly moves within its penalty box. The aim of the robots performing the CBR approach is to score a goal. A trial ends when either the ball goes out of the field or the goalie touches it.

Each experiment consists of 500 trials using the same sorting function and the same layout, i.e. initial positions of robots and ball. Figure 3.12 depicts the three scenarios we designed for the experimentation. Each experiment is repeated for every scenario and every sorting function defined previously. In this experimentation stage we are not interested in evaluating the outcome of the trial in terms of goals scored, goals stopped, etc., but in observing the behavior of the sorting criteria defined above based on the cases they propose as solutions to the different states of the game. Therefore, we computed two measures: number of different retrieved cases during a complete experiment (500 trials), and the average time for a trial to finish. Table 3.3 summarizes the results obtained for each configuration. We have also computed the number of cases retrieved per trial and then ordered them based on this measure. Figure 3.13 plots the outcome for the three scenarios.

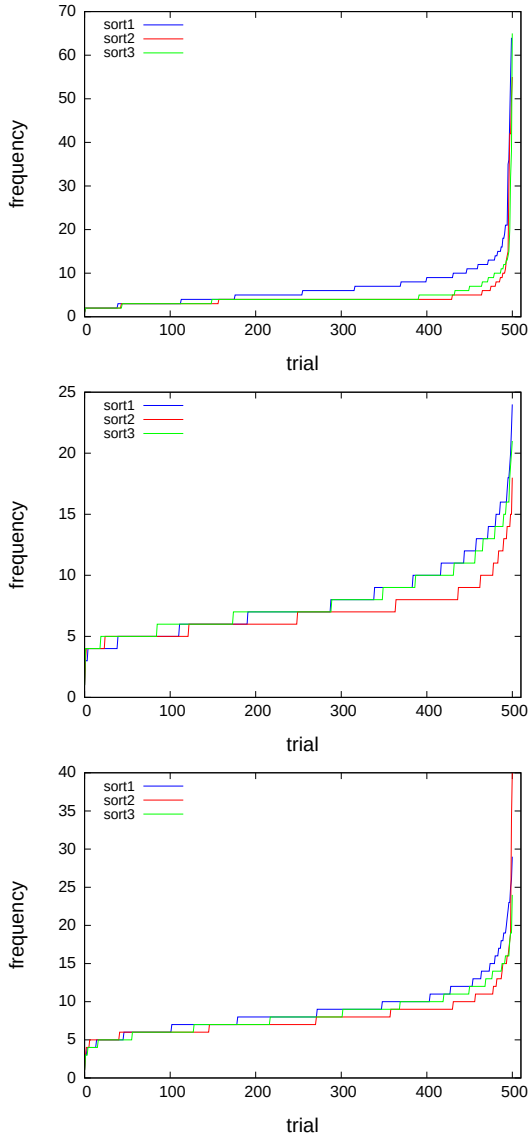


Figure 3.13: Trials sorted by frequency of cases retrieved in each scenario.

scenario	sort func	#diff cases	time (sec)
1	1	40	15.90
	2	16	7.51
	3	24	8.56
2	1	44	30.01
	2	31	22.63
	3	41	28.78
3	1	44	32.22
	2	40	24.44
	3	46	28.85

Table 3.3: Number of different cases reused in the experiments and average time per trial.

We can immediately observe that the second sorting function performs the fastest compared to the other two, and at the same time, makes use of less cases during the experimentation (shown both in the table and in the figures). These two facts are directly related since the reason why it takes less time for a trial to end and uses less cases is because the cases retrieved with the second sorting function were more appropriate than those retrieved by the other functions. To confirm this statement we studied the number of times each different case was completely executed[2] by each function. Figure 3.14 depicts the most relevant outcomes for all three scenarios (for visibility reasons we cannot show all the retrieved cases; we omitted those with very low frequency). Each letter corresponds to a different case. After analyzing the results in detail we conclude that:

- In general, cases B, C, D, E, V and W are retrieved by the three functions in every scenario. These cases correspond to situations where a robot is in front or at one side of the goal with or witout the goalie. It is easy to see that whenever the robots get near the goal they would need at some point to use any of these cases to try to score avoiding the goalie in case it was blocking the goal. Case A is often retrieved as well in the different scenarios. This case represents a robot in the middle front of the field having a short kick to the front. It is a very common situation that can take place in any of the scenarios at any moment.

- Scenarios 2 and 3 present a layout where the robots are initially positioned in the middle back of the field. Hence, it is necessary to consider cases located in this region, as cases G, H, I, J, K and L are. As we can notice, case I has a very high frequency in scenario 2. This is due to the fact that it is always the first case retrieved given the initial configuration. Thus, all trials always start retrieving this case to move the ball to the front region of the field.

- Case N is noticeably frequent in scenario 3. In this situation the ball is positioned in an edge of the field (side of the field). Once again, since

[2]We do not consider cases that were aborted during their execution. The execution of a case may be aborted when the state of the environment does not match anymore the case retrieved, i.e. the case is no longer applicable.

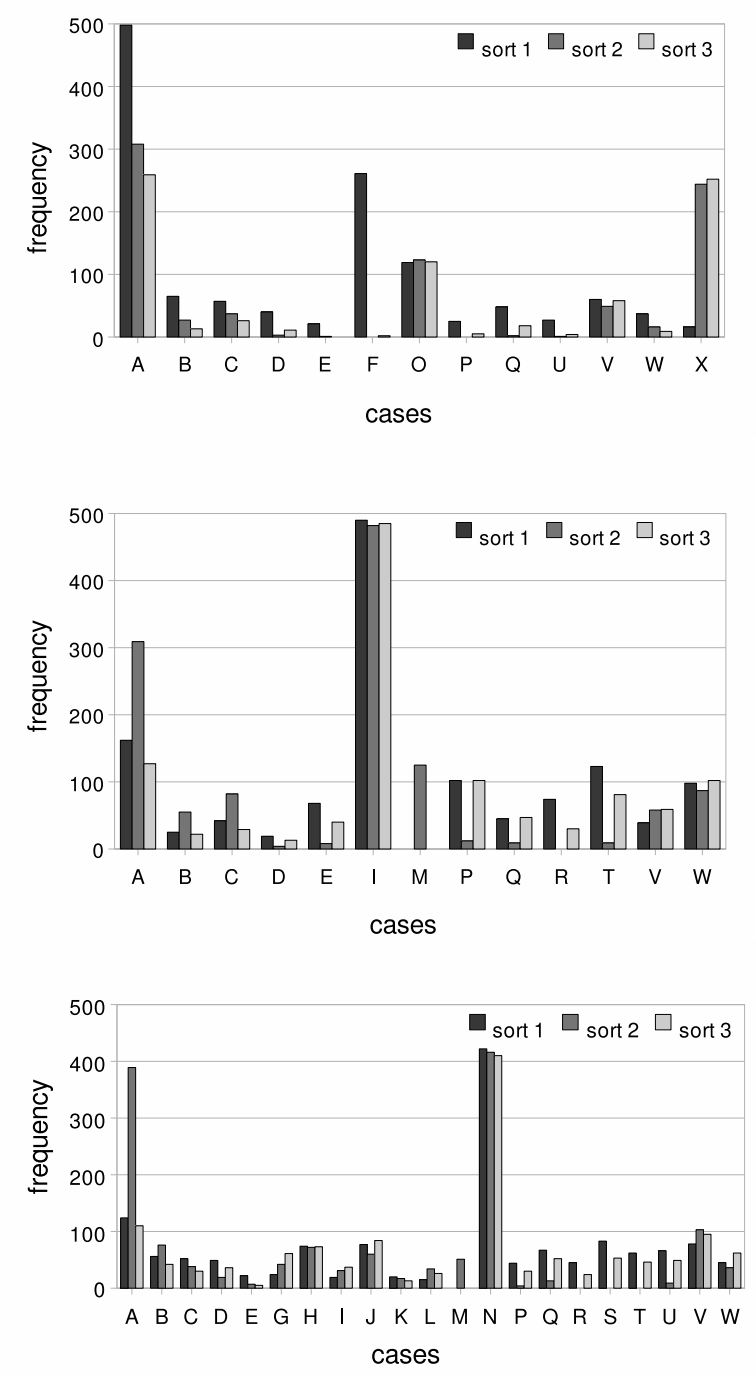


Figure 3.14: Case frequency for each scenario. From top to down, scenario 1, 2 and 3.

it perfectly matches the initial configuration of the scenario, this case is always retrieved when the trial starts. The gameplay consists in passing the ball to the other robot (teammate) and this latter kicks the ball towards the middle of the field. From that point on, middle cases as G, H, I, J, K and L are most likely to be retrieved depending on the location where the ball ends after the last kick.

- In scenario 1 case O is regularly retrieved by all three functions. It corresponds to a situation where both robots are near the goal, one near to the ball and having the goalie in front of it blocking the ball, and the other robot on one side with a free path to score. Hence, a pass towards the free robot is the most suitable action, so it can then try to score.

- Cases P, Q, R, S, T and U are specially retrieved by functions *sort1* and *sort3* in all three scenarios. They correspond to cases with two robots in the middle of the field with an opponent in front of the ball and another one to one side of it. Therefore, the solution corresponds to a side pass (the side free of opponents) to the other teammate.

  Both sorting functions give the highest priority to cases with more than one robot (criterion 2). Therefore, although a short kick could be enough to get closer to the goal, it is not preferred because that would imply having only one robot in the gameplay (individualistic behavior), instead of having a pass between teammates (cooperative behavior). Although for many adversarial situations cooperation among teammates is desired, in this situation is not needed since the robots are alone in the middle of the field without any opponents near (the only opponent is the goalie, but it remains in its penalty box). Having passes in these situations can even be disadvantageous because the pass could fail moving the ball further from the goal, instead of getting closer to it.

  On the contrary, function *sort2* first considers the number of restrictions fulfilled (criterion 1), and then the number of players (criterion 2). Therefore, a case with simple kick to move the ball forward and no opponents would be ranked first, while the cases mentioned above would be ranked lower because they do not fulfill any of the restrictions, i.e. regions occupied by opponents.

- We pay special attention to the first scenario, where function *sort1* often retrieves case F, while functions *sort2* and *sort3*, retrieve case X instead. In both cases the ball is diagonally located with respect to the goal, but in the first case, there is a goalie, while in the latter, there is not (Figure 3.11b draws this situation). Hence, the main difference between both cases is the fact of having or not an opponent, i.e. the number of fulfilled restrictions (criterion 1). As an example, consider the situation where there is no goalie in between the ball and the goal and that the similarity of case F is much higher than the similarity of case X. Since function *sort1* considers the restrictions fulfilled (criterion 1) after the similarity intervals (criterion 5), it is more likely that it would retrieve case F instead of case X. On the contrary, the other two functions consider criterion 1 before criterion 5, and therefore, they first maximize the number of fulfilled restrictions, and then the similarity. Since in this example there are no opponents, retrieving case X is the most adequate.

Summarizing, in general all three functions worked well, achieving the goal of the experiments and retrieving the appropriate cases to move the robots in a reasonable and expected way (first four points of the previous analysis). However, after evaluating the experiments in more detail we can deduce that criterion 1, i.e. number of fulfilled restrictions, is fundamental for ranking the cases and considering it in first place is indispensable (mainly for the last two points discussed). Moreover, the right selection of the case to reuse has direct impact on the time invested in successfully achieving the goal. Therefore, we conclude that function *sort2* is the most suitable one to use in the retrieval process for the remaining experiments of this work.

3.4 Conclusions and Future Work

In this chapter we have described the first step of the CBR cycle, i.e. the retrieval step. To this end, we have introduced the case structure, which corresponds to features that describe the environment, as well as derived features used during the different computational processes of the case retrieval. We have also classified the features in two sets: controllable and non-controllable features. The distinction between both types lies in the capability of the system to modify the values of the features in the problem to solve, in order to increase their similarity with the evaluated case.

The initial case base is composed of a set of hand-coded cases. Once the case library is loaded, it is automatically enlarged exploiting the symmetric properties of the case description through spatial transformations. The case base is divided in two sets of cases depending on the defending goal feature, reducing the search space during retrieval.

A case is considered as a candidate solution based on three measures: the similarity measure, which corresponds to the aggregation of individual similarities; the cost measure, which indicates the cost of modifying the controllable features; and the applicability measure, which verifies if the case is applicable or not based on the opponents' positions in the problem to solve. This latter measure is, in turn, composed of two different functions: the free path function that indicates whether the ball's trajectories are free of opponents or not, and the opponent similarity, which is used to reinforce the similarity between the problem and the case.

A filtering mechanism is applied to obtain a set of candidate cases to speed up the search. These candidates are sorted based on different criteria and the first case of the list corresponds to the retrieved case to use afterwards in the reuse step. We have shown empirical experiments in simulation to test the efficiency of the retrieval process.

As future work, several improvements to the current model that could be addressed are the following (although not limited to them):

- The cost function should also reflect the existence of obstacles in the robots' paths (in this domain, obstacles correspond to opponents) to reach the adapted positions. Thus, the cost could increase depending on the obstacles found in the planned trajectory. However, we must not forget that in this domain we are dealing with dynamic obstacles, and therefore, the cost at retrieval time can differ from the cost at reuse time. In

other words, while the robot moves to a given point, the opponents also modify their locations either moving from/to the robot's path, altering the initial computed cost. As proposed in [57] other features that could be considered in the cost computation are the orientation of the robots, velocities, etc.

- As already mentioned in Section 3.3.3 we could make use of the fuzzy representations of the free path function and the opponent similarity function instead of the boolean function used in the current model. Hence, the retrieval step would be more flexible when considering the applicability of a case.

- Cases may only differ in their solution description. The retrieval process presented in this chapter only evaluates cases based on their problem description. Therefore, two cases may result with the same ranking score. In order to discriminate between similar cases, probably the outcome evaluation of the reused solution should be introduced in the case description. The problem here is how to evaluate the outcome of the reused case, i.e. how well the execution of the case was performed? Is the outcome altered due to external factors (because of the world dynamics) or not?

- Through time, cases are retrieved and reused one after another. After the execution of a given case, there is a correlation between the last reused case and the candidate cases of the following retrieval step. Thus, if the last reused case corresponded to a given region of the field, it is most likely that cases within or near that region will be considered as candidate cases in the next retrieval step. On the contrary, cases describing situations in further locations, are less probably to have any similarity with the current state. We believe, then, that it would be interesting to include these relations in the case description. A simple way is to maintain for each case c_i, a list of cases that were retrieved after its execution (of case c_i). Those cases in the list with higher frequency correspond to the most likely cases to retrieve in the next CBR cycle. With this information we could reduce the search space, and instead of looking in the whole case base, the retrieval step could first center the search to a subset of cases. If no case were found, then it should search in the remaining of the case base. Moreover, we could obtain sequences of linked cases, where given a case indicates the most probable next case to retrieve, and in turn, this latter indicates its next probable case to retrieve, and so on. Thus, patterns of cases can be obtained and analyzed afterwards for instance to evaluate the overall behavior of the team, or used as predictions of future states.

Chapter 4

Case Reuse through a Multi-Robot System

We focus this chapter on the second step of the Case-Based Reasoning approach: the reuse step. In the previous chapter we described the retrieval step, including the case definition, the different measures used for computing the case similarity, and the retrieval process itself. Hence, after obtaining the retrieved case, we must center our attention on how to reuse this case.

In most case-based applications the reuse step consist in proposing a solution (or adapted solution) to the user who would then make use of this information as she requires it. In this work, the user querying the case-based system is not a single user, but a team of robots. The solution proposed by the system consists of a set of sequences of actions that each robot of the team should reuse (execute). Moreover, the execution must be done in a coordinated manner.

The first part of the chapter is devoted to the internal robot's architecture, while the second one is focused on the multi-robot system, the coordination mechanism and the case reuse. More precisely, we describe how we have integrated the retrieval and reuse steps of the CBR approach within a team of robots.

4.1 Robot Architecture

We define our robot architecture as a hybrid architecture with three layers (Figure 4.1):

- *deliberative system*, in charge of making the high level decisions. Two modules coexist in this layer: the case-based reasoning engine (CBR module), and the region-based algorithm (RBA module).
- *executive system*, responsible for the execution of plans or actions indicated by the deliberative system. The system is composed of the behaviors module and the perception module.
- *actuators/sensors*: they correspond to the physical components of the robot. Thus, the motors (legs, head and neck) correspond to the actuators, and

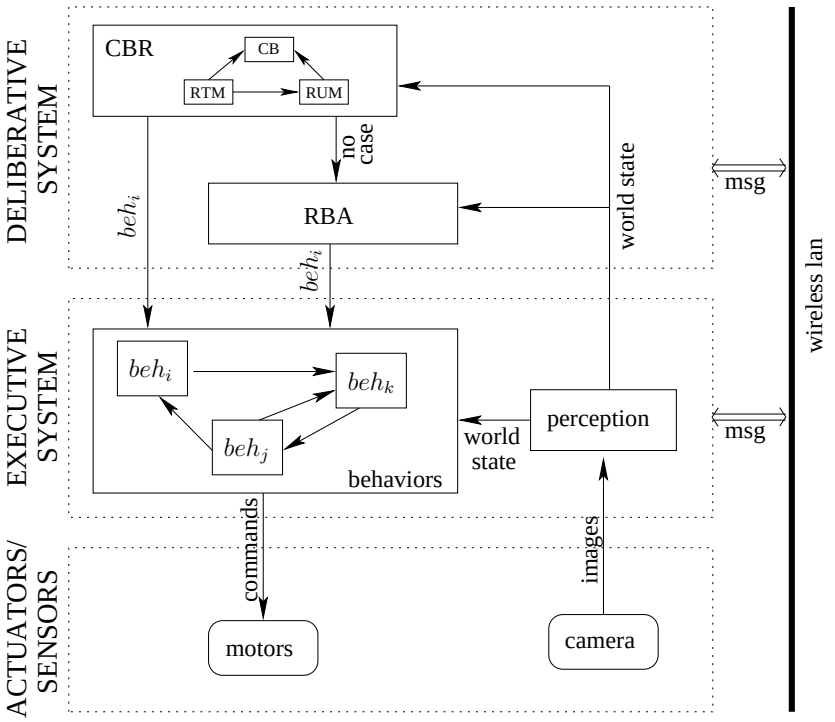


Figure 4.1: Robot architecture.

the camera correspond to the sensors. The actuators receive the low level commands from the executive system, i.e. from the behaviors, while the camera sends the images to the perception module.

We also include the communication channel between the internal robot architecture and the rest of the robots, i.e. the wireless lan. Robots are constantly exchanging messages among them. Two main types of messages are transmitted:

- periodical messages, which contain information about the robot's beliefs and state, such as teammate's Id, teammate's position, distance to the ball, etc.
- explicit messages, which are used to transmit specific information, such as an ABORT message.

The periodical information is included in the current robot's world model, while the specific information is queried by the specific modules that are expecting it. The modules are also able to send messages to the remaining robots.

Most part of the implementation of this architecture has been realized using the Carnegie Mellon's team code, CMDash'06. More precisely, we have used the executive system and the RBA module. We have extended their implementation to include the CBR module and its relation with the rest of the architecture.

4.1.1 Deliberative System

As many other robot architectures, the deliberative system is in charge of making the high level decisions in order to achieve a given goal. Moreover, in our work the system is composed of two hierarchical modules: the CBR module and the RBA module.

CBR Module

The CBR module contains the Case-Based Reasoning system. It is in charge of proposing a solution for the current state of the game and monitoring its execution afterwards, i.e. it is responsible for the case retrieval and the case reuse. Hence, this module is composed of two components: the retrieve module (RTM) and the reuse module (RUM). Since both components need access to the case base, we also include it in the module.

The retrieval process defined in Section 3.3 takes place in the retrieve module (RTM). Hence, given the current state of the game (the world model obtained through the perception module), the RTM proposes a case as a solution to that problem. It not only indicates the case identifier to the reuse module, but also the matching between robots (so each robot knows which sequence of actions indicated in the case to perform).

The reuse module (RUM) is triggered when it receives the information about the retrieved case: the case identifier and the robots' correspondence. It is in charge of first moving the robot towards its adapted position, and next, executing and monitoring the sequence of actions indicated in the retrieved case. If at any point, the case is not applicable anymore (we must recall that we are working in a dynamic domain and therefore, the state changes constantly), then the execution of the case is aborted. We will go through the reuse step in more detail in the next section since it implies the multi-robot system and we first need to define some new concepts before continuing with the reuse description.

RBA Module

The region-based algorithm (RBA) is a general behavior-based algorithm that activates different behaviors based on the region of the field where the ball is located at a given time. It could be seen as a rule-based approach combined with a decision tree algorithm. Each rule corresponds to a region of the field. Thus, it defines a small set of rules of type: *if ball in region* reg_i*, then apply behavior* beh_i. Each behavior is defined as a procedural process where typically the robot will first approach the ball, and then based on a decision tree, it will try either to get closer to the attacking goal or to score if it is close enough to the goal.

The RBA also includes an implicit coordination mechanism to avoid having the robots go after the ball at the same time. Thus, when a robot possesses the ball, it informs the others so they move away from its path. In general, the robots back up from their current positions on the field. The algorithm also includes a set of roles that are assigned to each robot so they cover different regions of the field. For instance, a defender stays at the back of the field, while a striker remains in the front of the field waiting for an opportunity to attack when the ball gets within its region.

Combining the Modules

As we can see, the region-based approach describes a general player's behavior taking into account only local information for fast and reactive response to the current state of the world. In other words, although it has some degree of deliberation, it still lacks a broader view of the state of the game to try to achieve more ambitious strategies including collective actions with teammates. It is mainly focused on taking the ball and moving towards the attacking goal in an individualistic way, which of course, is also beneficial for certain situations where a fast attack is fundamental.

On the contrary, the case-based approach uses a more complete model of the state of the world considering, not only the ball's position, but also other aspects such as the positions of all the robots and evaluating the appropriateness of executing a set of actions.

Hence, in the proposed deliberative system we combine both strategies, a more deliberative one (the CBR approach) with a more general and reactive one (the RBA approach), in a way that the latter one is triggered only when the former does not find an appropriate solution for the current problem, i.e. there is no case that matches the current problem well enough to be of use.

4.1.2 Executive System

This system is responsible for the execution of the actions indicated by the deliberative system (in this case the actions correspond to behaviors), and the world model generation.

A behavior is a sequence of actions that a robot executes to perform a task. There is a wide range of behaviors, varying from very simple tasks, such as "walk forward", to very complex ones, such as "move away from ball". Usually complex behaviors are compositions of simpler behaviors, or make use of them for specific substasks, i.e. they have a hierarchical structure. For instance, in the "move away from ball" behavior, the high level behavior must constantly know the position of the ball. Hence, it makes use of the "track ball" low level behavior. In general all behaviors, except for the very simple ones, make use of the world model to know where the robots and the ball are located on the field.

The perception module is in charge of building the world model of the robot, i.e. the robot's beliefs of the state of the world (its position, the ball's position, etc.). Although the perception system is much more complex than the one we show here, for the purpose of this work we note two main sources of incoming information from the outside world: the images from the camera and the messages sent by other robots. The module processes the images sent by the camera and infers the state of the world (objects positions). It also incorporates the incoming information from the teammates, i.e. the messages sent through the wireless network. Hence, the world model of the robot not only considers its own perception, but the teammates' perceptions as well.

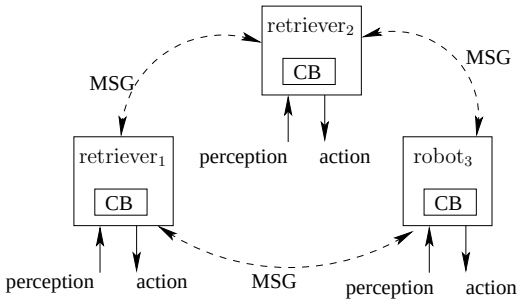


Figure 4.2: Multi-robot system for $n = 3$ robots and $k = 2$ *retrievers*. Each robot has a library of cases (CB).

4.2 Multi-Robot System and Case Reuse

After detailing the internal robots' architecture, we next describe the architecture for our multi-robot system integrating the retrieval and reuse steps of the CBR approach.

The multi-robot system is composed of n robots. All robots interact with the environment and with each other, i.e. they perceive the world, they perform actions and they send messages (MSG) to each other to coordinate (e.g. retrieved case, match, abort execution,...) and to exchange information about their internal state (e.g. retrieving, adapting, reusing,...). There is no external system observing the complete environment where the robots execute their tasks, nor a centralized system to collect the incoming information from the robots to make decisions and to organize the task. Therefore, the team as a whole must decide how to fulfill the task collaborating with each other, or even exchanging useful information that some of the robots of the team may not perceive. This characteristic is common in those domains where human access is not feasible, and therefore, installing a centralized system is impracticable. Some examples of such systems are planetary explorations or disaster rescue operations.

We distinguish a subset of k ($1 \leq k \leq n$) robots, called *retrievers*. These robots are capable of retrieving cases as new problems arise. We refer as *executors* to the rest of the robots, i.e. those that are not retrieves and are only capable of reusing the solution of cases. All robots, retrievers and executors, have a copy of the same case base so they can gather the information needed during the case reuse. Figure 4.2 shows the described multi-robot system.

Given a new problem to solve, the first step of the process is to decide which of the retriever robots is going to actually retrieve a case to solve it (since only one case can be reused at a time). The most appropriate robot to perform this task should be the one that has the most accurate information about the environment. From the set of features described in a case, the only feature that might have different values from one robot to another is the ball's position. Moreover, this is the most important feature in order to retrieve the correct case and we must ensure as less uncertainty as possible. The remaining features are either common to all the robots, i.e. robots' positions, or given by an external system, i.e. defending goal, the score and time of the game. Therefore, we propose that the robot retrieving the case should be the closest to the ball, since its information will be the most accurate (the further a robot is from an

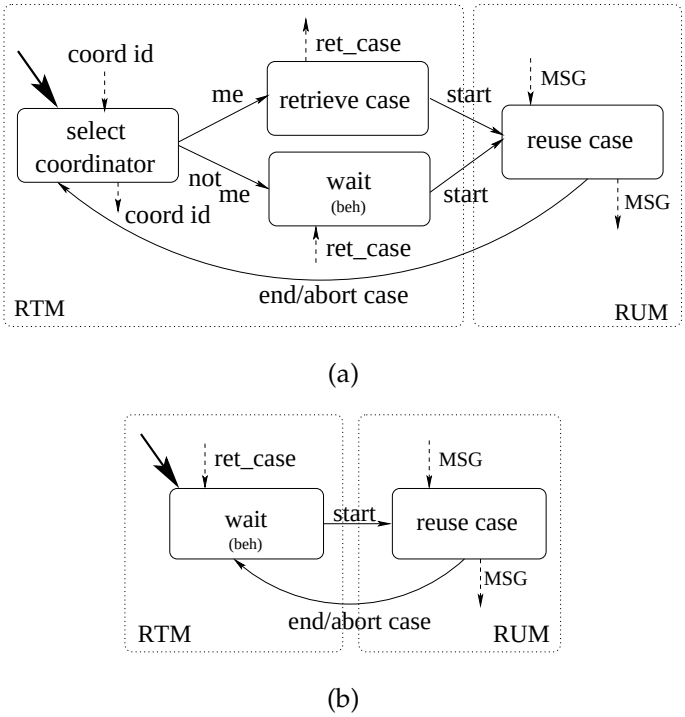


Figure 4.3: Finite state machine for (a) the *retriever* robots and (b) the *executor* robots. Solid arrows indicate transitions, while dashed ones correspond to messages sent between robots.

object, the higher the uncertainty about the object's information). From now on, we will refer to this robot as the *coordinator*. While the selected coordinator is retrieving a case, the remaining robots wait (either remaining in their current positions or performing some other behavior). Figure 4.3 depicts the finite state machines for the retriever robots and for the executor robots.

Since we are working with a distributed system, the robots may have different information about each other at a given time. Their beliefs about the state of the world are constantly updated. They are also constantly sending messages about their current internal beliefs (robot's Id, position, ball's position, etc.) to the rest of the robots. As a consequence, we cannot ensure that all robots agree on who is the one closest to the ball at a given time. To solve this issue, only one robot is responsible for selecting the coordinator. In order to have a robust system (robots may crash, or be removed due to a penalty), the robot performing this task is always the one with lower Id among those present in the game (since each robot has a unique fixed Id). Once it selects the coordinator, it sends a message to all the robots indicating the Id of the new coordinator.

After the coordinator is selected, it retrieves a case according to the process described in Section 3.3.4 and informs the rest of the team the case to reuse. It also informs the correspondences between the robots in the current problem and the robots in the retrieved case (so they know what actions to execute accessing their case bases. The correspondences are obtained following the pro-

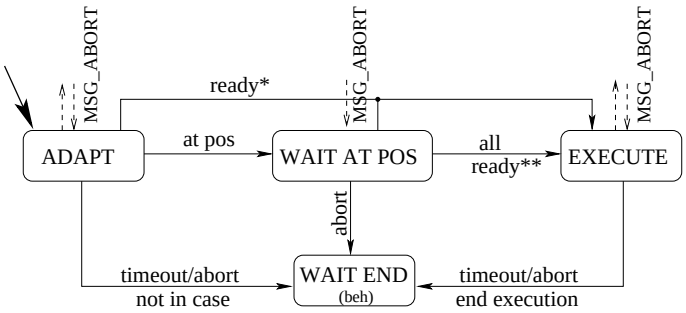


Figure 4.4: Finite state machine for the case execution (*independent positioning strategy, **dependent positioning strategy).

cedure detailed in Section 3.3.2). This process takes place in the RTM module of the robot's internal architecture.

Then the case execution begins. Figure 4.4 describes the finite state machine for the case reuse process (corresponding to the RUM module within the robot's architecture). First, all robots that take part of the solution of the case start moving to their adapted positions (ADAPT state). As explained in Section 3.3.2 these positions correspond to relative positions indicated in the case with respect to the current ball's position. Hence, a robot can easily derive the location where it should move using the matching information transmitted by the coordinator. The robots that do not take part of the case reuse remain in the WAIT END state (either waiting at their positions or performing an alternative behavior) until the execution ends. At this point we can choose between two strategies:

- *independent positioning*: the robots move towards their adapted positions independently from each other until reaching it. Once they reach their adapted positions, they send a message to the coordinator. In this case, the coordinator is in charge of receiving the messages from the robots indicating they are at their adapted positions, and then sending a message to all the robots to start executing the gameplay. Hence, the case reuse only starts when all robots arrive to their initial positions.

  In Figure 4.4 this strategy would correspond to the WAIT AT POS state and switching to the EXECUTE state when all robots are ready waiting at their adapted positions.

- *dependent positioning*: the robots do not have to wait for all robots to reach their adapted positions. As we described in Section 3.1.2, there is always a robot that goes first to get the ball. All robots know who this robot is and also know in which state all robots are (adapting, reusing, waiting, etc.). Hence, they only wait for this robot to arrive to its adapted position and immediately start executing the gameplay, even if they have not reached their own adapted positions yet. In other words, the robots' positioning depends on a given robot (i.e. the one going first to the ball).

  In the finite state machine depicted in Figure 4.4 the robot can either transit from the ADAPT state or the WAIT AT POS state to the EXECUTE state

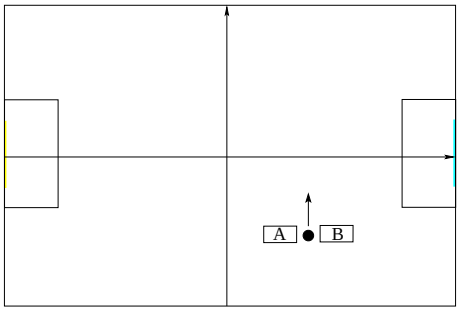


Figure 4.5: Kick adaptation during the case reuse. The arrow represents the ball direction indicated in the case. Thus, robot A should perform a left kick, while robot B, a right kick.

when the robot getting the ball first has already reached its adapted position. In the former situation, switching from the ADAPT state, the robot is still moving towards its adapted position when the transition takes place. In the latter situation, switching from the WAIT AT POS state, the robot has reached its adapted position but the robot getting the ball first has not yet.

In an adversarial game domain, as the one this work is focused on, we realized that it is more advantageous to use the dependent strategy than the independent one. Otherwise, while the robots are all moving to their adapted positions, the opponents may steal the ball. An independent strategy is more convenient to use in other domains where reaching the initial positions is crucial to successfully fulfill the task.

Either using the independent or the dependent positioning strategy, the execution of the solution of a case starts (state EXECUTE in Figure 4.4) and all robots perform their sequences of actions. Each action corresponds to a behavior in the executive system (Figure 4.1). A last adaptation process takes place when the action corresponds to a kick. In this case, depending on the direction from where the robot reaches the ball, the robot might perform the symmetric of the kick indicated in the case solution to move the ball towards the expected direction. Figure 4.5 shows an example. As we can observe, the kick varies if the robot is coming from the front (situation described by robot A in the figure) or from the back (robot B representation in the figure) of the ball. The execution continues until all robots finish their sequences of actions.

Finally, they report to the coordinator that they finished the execution and wait for the rest of the robots to end (WAIT END state in Figure 4.4). In this state the robots may perform some other behavior while waiting for the other robots to end the execution. When the coordinator receives all messages, it informs the robots so they all go back to the initial state of the process, i.e. selecting a new coordinator, retrieving a case and executing its solution.

The execution of a case may be aborted at any moment when any of the following situations occur:

- Any of the robots detects that the retrieved case is not applicable anymore. Once the execution of a case has started, we consider a case to be

applicable if the remaining ball's path is still free, i.e. there are no opponents within the trajectories the ball is about to follow.

- A robot receives an unexpected message. Because of the noise in the wireless network, a message can be delayed or even get lost. Hence, when a message arrives and it is not coherent with the current case reuse state, we opt for aborting the current case execution. Although we are aware that it is a drastic solution, due to the domain requirements (real time response and limited computational capacities), we believe that it is more suitable since it is a simple and fast tactic.

- A timeout occurs. We include timeouts for the states ADAPT and EXECUTE since we want to make sure that the case reuse is not going to take too long. In general this situation occurs when a robot does not receive all necessary messages or its perception fails. The former situation could cause the robot to remain in a state infinitely, while the latter could lead the robot to search for the ball in a wrong location, lengthening the behavior execution.

In any case, the robot detecting the exceptional situation sends an aborting message to the rest of the robots so that they all stop executing their actions. Then, they once again go back to the initial state in order to restart the process, i.e. select a coordinator, retrieve a case and reuse it. We must remark that most of the aborting situations occur due to the first situation (case applicability), while the remaining ones are mainly defined to ensure robustness in case of system failure and are not so common.

4.3 Conclusions and Future Work

This chapter has been addressed to the internal architecture of a robot, as well as the multi-robot architecture of the team. Regarding the robot's internal architecture we have proposed a hybrid architecture, where the deliberative layer is responsible for the high level decision-making, i.e. the combination of the CBR reasoning system and a region-based algorithm, while the executive layer controls the execution of the behaviors proposed by the deliberative layer.

Within the multi-robot architecture we define two types of robots: the retrievers, who include the CBR system and therefore are in charge of proposing cases, and the executors, who only execute the cases indicated by the retrievers, i.e. cannot propose solutions to the current state of the world. In order to determine the next case to reuse, a coordinator is selected among the retrievers based on their distances to the ball. After the coordinator is chosen, it retrieves a case and informs the rest of the team (retrievers and executors). At this point the reuse step of the case starts. Two positioning strategies have been proposed to start the execution of the actions indicated in the case. During the case reuse, any robot (retriever or executor) may abort the case reuse if it considers that the case is not applicable anymore.

As future work we propose to improve the case selection for reuse. Each retriever may propose a different case based on its internal beliefs of the state of the world. Hence, a negotiation protocol could be included to decide which is the most suitable retrieved case to reuse. This way we provide the team

with a more cooperative mechanism where all robots participate in the global decision-making and the achievement of the goals of the team.

Chapter 5

Learning the Scopes of Cases

The case base is the most fundamental component of a case-based reasoning system since it provides the domain knowledge of the reasoner, and therefore, determines the system's accuracy performance. As Ram states in [50], a reasoner program may fail on finding the right solution, if any, due to incomplete knowledge in the system. More precisely, he detects three sources for gaps in the system's knowledge: (*i*) novel situations, there is no case to solve the new problem, (*ii*) mis-indexed cases, although there might be a case in the case library to solve the new problem, the system is not able to retrieve it because of mistaken case indexation, and (*iii*) incorrect or incomplete case, the situation represented by the case may not be completely understood, and thus, the case is incorrect or incomplete. The work we present in this chapter is mainly addressed to knowledge adaptation, i.e. incorrect or incomplete cases (source *iii*), and initial steps for knowledge acquisition, i.e. novel situations (source *i*):

- *knowledge adaptation*: the perception of the expert providing the initial knowledge to the reasoner may largely vary from the perception of the reasoner. This a very common situation when the reasoning system is applied in the real world and depends on the system's sensors accuracy. Hence, although the knowledge provided to the reasoner might be "correct", from its point of view it is not. For this reason, an adaptation of the knowledge, in this case the case base, is necessary to achieve a correct performance of the reasoning system.

  A second reason for including a mechanism for adapting the reasoner's knowledge is that the environment may change (either gradually evolving or being suddenly altered) through time. Hence, the initial knowledge may become useless degrading the performance of the system.

- *knowledge acquisition*: although the expert may try to provide the necessary knowledge to the reasoner, she may miss some situations, and therefore, generate gaps in the system's knowledge. Besides, again related to the changes in the environment through time mentioned above, it is most likely that the expert cannot predict all future situations the reasoning system will have to deal with. Thus, to overcome these situations, it is fundamental for the system to automatically incorporate knowledge to cover these gaps.

In this chapter we present a first attempt to automate the adaptation and acquisition of the case-based reasoning system's knowledge with respect to the scope of a case, i.e. the case coverage. The motivations for studying an algorithm that allows the automation of this process in the working domain are:

1. because of the nature of the domain this work is focused on (real robots with high uncertainty in its perception), we believe that it is fundamental that the knowledge of the reasoner system corresponds to its actual perception, and not only to the expert's one;

2. the opponents of a game are part of the environment the reasoner system is dealing with. If we consider the opponent's strategy as part of the environment, it is easy to see that the environment may radically vary based on the strategy the opponents apply during a game, i.e. a change in the opponent's strategy (even more, changing opponents) implies a modification on the environment. Hence, it is essential to provide the reasoning system with an engine to alter its current knowledge through adaptation. This adaptation may lead to gaps in the knowledge, and therefore, introducing new cases to cover these gaps is crucial.

Thus, we propose a supervised learning algorithm for the adaptation of the knowledge of the reasoning system. More precisely, we focus the algorithm on learning the ball's scope of a case (which matches with the case scope), although the concepts and mechanisms presented here can be equally applied to the opponents' scopes.

The idea is to provide the robot with a set of cases which must be adapted to its actual perception. The expert knows several generic situations (prototypical cases) and their corresponding solutions, but cannot predict the real scope of the cases from the robot's point of view. The learning starts with a classification task, where the robot classifies the problems proposed by the expert with respect to the available cases in the case library (every case corresponds to a different class). If a class is returned, i.e. a case, then the robot adjusts the case scope based on the feedback classification. Ideally, this feedback could be automatically inferred by the robot itself after observing the outcome of its actions, but this is a far more complex task that we do not address in this work. On the contrary, if no class is returned, then it means that there is a gap in the case library. Thus, an automatic mechanism for creating new cases should be included in the system. Once a case is created, the system must determine the case coverage, i.e. the case scope. At this point, the adaptation process presented in this chapter can be applied.

The chapter is organized as follows. We first introduce the learning algorithm for adapting the case scope, i.e. when and how to modify the scope. Next, we present a simple mechanism to introduce new cases when no solution is found. Experiments to show the effectiveness of the leanring algorithm follow next, and finally, conclusions and future work conclude the chapter.

5.1 Scope Adaptation Algorithm

The adaptation mechanism is addressed to existing cases, and does not deal with gaps in the case library. As we have described in Section 3.1.3, the ball's

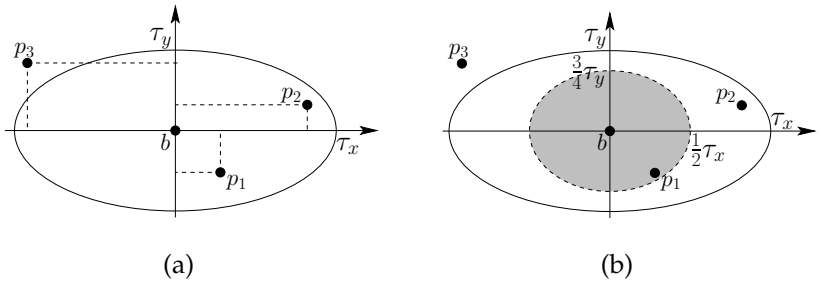


Figure 5.1: Case scope representation. The center of the ellipse, b, corresponds to the ball's position indicated in the case, while problems p_1, p_2 and p_3 are the problems to classify. (a) Graphical similarity evaluation of three problem examples. (b) Example of the security region (gray region) and the risk region (white region) of a case defined by $\gamma_x = 0.5$ and $\gamma_y = 0.75$.

scope is defined by an ellipse with radius τ_x^B and τ_y^B. Since we are focusing this section on learning the ball's scope, for simplicity we will generalize the concept of ball's scope to the scope of the case and omit the superindex B in the τ parameters. Moreover, henceforward we will refer to the scope of the case, either as the scope or the ellipse. The algorithm consists in updating the size of the cases scopes varying their τ parameters, i.e. the radii of the ellipse. To this end, we must define a policy to determine when and how these parameters should be adjusted given the expert's feedback.

The center of the ellipse represents the position of the ball (b) on the field specified in that case. As we move towards the boundary of the ellipse, the uncertainty about whether the points belong to the scope of the case increases. The points next to the boundary of the ellipse represent higher degree of uncertainty. Figure 5.1a depicts the scope of a case c_i and three problems to classify. Problems p_1 and p_2 are similar to the case, i.e. they belong to the class c_i, since they are within its scope, although p_1 with higher similarity degree compared to p_2 (since it is closer to the center of the scope). In contrast, p_3 is out of the scope, and therefore we do not consider it similar to the case, i.e. it does not belong to class c_i. If two cases overlap their scopes, a problem may belong to one or more classes, i.e. the problem to classify has some similarity degree with both cases. Thus, the outcome of the classification corresponds to the class with higher similarity. The similarity is computed with the gaussian function presented in 3.3.1.

5.1.1 When to Adjust the Values

The goal of increasing or decreasing the size of the ellipse is to reach the expected region that a case should cover. The expected region corresponds to the region that the expert believes the case should cover. We call *security region* the set of points next to the center of the ellipse, and *risk region* those near the boundary of the ellipse. We define γ_x and γ_y as the proportion of radius τ_x and τ_y that corresponds to the security region (where $\gamma_x \in [0, 1]$ and $\gamma_y \in [0, 1]$).

Figure 5.1b shows an example of these two regions. Problem p_1 is within the security region, while problem p_2 is within the risk region of the case scope.

When the system classifies a new problem, i.e. returns a case c to which the problem belongs with higher similarity degree, we use the expert's feedback for tuning the scope's parameters τ_x and τ_y of that case. If the proposed case is correct, the scope of the case is increased. Otherwise, it is decreased.

- *Increasing the scope*: If the problem is located within the security region (i.e. the position of the ball is in this region) the system cannot introduce new knowledge to case c. Its current information is robust enough to determine that the problem corresponds to the scope of that case. On the contrary, if the problem is inside the risk region the system can confirm that the current scope of the case is correct. Thus, we increase the size of the ellipse modifying the scope's parameters to (*i*) enlarge the security region and (*ii*) to evaluate a bigger scope of the case. Since the security region is computed as a proportion of the ellipse size, expanding the ellipse results in expanding this region as well.

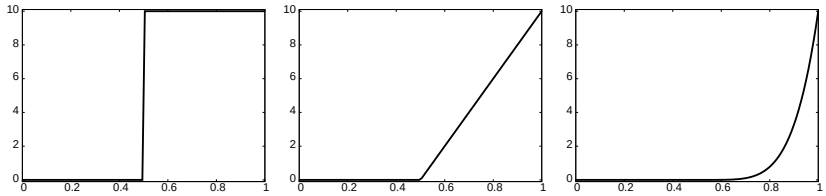
- *Decreasing the scope*: A problem is incorrectly classified using case c because of its scope overestimation. Hence, we have to reduce the size of the ellipse. If the ball is inside the security region, we do not decrease the parameters since it corresponds to a robust region. If the problem is within this region and the feedback is negative, we assume that the error is originated by other reasons (wrong localization) and not because of wrong information of the case. Suppose the following situation: the robot is not well localized and as a consequence, it perceives the ball in a wrong position. It could happen that it correctly classifies the problem given its own perception. But from the external observer perception, the returned case that classifies that problem is not the right one. Therefore, the feedback given to the robot is negative. If the system reduces the ellipse, it could radically reduce the scope of the case, not because it was overestimated, but because of the high uncertainty in the robot's perception. However, when the problem is inside the risk region, the system does reduce the scope of the case, since the scope overestimation might be the cause of the negative feedback.

In summary, the adaptation algorithm enlarges or reduces the scope of a case when the problem to solve is correctly or incorrectly solved and it is within the case's risk region.

5.1.2 How to Adjust the Values

After describing when to enlarge or reduce the scope of a case, we will now detail how to update the τ parameters, or in other words, how much to increase or decrease them. First we introduce some notation. Given a new problem p to classify, and given the case (class) c that classifies that problem at time t, we define:

- $\hat{\delta}_x, \hat{\delta}_y$: the maximum increasing values for τ_x and τ_y respectively. These values are assigned by the expert.



(a) (b) (c)

Figure 5.2: Increasing policy functions, f_{inc}, with $\hat{\delta} = 10$ and $\gamma = 0.5$: (a) step function, (b) linear function, and (c) polynomial function.

- δ_x^t, δ_y^t: the actual increasing values for τ_x^t and τ_y^t respectively. These are the values to compute.
- $\triangle_x^t, \triangle_y^t$: the relative distances between problem p and the center of the case scope c, i.e. the ball's position described in the case:

$$\triangle_x^t = p_x - b_x \qquad \triangle_y^t = p_y - b_y$$

  where p_x and p_y correspond to the ball's coordinates in the problem and b_x and b_y, the ball's coordinates in the case.

In order to compute the δ_x^t, δ_y^t values for adjusting the τ^t values at time t we propose three increasing policy functions, f_{inc} (henceforward we will refer to them as the *policies*):

- *fixed*: the increasing amount is a fixed value. Thus, we define a step function (Figure 5.2a):

$$\delta_x^t = \begin{cases} \hat{\delta}_x & \text{if } \gamma_x \tau_x^t \leq \triangle_x^t \leq \tau_x^t \\ 0 & \text{otherwise} \end{cases} \qquad \delta_y^t = \begin{cases} \hat{\delta}_y & \text{if } \gamma_y \tau_y^t \leq \triangle_y^t \leq \tau_y^t \\ 0 & \text{otherwise} \end{cases}$$

- *linear*: we compute the increasing value based on a linear function (Figure 5.2b):

$$\delta_x^t = \begin{cases} \frac{\triangle_x^t - \gamma_x \tau_x^t}{\tau_x^t - \gamma_x \tau_x^t} \cdot \hat{\delta}_x & \text{if } \gamma_x \tau_x^t \leq \triangle_x^t \leq \tau_x^t \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_y = \begin{cases} \frac{\triangle_y^t - \gamma_y \tau_y^t}{\tau_y^t - \gamma_y \tau_y^t} \cdot \hat{\delta}_y & \text{if } \gamma_y \tau_y^t \leq \triangle_y^t \leq \tau_y^t \\ 0 & \text{otherwise} \end{cases}$$

- *polynomial*: we compute the increasing value based on a polynomial function (Figure 5.2c):

$$\delta_x^t = \begin{cases} \frac{(\triangle_x^t - \gamma_x \tau_x^t)^5}{(\tau_x^t - \gamma_x \tau_x^t)^5} \cdot \hat{\delta}_x & \text{if } \gamma_x \tau_x^t \leq \triangle_x^t \leq \tau_x^t \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_y^t = \begin{cases} \frac{(\triangle_y^t - \gamma_y \tau_y^t)^5}{(\tau_y^t - \gamma_y \tau_y^t)^5} \cdot \hat{\delta}_x & \text{if } \gamma_y \tau_y^t \leq \triangle_y^t \leq \tau_y^t \\ 0 & \text{otherwise} \end{cases}$$

Once we compute the increasing values, we update the case scope:

$$\tau_x^{t+1} = \tau_x^t + \delta_x^t \qquad \tau_y^{t+1} = \tau_y^t + \delta_y^t$$

where τ_x^{t+1} and τ_y^{t+1} represent the updated radius of the case scope.

The goal of reducing the scope of a case is to resize the ellipse so that the incorrectly solved problem is not considered similar to the case anymore, i.e. the problem remains outside the ellipse. We compute the new values as follows:

$$\tau_x^{t+1} = \begin{cases} \triangle_x^t & \text{if } \gamma_x \tau_x^t \leq \triangle_x^t \leq \tau_x^t \\ \tau_x^t & \text{otherwise} \end{cases} \qquad \tau_y^t + 1 = \begin{cases} \triangle_y^t & \text{if } \gamma_y \tau_y^t \leq \triangle_y^t \leq \tau_y^t \\ \tau_y^t & \text{otherwise} \end{cases}$$

Note that we only update the τ_x value if the problem x component ($\triangle_x$) is within the risk region. Similarly, we modify the τ_y if the y component ($\triangle_y$) is within the risk region. Updating both values separately prevents from radically reducing the scope of the case.

Algorithm 3 describes the overall adaptation algorithm. Given a problem p, the case c that classifies p, the expert's feedback and the parameters presented above (the increasing policy function f_{inc}, $\hat{\delta}$, and γ), the algorithm updates the τ values for resizing the case scope if the problem is within the scope's risk region (line 1). When the feedback is positive (line 4), the algorithm increases the current size of the scope (lines 5 to 8). Otherwise, the scope is reduced (lines 10 to 14). The process is repeated until the expert determines that the expected scope has been reached, i.e.the new incoming problems are classified with little error. Next we describe a simple example for illustrating the algorithm.

Algorithm 3 Update_Scope(p, c, feedback, f_{inc}, γ_x, γ_y, $\hat{\delta}_x$, $\hat{\delta}_y$)

```
1:  if p ∈ risk_region(c) then
2:     △_x ← p_x − b_x
3:     △_y ← p_y − b_y
4:     if feedback is TRUE then
5:        δ_x ← f_inc(△_x, γ_x, δ̂_x, τ_x)
6:        δ_y ← f_inc(△_y, γ_y, δ̂_y, τ_y)
7:        τ_x ← τ_x + δ_x
8:        τ_y ← τ_y + δ_y
9:     else
10:       if △_x ≥ γ_x τ_x then
11:          τ_x ← △_x
12:       end if
13:       if △_y ≥ γ_y τ_y then
14:          τ_y ← △_y
15:       end if
16:    end if
17: end if
```

5.1.3 Example

Figure 5.3 depicts four steps of the adaptation process. The gray region represents the security region, while the dashed ellipse corresponds to the expected

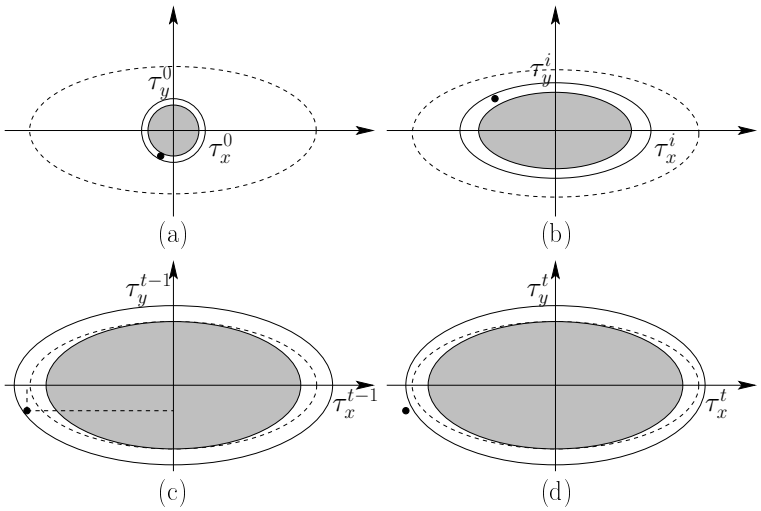


Figure 5.3: Case scope evolution. The dashed ellipse represents the "ideal" scope. In gray, the security region with $\gamma_x = \gamma_y = 0.8$.

scope of the case (defined by the human trainer) we attempt to reach. Any problem located within this ideal area produces a positive feedback by the expert. The black dot represents a new solved problem (ball position with respect to the case). Figure 5.3a shows the initial stage at time 0, where the scope of the case is minimum (τ_x^0, τ_y^0). Since the new solved problem is within the risk region and the feedback is positive, we proceed to enlarge the size of the ellipse using one of the policies defined.

At time i, Figure 5.3b, we can observe that the ellipse has increased, but still has not reached the expected size. Hence, we keep on enlarging the scope by solving new problems as long as the expert feedback is still positive.

Figure 5.3c, time $t-1$, depicts a situation where the updated ellipse is bigger than the expected size. From now on, the feedback may be positive or negative. If a new problem is within the risk region and the feedback is positive, then we would proceed to increase the ellipse. But, if the feedback is negative, then the decreasing process is used to reduce the ellipse. The figure shows an example of this situation. As we can see, the new problem is located within the risk region, but out of the ideal scope. Thus, the current scope is reduced, but only updating τ_x since $\triangle_y < \gamma_y \tau_y$.

Figure 5.3d shows the updated scope, where the problem remains outside the scope of the case. As more problems are solved, the scope of the case will converge to the ideal scope.

In conclusion, we distinguish two phases in the adaptation process: growing the scope of the case, and converging to the ideal scope. During the first phase, the feedback is always positive and the scope is always being expanded. The second phase occurs once the expected scope is exceeded. Then, the feedback could either be positive or negative. The goal of the first one is to enlarge the scope, while the second one, is to converge to the ideal scope the human trainer expects.

5.2 Acquiring New Cases

After the adaptation step, the knowledge of the system might present some gaps, i.e. the scope of the cases may not cover the whole field. The coverage depends on the number of cases used during the first learning stage. However, as we previously mentioned, the expert cannot a priori define all possible cases. Hence, we present the first step towards a learning mechanism to acquire new knowledge when novel situations occur and the system does not have any case to solve them. Although with this mechanism we create simplified cases, it is useful to guide the expert in completing the knowledge of the system, providing support on the manual generation of the case base.

A new case is created using the description of the environment (problem description, i.e. robot's and ball's position), and a generated gameplay (solution of the new case). To create a gameplay, we provide the system a set of possible actions the robot can perform. Hence, given a new problem to solve, if the system does not retrieve any case, i.e. the problem does not belong to any of the classes (either due to imprecision problems or because the problem is actually in a gap) the system randomly selects a gameplay. The robot executes the suggested action and the expert evaluates the correctness of the solution proposed. Only if it succeeds, the new case is created. We are aware that this procedure is too simplistic. But as mentioned before, at least the process provides information to the expert regarding the coverage of the current system's knowledge, the case library, allowing her to afterwards improve the case description with more suitable information.

When a new case is inserted into the system, it is created with a minimum scope (a small ellipse). From that moment on, the evolution of the new case depends on how often the robot reuses it, enlarging or reducing its scope using the adaptation mechanism presented previously. The idea is that at the beginning, the new case could seem to be a good solution for that concrete situation, but its actual effectiveness has to be evaluated when the robot reuses it. As time passes, if the scope of the case does not increase, and instead, it is reduced, we can deduce that the case is not useful for the robot's performance. On the contrary, if its scope increases, or at least, it remains stable, then we consider that the case contributes to the system's knowledge.

5.3 Experiments

This section describes the experiments performed in order to test the learning algorithm introduced above. We divide the experimentation in two stages: simulation and real robots.

5.3.1 Simulation Experiments

The goal of this first phase is to determine the behavior of the policies using different values for the parameters presented in Section 5.1.2. Since we had to test different combinations of values, simulation was the fastest way to obtain orientative results. The most relevant were selected for the experimentation with real robots.

We based the experiments on the adaptation of a single case to observe how the different values (τ_x,τ_y) computed through the learning process affect the evolution of its scope, i.e. the resulting size of the ellipse in mm. The initial case was defined with a small scope, $\tau_x = 100$ and $\tau_y = 100$. The expected outcome ("ground-truth") was $\tau_x = 450$ and $\tau_y = 250$. A trial consists of 5000 random problems which are iterated as the input for the learning algorithm. For each trial we fix the increasing policy and the parameters γ and $\hat{\delta}$. The outcome of the trial are the new learned τ values, i.e. the radius of the scope. An experiment consists of 10 trials with the same parameters per trial. Since the problems are randomly generated, each trial generates a different outcome. For every experiment we combined each policy with the following set of values per parameter:

- security region proportion size:

$$\gamma_x = \gamma_y = \{0.5, 0.6, 0.7, 0.8, 0.9\}$$

- maximum increasing parameter (expressed in mm):

$$\hat{\delta}_x = \hat{\delta}_y = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$$

Figure 5.4 shows the average of the results obtained for the experiments performed. The x axis corresponds to the maximum increasing parameter, $\hat{\delta}$, while the y axis indicates the average τ computed for each experiment. Each curve is obtained varying the values of the security region parameter, γ. On the one hand, $\hat{\delta}$ defines how much the ellipse may increase at each time. Hence, the higher their values, the bigger the resulting scope of the case. On the other hand, γ determines the size of the security region and the risk region (low values represent small security regions and large risk regions). The risk region determines when the scope of the case has to be modified. As this region increases, there are more chances of modifying the scope as well. Thus, for all three policies, the curves tend to increase from left to right, i.e. obtaining larger scopes (higher τ values) on the right side of the figure.

With respect to the evaluation of the policies behavior, the fixed policy obtains the highest τ values, while the polynomial, obtains the lowest ones. The former function has a more aggressive behavior, radically increasing the size of the ellipse always with the maximum increment allowed. The latter function has a more conservative behavior, computing small increments (δ's) for problems near the boundary between the security and the risk region, and enlarging them as the problems to solve reach the boundary of the ellipse. We can easily observe the behavior differences between all three policies and their influence on the outcome, τ_x and τ_y, in Figure 5.5. We depict the evolution of the learned values for a single trial with $\hat{\delta} = 10$ and $\gamma = 0.5$. During the converging step of the adaptation process (region marked with a dashed rectangle) the variations of the radius using the fixed policy are much larger (the oscillation of the τ_x ranges from 450 to 760, and the τ_y, from 250 to 469) than the ones obtained with the polynomial policy (the τ_x varies from 450 to 552, and the τ_y, from 250 to 326).

As a consequence of the influence of the two factors mentioned above (i.e. the behavior of the policy itself and the values of the $\hat{\delta}$ and γ parameters used

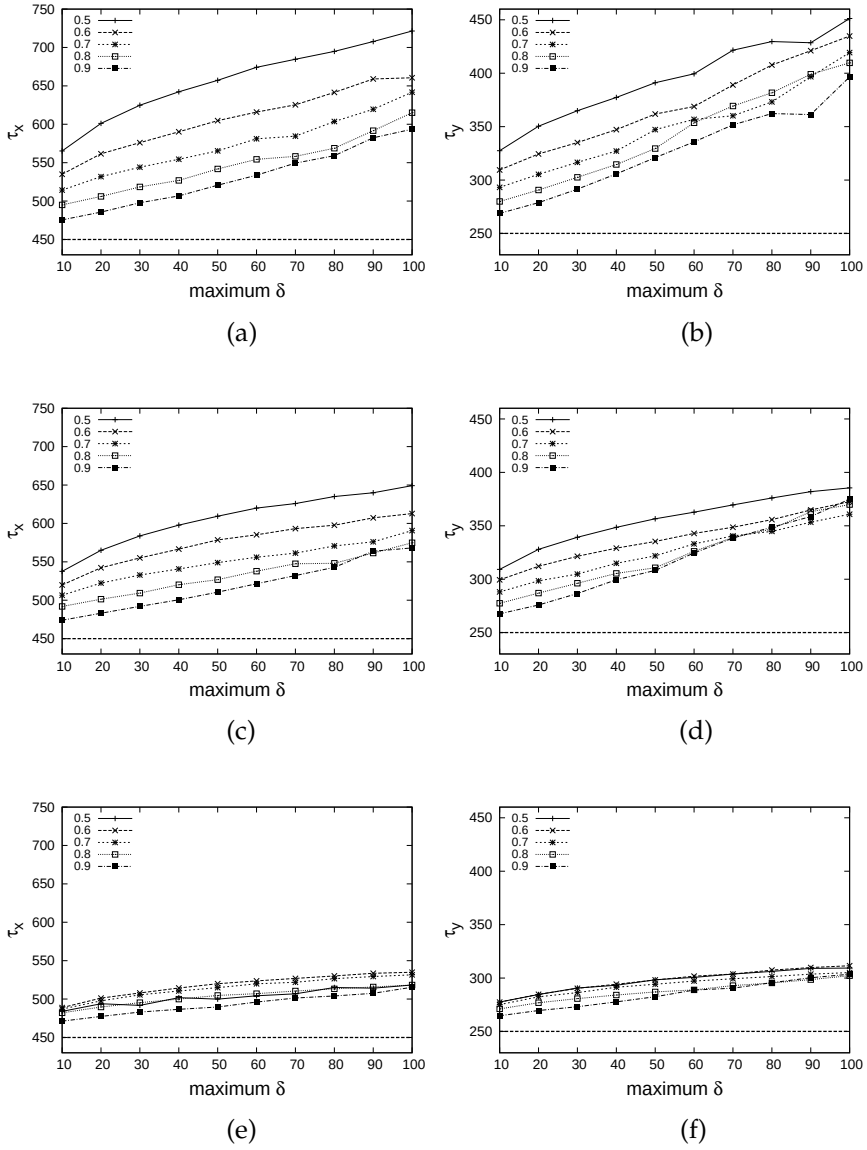


Figure 5.4: Resulting τ_x (left column) and τ_y (right column) using the following policies: fixed (a) and (b); linear (c) and (d); and polynomial (e) and (f). Ground truth represented with a dashed line.

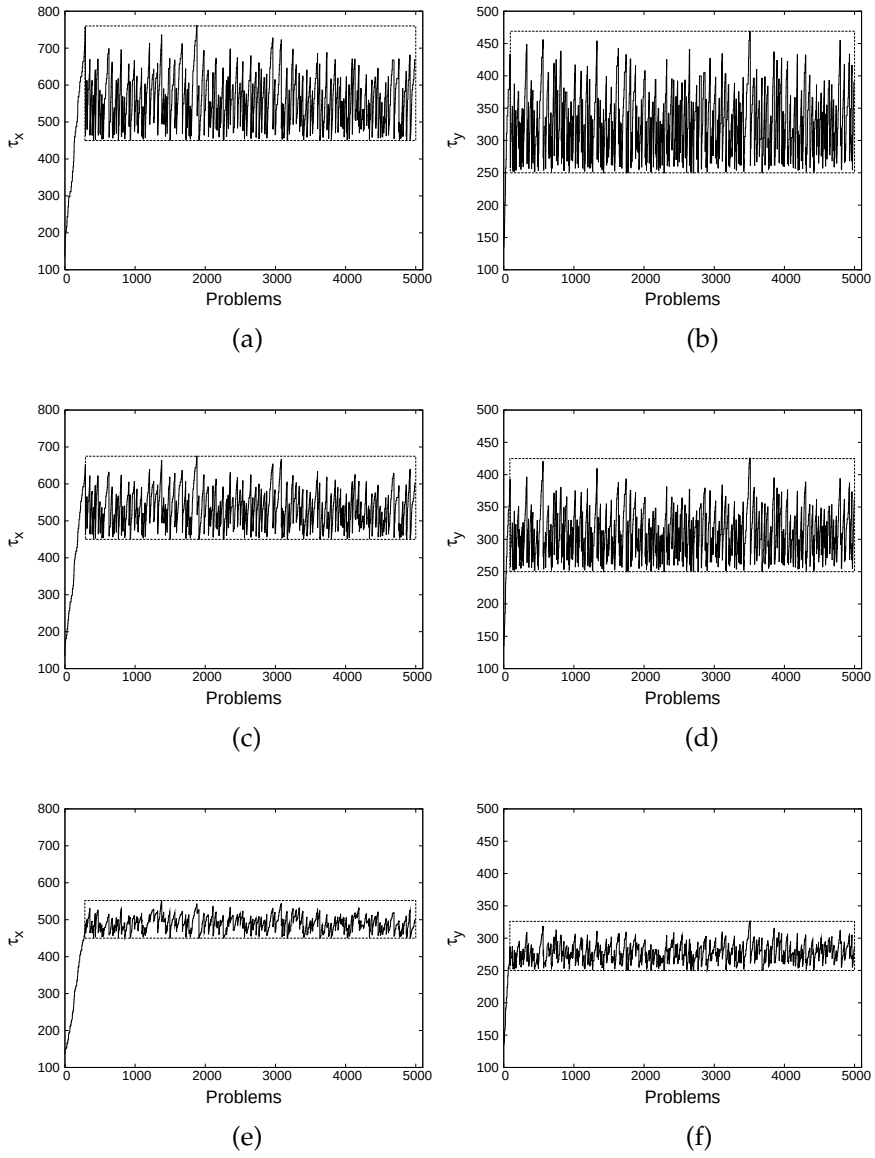


Figure 5.5: Single trial example of the evolution of the τ_x (left column) and τ_y (right column) scope parameters using the following policies: fixed (a) and (b); linear (c) and (d); and polynomial (e) and (f). The dashed rectangles contain the convergence steps for each curve.

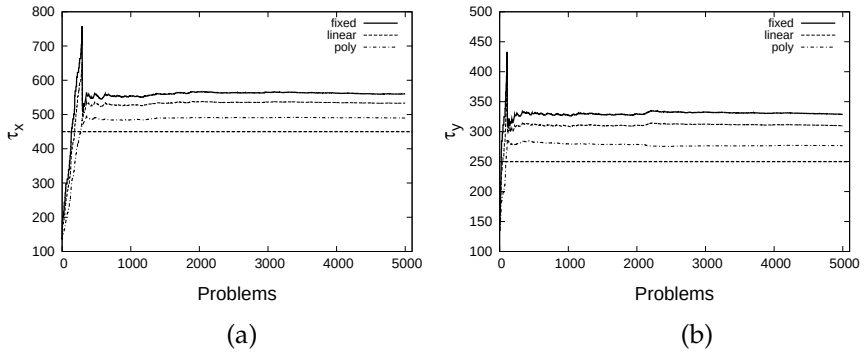


Figure 5.6: Convergence of the average value of (a) τ_x and (b) τ_y of a specific trial. Ground-truth in dashed line.

in the learning process), the distance between the curves in Figure 5.4 using the fixed policy is significantly larger than the ones using the polynomial policy. For instance, in Figure 5.4a the average τ_x computed with the fixed policy varies on the left side (low $\hat{\delta}$) from 475 to 565, and on the right side (high $\hat{\delta}$) from 593 to 721. On the contrary, the curves computed with the polynomial functions (Figure 5.4e) vary on the left side from 471 to 488, and on the right side, from 515 to 534. Regarding the linear policy, we can observe that it has an intermediate behavior between the other two (tending to behave more as the fixed policy). Furthermore, the differences between the curves obtained with the fixed policy and the ground-truth ($\tau_x = 450$) are much larger than the ones obtained with the polynomial policy. As expected, we can state that the polynomial policy has a more stable behavior compared to the fixed one, besides of achieving similar results to the ground-truth. Figure 5.6 shows the average value of τ_x and τ_y computed after each problem has been classified for a specific trial (with $\hat{\delta} = 10$ and $\gamma = 0.5$). It is clear that the polynomial policy obtains the closest values to the ground-truth.

The configuration that obtained the closest values to the "ideal" ones, $\tau_x = 450$ and $\tau_y = 250$, was: polynomial policy, $\gamma = 0.9$ and $\hat{\delta} = 10$. Hence, after the experimentation we can confirm that a more conservative strategy, i.e. low increasing values and small risk regions, is the most appropriate combination to obtain the desired scope of the cases. This conclusion is clear when performing the experiments in a simulated environment. But two problems arise when extending the experiments to the real world: time and uncertainty. First, the number of iterations needed to reach the expected result is not feasible when working with real robots; and second, a noise-free environment is only available under simulation. Although we have observed different behaviors in the graphics obtained when gradually modifying the parameters, these differences are not so obvious in a real environment because other issues modify the expected result. Therefore, the next stage is to experiment with the robot in the real world evaluating the most relevant parameters (understanding relevant as the ones that show more contrasting behaviors) to determine the effectiveness of the presented learning algorithm.

5.3.2 Real World Experiments

Two types of experiments were performed in a real environment. The first one is aimed at finding out the most appropriate parameters and policy to use during the learning process. The second one consists in evaluating the convergence of the cases in a given case base, and the acquisition of a new case.

Testing the Parameters and Policies

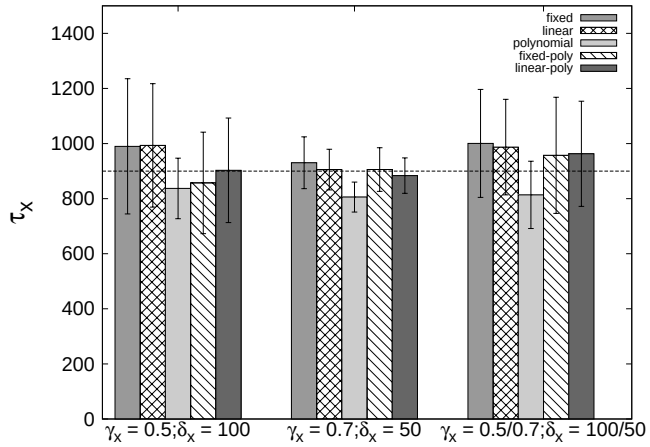
As mentioned in the example of the learning algorithm (Section 5.1.3), we can divide the training process in two steps: growing the scope of the case and converging to the expected scope. We are interested in rapidly enlarging the size of the ellipse until almost reaching the expected one (i.e. use as less problem examples as possible since in the real world we cannot afford to generate and classify a large set of problems), and then opt for a more conservative behavior to adjust it. The growing step finalizes when a negative feedback is given by the expert's for the first time. As we mentioned in the explanation of the algorithm, a negative feedback means that the scope of the case is overestimated, and therefore it has to be reduced to converge to the expected one. Thus, we modify the algorithm such that it switches from one policy to the other when the convergence step starts, i.e. the size of the ellipse is decreased for the first time. Moreover, besides alternating the policies used for determining the growing size of the ellipse, we can also vary the parameters that define the size of the security region (γ), and the maximum increment for enlarging the scope ($\hat{\delta}$) as observed in the simulation evaluation.

We have defined a set of strategies to study their resulting behavior and thus, select the most appropriate one to use in the second part of the experimentation. Table 5.1 summarizes the set of strategies. We differentiate between the two steps of the learning algorithm, growing and converging the scope, indicating the policy used at each step, as well as the values of the parameters γ and $\hat{\delta}$ (the same values for both x and y axis). The experimentation is similar to the simulation stage, where the experiments are based on a single case with initial scope $\tau_x = \tau_y = 100$. The expected scope of the case is $\tau_x = 900$ and $\tau_y = 600$. A single trial consists in positioning the robot in a fixed location on the field and randomly generating 100 problems that the robot detects as inputs for the learning algorithm. We manually move the ball from one position to another within the field of view of the robot since for this experiments the robot only moves its head to search for the ball. The learning algorithm is tested with one of the strategy configurations shown in Table 5.1. An experiment consists of 10 trials for each strategy.

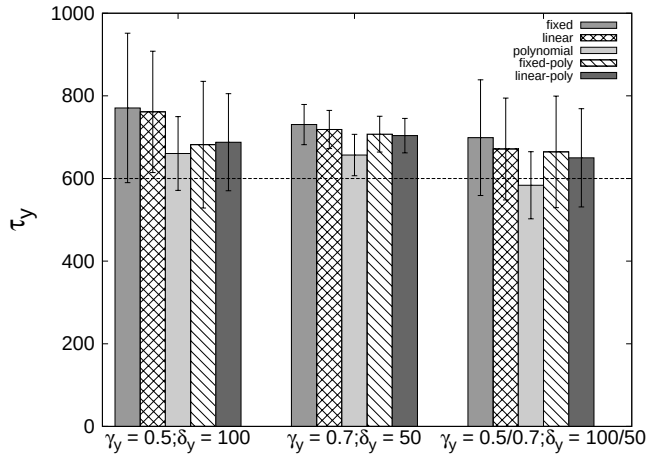
Figure 5.7 illustrates the average size of the learned scope for each experiment. Comparing the results obtained with respect to the expected scope (ground truth in both figures) we conclude that:

- the strategies using only the fixed and linear policies for both learning steps generate the highest τ values, while the strategy using only the polynomial policy obtains the lowest. As concluded in the simulation experiments, the former policies have a more aggressive behavior compared to the latter.

- the strategy defined with only the polynomial policy does not even reach



(a)



(b)

Figure 5.7: Average outcomes and deviations for the different strategies. The dashed line represents the ground truth. (a) τ_x average and ground truth $\tau_x = 900$. (b) τ_y average and ground truth $\tau_y = 600$..

growing step			convergence step		
policy	γ	$\hat{\delta}$	*policy*	γ	$\hat{\delta}$
fixed	0.5	100	fixed	0.5	100
fixed	0.7	50	fixed	0.7	50
fixed	0.5	100	fixed	0.7	50
linear	0.5	100	linear	0.5	100
linear	0.7	50	linear	0.7	50
linear	0.5	100	linear	0.7	50
polynomial	0.5	100	polynomial	0.5	100
polynomial	0.7	50	polynomial	0.7	50
polynomial	0.5	100	polynomial	0.7	50
fixed	0.5	100	polynomial	0.5	100
fixed	0.7	50	polynomial	0.7	50
fixed	0.5	100	polynomial	0.7	50
linear	0.5	100	polynomial	0.5	100
linear	0.7	50	polynomial	0.7	50
linear	0.5	100	polynomial	0.7	50

Table 5.1: Strategies defined in the real world experiments varying the policies and the scope parameters γ and $\hat{\delta}$.

the expected scope most of the times due to its low increasing speed during the growing stage. At every time step the increment is not as high as the one computed with the other policies and therefore, reaching the expected scope requires a larger amount of problems to solve until starting the convergence stage.

- the strategies combining two types of policies (fixed/polynomial and linear/polynomial) as well as the values for the scope parameters ($\gamma = 0.5/0.7$ and $\hat{\delta} = 100/50$) obtain the closest scopes to the expected ones, since they combine the advantages of both policies and the parameters increasing properties, i.e. first large risk regions and high increments for the growing step, and then small risk regions and low increments for the converging step.

- comparing the results for the x and the y axis with respect to their ground truths (horizontal lines in the figures) we can observe that the ones obtained for the y axis reach and exceed the ground truth most of the times. During the experiments, the robot is positioned parallel to the x axis. Therefore, variations in the x component of the ball's position are harder to identify by the robot, compared to the y component. For instance, moving the ball 10cm closer to the robot is not as easy to distinguish as moving the ball 10cm to the left. Hence, the number of problems needed to rapidly fulfill the growing process for the x axis is higher than for the y axis. Since the problems were randomly generated without taking into account this issue, we can estimate that 50% of the problems were used for each axis, and therefore, not enough problems remained for the converging step in the x axis.

We conclude that the best strategy within the real world is to have aggres-

sive strategies for the growing step in order to reach as fast as possible the ideal scope, and then progressively adjust it to the robot's perception using a more stable strategy.

Adapting the Case Base

The final experiment consists in training a small case base in order to evaluate if the robot is able to learn the expected scopes. We created a simple case base of four cases (Figure 5.8a) that covered a quarter of a field:

- *center*: midfield.
- *side*: left edge.
- *corner*: left corner.
- *front*: between the *center* case and the goal.

All cases were initiated with the same scope ($\tau_x = \tau_y = 100$) as depicted in Figure 5.8b. A trial consists of 50 random problems manually positioning the ball in a quarter of the field and let the robot move searching for the ball until facing it. During the growing step we used the fixed policy with large risk region and high increment parameters ($\gamma = 0.5$ and $\hat{\delta} = 100$) to rapidly reach the expected scopes. For the converging step the algorithm switched to the polynomial policy with small risk region and low increment values ($\gamma = 0.7$ and $\hat{\delta} = 50$). We performed 25 trials in total.

The outcome of a trial example of the adaptation algorithm is drawn in Figure 5.8c. The classified problems are represented with crosses ($\times$) for the *center* case, circles ($\circ$) for the *side* case, plus ($+$) for the *corner* case and squares ($\square$) for *front* case. As we can observe, the modified scopes approximate with high accuracy the expected outcome. Figure 5.8d shows the final steps of a trial where the size of the ellipses are converging towards the final outcome. Finally, Figure 5.8e shows the average of the 25 trials outcomes. As we can see, the robot successfully acquired the estimated scopes for the cases in the case base designed by the expert (Figure 5.8a). Moreover, in spite of the high uncertainty in the robot's perception, we can deduce that it is still close enough to the expert's own perception.

Acquiring Knowledge

After the adaptation process of the case base, the robot is ready to acquire new cases. The goal is to verify that the robot is able to fill in the gaps of the adapted knowledge. We focused the experiment on learning a single case located between the four cases. The expected action was to get near the ball facing the goal and bump it (the intention is to bring the ball closer to the goal not with a forward kick since it is too strong and would push it outside the field).

We performed 20 trials, each composed of 50 random problems. Through all the trials the new case was at some point created and adapted to cover the empty region, while the other cases' scopes stayed stable even though the adaptation algorithm was being used. This confirms that the conservative policy used during the converging step (polynomial policy, small risk region and low increment values) ensures stability of the learned parameters. Figure 5.8f

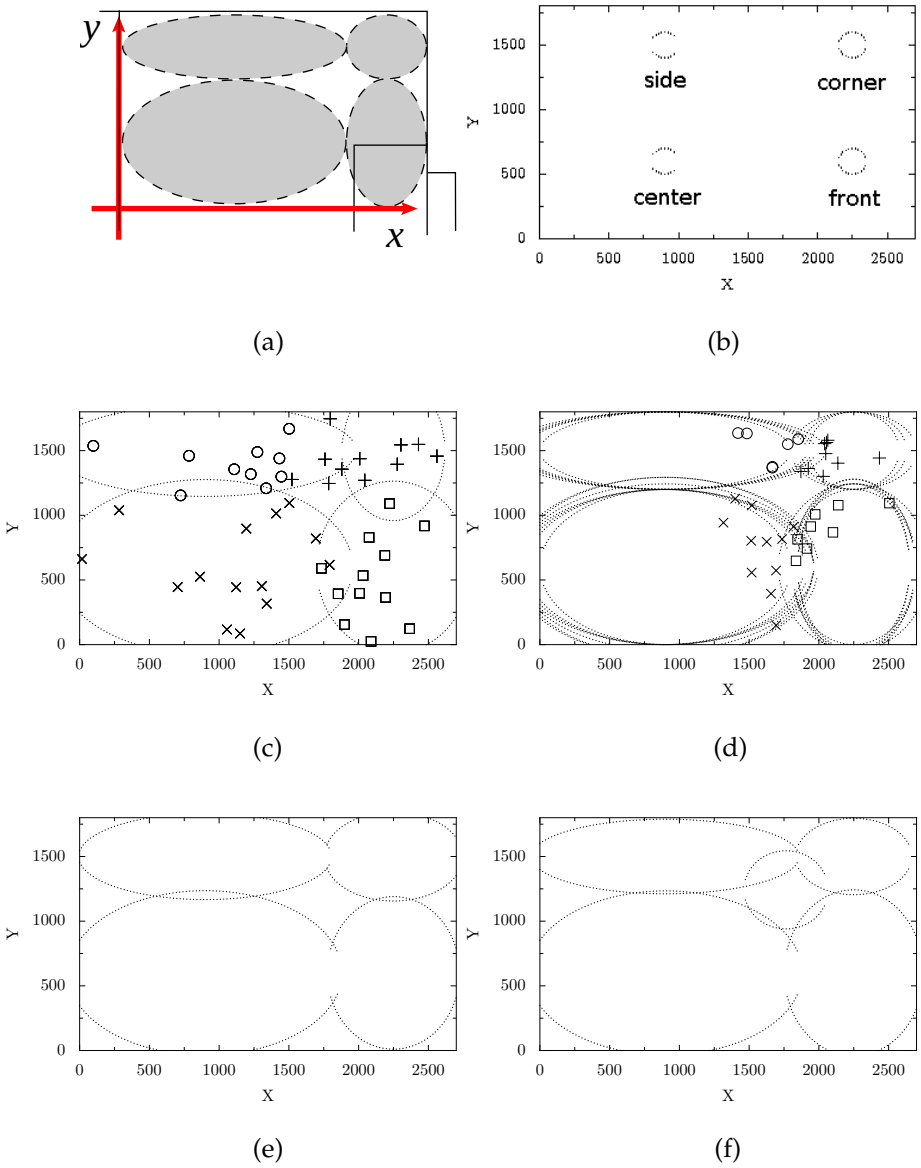


Figure 5.8: (a) Case base designed by the expert. (b) Initial case base for the learning process. (c) Trail example outcome. (d) Converging step of the learning process. (e) Final case base obtained after the adaptation process (average of the 25 trials). (f) Acquiring a new case (average of 20 trials).

shows the scope (average of the 20 trials) of the new case after expanding it. We can observe a slight modification of the remaining scopes' size with respect to Figure 5.8e (the initial case base for this last experiment). As we can see, the gap is almost completely covered with the expected case.

5.4 Conclusions and Future Work

We have presented the first steps towards developing an automated mechanism to adapt and acquire the knowledge of a reasoner engine. More precisely, the mechanism focusses on learning the scope of a case. The algorithm consists in a two-step process: growing the scope, and converging the scope. We have presented different policies to determine how to increase the size of the scope, and when to do it. We have also introduced a simple mechanism to create new cases. To analyze the learning algorithm, we have performed experiments both in simulation and with the real robot.

The experiments we have presented confirm that the proposed algorithm is able to successfully evolve the cases scopes based on the robot's perception. Moreover, it also confirms that the resulting scopes are similar to the ones expected by the expert's perception. Hence, although the adaptation is not necessary for the initial cases created by the expert, it is indeed useful to guide her through the acquisition of new cases and the coverage of the existing ones.

The introduced knowledge acquisition algorithm is not really meant for the reasoning system to successfully acquire new knowledge, but to help the expert on detecting possible gaps in the system's knowledge. As future work we would like to develop a more complete process for knowledge acquisition, so that not only serves as a support engine for the expert, but in fact, automatically creates new cases with complete descriptions, i.e. including teammates, opponents, and more complex solutions. Once a case has been introduced, the adaptation algorithm presented in this chapter can be used to generate its coverage.

We believe that the learning approach should observe the robot's own performance and automatically detect new interesting situations to reproduce in the future. This is a very challenging task since the main difficulties are: (*i*) when to consider that a potential case has started, i.e. the initial situation, (*ii*) when does it finish, i.e. which are the actions to store as part of the solution of a case, and (*iii*) how to evaluate the outcome of the performed actions to determine if the case is useful or not for achieving the robot's goals (i.e. positive or negative feedback). While the challenges of the first two questions are easy to see, the latter one may seem less complex because it consists in judging whether the actions were successful or not. But in general, and specially in these kind of domains, this judgement is not so simple because the consequences of the actions taken at a given time cannot be analyzed in a short term, but in the long term. Thus the robot is faced with the *credit assignment problem*, i.e. which actions contributed to the success or failure of the execution? The problem can become even more difficult if there are more robots involved in the execution, i.e. who did well? This is a very ambitious and complex research area which would complete the cycle of the case-based reasoning process presented in this thesis.

Chapter 6

Experimentation

We focus this chapter on the evaluation of the approach presented in the previous chapters. The goal of the experimentation is to empirically demonstrate, not only that the robots successfully perform the task, but also that the performance results in a cooperative behavior where the team works together to achieve a common goal, a desired property in this kind of domain. The approach allows the robots to apply a more deliberative strategy, where they can reason about the state of the game in a more global way, as well as to have special consideration of the opponents. Thus, they try to avoid the opponents by passing the ball between them, which increases the possession of the ball by the same team, and therefore, the team has more chances to reach the attacking goal.

We compare our approach with the approach implemented in the Carnegie Mellon's CMDash'06 team, i.e. the region-based algorithm (RBA) described in Section 4.1.1. As we already mentioned, the approach includes an implicit coordination mechanism to avoid having two robots "fighting" for the ball at the same time. The robot in possession of the ball notifies it to the rest of the team, and then the rest of the robots move towards different directions to avoid collisions. The robots also have roles which force them to remain within certain regions of the field (for instance, defender, striker, etc.). The resulting behavior of this approach is more individualistic and reactive in the sense that the robots always try to go after the ball as fast as possible and move alone towards the attacking goal. Although they try to avoid opponents (turning before kicking, or dribbling), they do not perform explicit passes between teammates and, in general, they move with the ball individually. Passes only occur by chance and therefore are not previously planned. Henceforward we will refer to this approach as the *reactive* approach.

The experiments presented in this chapter are focused on verifying the following hypotheses:

> **Hypothesis 1.** *Action selection in multi-robot domains is feasible applying Case-Based Reasoning techniques, since it facilitates the design of the robots behaviors, it is close to what humans do, and it provides a clear model for defining the situations to solve and their corresponding solutions.*

Hypothesis 2. *The approach proposed is robust enough to deal with uncertainty in the incoming information (perception) and to recover from imprecision in the outcome solution (robot's action execution).*

Hypothesis 3. *Due to the adversarial component in the working domain, a team of robots using a cooperative strategy that includes passes between robots outperforms an individualistic strategy where the robots do not plan joint actions to achieve the common goal.*

The first part of the chapter corresponds to the Case-Based Reasoning system settings, mainly the case base description. Then we detail the experiments setup including the evaluation measures used to assess the experimentation results, as well as the different scenarios used. Finally, we present and analyze the outcome of the experiments performed, both in simulation and with real robots.

6.1 CBR System Settings

The case base used for the experimentation is composed of 136 cases. From this set, 34 cases are hand-coded, while the remaining ones are automatically generated using spatial transformations exploiting the symmetries of the soccer field as described in 3.2. After some experimental tests we concluded that this set of cases was large enough, at least for the purpose of the evaluation presented in this work. As we describe later, in the experiments we have designed, the robots always attack the same goal. Hence, during the retrieval process only half of the case base (68 cases) is actually processed in the search due to the indexed list used to store the cases when they are loaded, i.e. we only consider those cases with attacking goal equal to the current problem to solve.

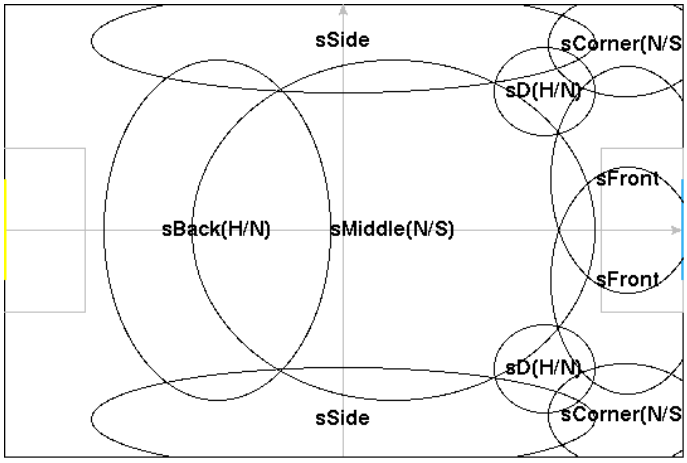
In general we can classify the cases along the following components:

1. *strategic* component: based on the region of the field that the case covers, it can vary from offensive to defensive. Regions close to the defending goal correspond to a more defensive strategy, while regions close to the attacking goal imply a more offensive strategy. Regions in the middle represent a neutral strategy.

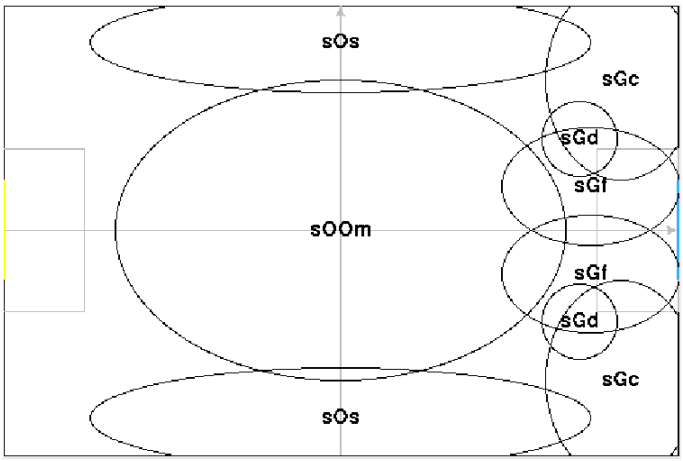
2. *teamwork* component: the number of robots (teammates) described in the case indicates the degree of teamwork, ranging from individualistic to cooperative, i.e. ranging from one robot to n robots. The larger n is, the more cooperative is the team behavior.

3. *adversarial* component: the number of opponents in the case description ranks a case from highly adversarial to non-adversarial (no opponents at all).

In this work the case base is composed of cases which combine the components defined above in different degrees. Figures 6.1 and 6.2 depict half of the case base (the 68 cases with yellow defending goal). For simplicity we only show the ball's scope, which is useful to evaluate the region of the field where the case is triggered. We define the following types of cases for our experimentation:

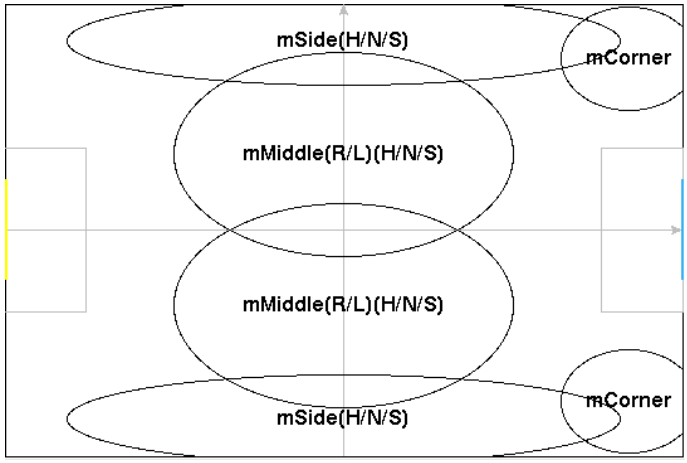


(a)

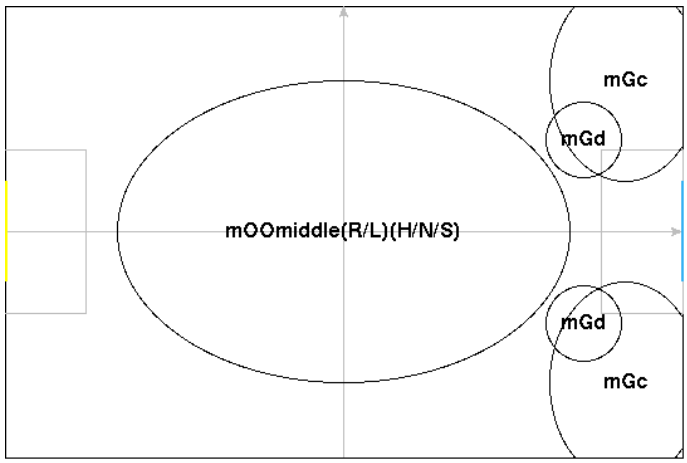


(b)

Figure 6.1: Case Base: cases with one teammate and (a) no opponents or (b) one or two opponents.



(a)



(b)

Figure 6.2: Case Base: cases with two teammates and (a) no opponents or (b) one or two opponents.

1. neutral to offensive cases: most of the cases are situated on the half of the field containing the attacking goal, although we also include some cases for the middle-back of the field in case the ball moves towards that region. In the figures we identify the following regions: front (f), corner (c), diagonal (d or D), middle (m), side (s) and back (b).

2. individualistic and cooperative cases: mainly two types of cases, either considering one teammate (single cases, Figure 6.1) or two (multiple cases, Figure 6.2). Although we are interested in achieving a high degree of collaboration among robots, there are situations where having passes between teammates is not the best choice. More precisely, this happens in those situations where the ball is in front of the attacking goal. In this case, trying to score individually may yield better results, rather than planning a pass with other teammate. As we can observe in Figure 6.2 there are no cases in front of the goal. Another reason for including single cases is to cover situations where the robots are too distant from each other. In these situations the resulting adaptation cost could be too high, and therefore, the filtering mechanism would reject all possible multiple cases as candidate solutions, considering only single cases as feasible solutions.

3. non-adversarial and adversarial cases: we include cases without opponents (Figures 6.1a and 6.2a) and with one or two opponents (Figures 6.1b and 6.2b). Since the opponents in the experiments are moving robots, it can often happen that there are no opponents near the ball. Hence, a case with no opponents can be then reused.

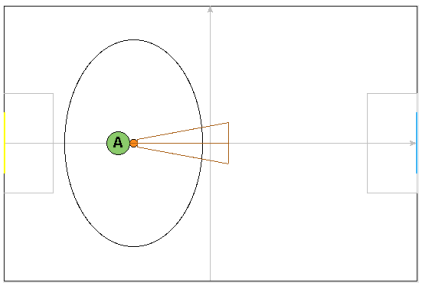
Thus, the case identifier is defined by the regular expression:

$$case\_id = (\mathrm{s}|\mathrm{m})[\mathrm{G?O}^*](region)[\mathrm{R}|\mathrm{L}][\mathrm{H}|\mathrm{N}|\mathrm{S}]$$

where, s and m indicate the teamwork degree (single or multiple); G and O correspond to the optional adversarial component (G stands for goalie and O, for opponent; the number of O's matches the number of opponents). Omitting this expression would correspond to a non-adversarial case, i.e. no opponents; $region$ corresponds to the strategic component represented by either the complete name region or its first letter; and finally, the optional reduced solution description, represented by two parameters that indicate the side of the attacking goal to point the kick (right side or left side) and the strength of the kick (hard, soft, or normal if no parameter is set).

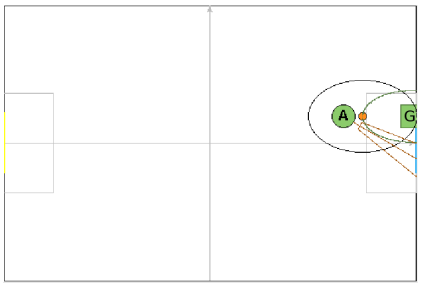
Thus, for instance, the encoded identifier `mMiddle(R/L)(H/N/S)` in Figure 6.2a corresponds to cases in the middle of the field with two teammates (multiple) without opponents, performing a hard, normal or soft kick (H/N/S) towards either the right side or the left side of the attacking goal (L/R). Figure 6.3 illustrates four case examples.

We do not include the time and score difference features of the problem description of a case to simplify the experiments. Hence, we set both indices in the cases and the problems to solve to default values, i.e. $t = 0$ and $S = 0$, so their resulting similarity is equal to 1 ($sim_{tS}(t_p, S_p, t_c, S_c) = 1.0$). Regarding the thresholds for the ball similarity (thr_b) and the adaptation cost (thr_c) used in the filtering mechanism (Section 3.3.4, Algorithm 1) we set them to the following values: $thr_b = 0.367$ (which corresponds to the Gaussian projection on



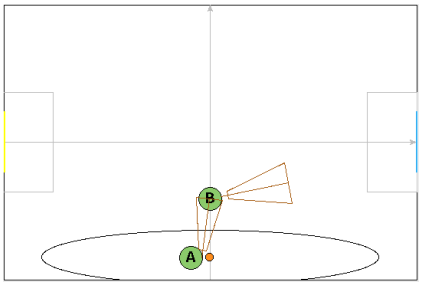
ID: sBack
#tm: 1
#opp: 0
region: back
sol: normal front kick

(a)



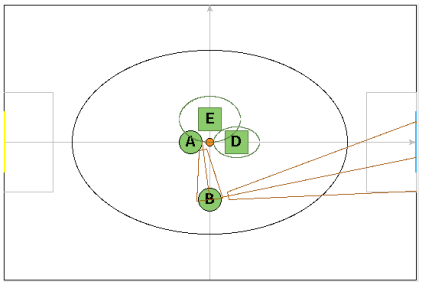
ID: sGf
#tm: 1
#opp: 1 (goalie)
region: front
sol: turn to free goal and kick

(b)



ID: mSideS
#tm: 2
#opp: 0
region: side
sol: left pass and soft kick to goal

(c)



ID: mOOmRH
#tm: 2
#opp: 2 (defenders)
region: middle
sol: right pass and hard kick to right side of the goal

(d)

Figure 6.3: Case examples: (a) sBack, (b) sGf, (c) mSideS, and (d) mOOmRH. The small circle is the ball. The big circles A and B correspond to the teammates, and the squares D and E, to the opponents. The ellipses represent the case scope, both for the ball and the opponents, while the trapezoids correspond to the ball's path.

the XY plane) and $thr_c = 1500$mm (obtained through empirical experimentation). The sorting function for ranking the candidate cases is set to *sort2* as we discussed in Section 3.3.5. Finally, with respect to the multi-robot system, since we are dealing with only two robots, they both are *retrievers* ($k = 2$), i.e. both are able to reason and to propose solutions to the problems presented in the experimentation stage.

6.2 Experiments Setup

Two types of experiments were performed: experiments with simulated robots and experiments with real robots. As in most robotic domains, the first experiments are performed in a simulated environment in order to easily detect problems and therefore, correct and improve the evaluated approach until obtaining satisfactory results. Once this stage is achieved, the next step is to experiment with the real environment, i.e. the robots, and thus, prove that the approach works as expected.

For both experiments we initialize a trial positioning the robots (two attackers vs. a defender and a goalie) and the ball in a fixed location. A trial ends either when the ball goes out of the field, enters the goal, or the goalie touches it.

6.2.1 Robot's Behaviors

The attackers are the players to be evaluated, i.e. they use either the CBR approach or the reactive approach. We must recall that within the CBR approach the robots may perform the region-based algorithm, i.e. the reactive approach, when no case is retrieved as explained in Section 4.1.1. However, these situations usually occur when the cost of any of the available cases is over the cost threshold, and not because there are no cases defined. Hence, while the attackers move towards the ball performing the region-based algorithm, they reduce their distances with respect to the ball. At some point, any of the available cases that was previously filtered out due to cost issues, may now become a candidate solution.

Both approaches also share a behavior, which we will refer to as the *default* behavior. This behavior consist in moving next to the ball, far enough to not interfere in its movements, but close enough to easily approach it and take it (around a meter away). Within the CBR approach the robots performing this behavior are those robots that do not take part of the case reuse, or that finished their gameplays while the case reuse continues (i.e. when the robots are in the WAIT END state detailed in Section 4.2). Regarding the reactive approach, the robots perform the default behavior when a robot indicates that it has the ball, and therefore, the remaining ones have to move away from the ball's path.

With respect to the opponents, we have implemented a simple behavior for the defender and the goalie. Both perform the same behavior when playing against any of the two evaluated approaches (reactive and CBR).

We define the *action region* as the region of the field where the robot can freely move, go after the ball and perform any action with it. The robot cannot move outside its action region. Thus, when the ball is within the robot's action region, the robot grabs it and kicks it towards the center of field to prevent the

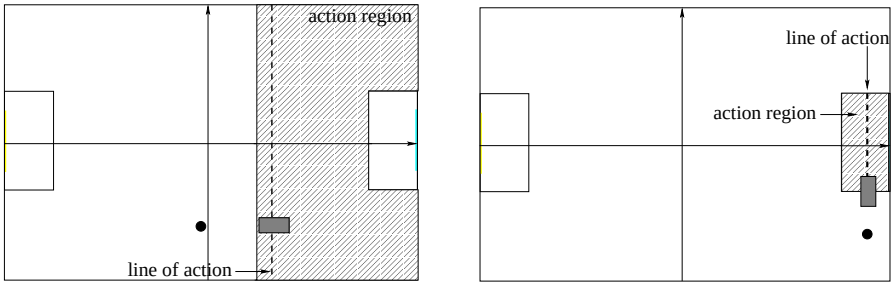


Figure 6.4: Action region for (a) the defender and (b) the goalie. The robots are facing the ball since it is out of their action region.

attackers from trying to score. We call *line of action* the imaginary line parallel to the y axis where the robot waits until the ball enters its action region. Hence, if the ball is out of the robot's action region, then the robot remains on its *line of action* facing the ball, perpendicular to the ball's position.

Figures 6.4a and 6.4b depict the action region for the defender and the goalie respectively, defined for the experiments. As we can observe, the defender's action region is also limited by the penalty area, since the rules of the robot soccer forbid any robot to enter its own penalty area, except for the goalie, of course. Regarding the goalie, its action region corresponds to the penalty area (although in a real game it can indeed walk out of this region). We have set the defender's line of action to $x = 900$, i.e. 90cm away from the midfield line, while the goalie's to $x = 2400$, close to the middle of the penalty area.

6.2.2 The Scenarios

We have defined four scenarios for the experimentation stage. We can classify them in two sets:

- Scenarios 1 and 2 (Figures 6.5a and 6.5b): the ball (small circle) and the attackers (A and B) are positioned in the middle-back of the field, while the defender (D) remains in its line of action facing the center of the field without blocking the ball, and the goalie (G) is situated within the penalty area. These scenarios correspond to general situations where the attackers are coming from the back of the field towards the attacking goal, while the defender is waiting at its position.

- Scenarios 3 and 4 (Figures 6.5c and 6.5d): the ball and attackers are located in the middle-front of the field, the goalie remains within the penalty area facing the ball and the defender is right in front of the ball. These type of scenarios are more interesting from a strategic point of view, since the first decision (action) the attackers make (execute) is critical in their aim to reach the goal while avoiding the defender whose main task is either to intercept or to steal the ball.

We believe that these two sets of scenarios are general enough to represent the most important and qualitatively different situations the robots can encounter in a game. Initiating the trials on the left or right side of the field

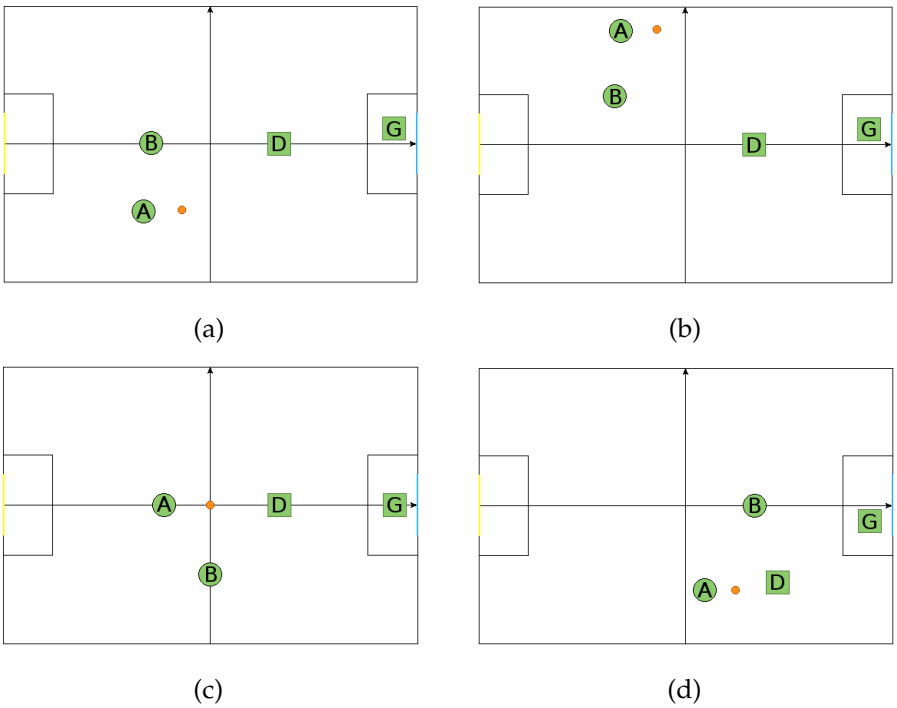


Figure 6.5: Scenarios used during the experimentation. Teammates are represented with circles, while opponents, with squares. (a) Scenario 1, (b) scenario 2, (c) scenario 3 and (d) scenario 4.

does not make much difference on the actions the robots might perform in any of the two evaluated approaches, since they would perform their symmetric actions instead.

We have neither defined any scenario with the ball near the attacking goal because the defender would not be able to do much since it cannot enter the penalty area, as mentioned in the previous section. Instead, we are interested in having the defender as an active opponent complicating the attackers' task.

Finally, regarding the corners, although they are also interesting areas to evaluate, we have not included any specific scenario with this initial layout because the big challenge within the corners is not really focused on the strategy to use, but on improving the localization of the robots. Computing the position of the robot with a minimum degree of accuracy when it is located in a corner is a very difficult localization task. The visible objects the robot can detect from that position are not enough to ensure a robust localization. Hence, we preferred to omit these initial situations because there are high chances for both approaches to perform poorly. Nevertheless, during the experiments the ball can end in a corner situation, and the approaches must somehow overcome these situations for the robots to achieve their goal.

6.2.3 Evaluation Measures

We have defined two main measures to assess the performance of the compared approaches. The first one is based on the final outcome of a trial, while the second one is based on the opponent's (more precisely, the defender) possession of the ball during the trial (a similar evaluation is performed in [15]).

As mentioned before, a trial ends when either the ball goes out of the field, enter the goal or the goalie blocks it. In order to evaluate each trial we classify the possible outcomes as:

- *goal*: the ball enters the goal.
- *close*: the ball goes out of the field but passes near one of the goalposts. More precisely, at most 25cm to the left (right) of the left (right) goalpost.
- *out*: the ball goes out the field without being a goal or close to goal.
- *block*: the goalie stops or kicks the ball.

We also consider the *to goal* balls, which correspond to balls that are either *goals* or *close* to goal. This measure indicates the degree of goal intention of the kicks. Thus, although the balls might not enter the goal, at least they were intended to do so.

Regarding the ball's possession by the defender, for every trial we count the number of times that the defender touched or kicked the ball away. This measure shows the effectiveness of a cooperative behavior. We can intuitively state that having a pass when a defender is in front reduces the chances of the defender to get the ball, if the pass does not fail. Therefore, the likelihood of successfully completing the task increases.

6.3 Simulation Experiments

In this section we evaluate and discuss the experiments performed in simulation using the four scenarios described before. The goal of this experimentation stage is mainly to verify hypotheses 1 and 3.

6.3.1 The Simulator

The simulator used for this part of the experiments is *PuppySim 2*, created by the Carnegie Mellon's team. We had to implement some additional features for our experiments, such as managing team messages, robots walking while grabbing the ball, etc. The final version of the simulator is a simplified version of the real world. The robots' perception is noiseless, i.e. the ball's position and the location of all robots on the field is accurate. However the actions the robots perform have a certain degree of randomness. The kicks are not perfect and the ball can end in different points within its trajectory (defined in Section 3.3.3). In addition, when the robot tries to get the ball, it does not always succeed, simulating a "grabbing" failure (a very common situation with the real robots). The ball's movement is modeled taking into account the friction with the field, starting with a high speed and decreasing through time and gradually ceasing

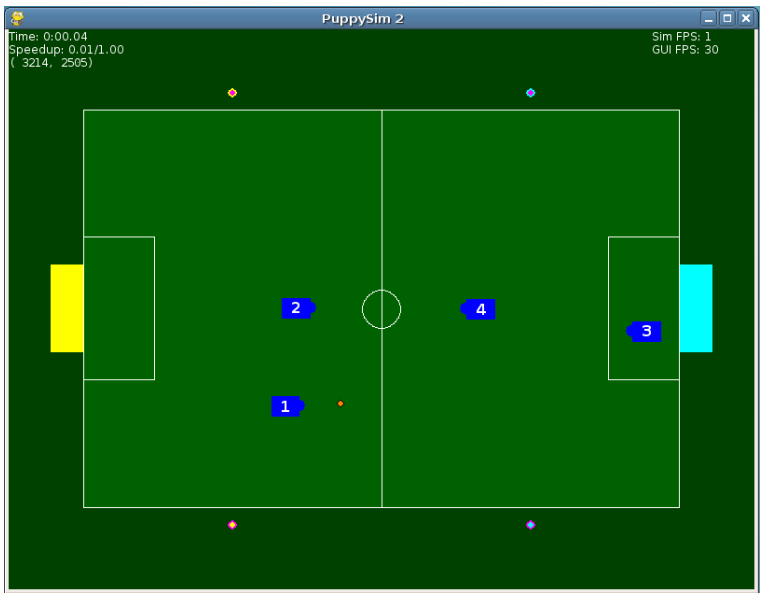


Figure 6.6: Snapshot of the robot soccer simulator PuppySim 2. Robots 1 and 2 correspond to the attackers (A and B), while robots 3 and 4, to the goalie and the defender respectively.

(if no one intercepts it before). A snapshot of the simulator is presented in Figure 6.6.

When a trial ends, the simulator stores the ball outcome (based on the ball classification stated before), the position where the ball ended and the duration of the trial in seconds. Then it restarts a new trial. This information is afterwards used to compute the statistics shown in the next section.

6.3.2 Simulation Results

We performed 500 trials for each approach and each scenario, i.e. a total of 4000 trials. Table 6.1 shows the ball classification outcome obtained for all four scenarios (results in percentage). We also computed the *to goal* measure, which results from the sum of the *goal* balls and *close* balls.

As we can see the percentage of balls *to goal* with the CBR approach is higher in all four scenarios compared to the reactive approach. Moreover, the percentage of balls *out* are lower when using the CBR, indicating that the defender had less opportunities to take the ball and kick it out of the field. The differences are specially significant in scenarios 3 and 4, where the initial position of the defender is right in front of the ball. In these situations, it is difficult for a robot to move with the ball by itself without losing it, which is what the reactive approach would do. Thus, the chances for the opponent to steal the ball increase. On the contrary, performing a pass between teammates is more useful, since the team keeps the possession of the ball, decreasing the opportunities for the defender to take it. This is the aimed strategy using the CBR approach.

Figures 6.7 and 6.8 graphically compare the ball classification outcome be-

scenario	approach	**ball classification (%)**				
		goal	close	out	block	to goal
1	cbr	25	9	28	38	**34**
	reactive	25	3	37	35	28
2	cbr	26	8	28	38	**34**
	reactive	25	6	41	28	31
3	cbr	25	6	**40**	29	**31**
	reactive	13	4	59	24	17
4	cbr	36	8	**11**	45	**44**
	reactive	22	4	25	49	26

Table 6.1: Ball outcome classification (simulation).

scenario	approach	**ball possession**	
		average	stdev
1	cbr	1.34	1.37
	reactive	1.91	1.39
2	cbr	1.38	1.29
	reactive	2.13	1.82
3	cbr	1.35	1.23
	reactive	2.20	1.33
4	cbr	0.43	0.94
	reactive	0.85	1.42

Table 6.2: Defender's ball possesion (simulation).

tween both approaches (the CBR approach outcome on the right column and the reactive approach results on the left). We can easily observe that the density of points corresponding to *out* balls is higher for the reactive approach, as the percentage in Table 6.1 shows. More interestingly, with the CBR approach these points are mainly located on the half of the attacking field (i.e. the right half side of the field), while with the reactive approach, they are dispersed along the width of the field (specially in the third scenario, Figure 6.8b, where the points are even more concentrated on the left half of the field, rather than on the right half). This occurs because the defender had more opportunities to steal the ball close to the middle of the field before the attackers managed to reach the middle-front of the field (right side of the field closer to the attacking goal). The defender then clears the ball sending it towards the back of the field, and easily kicking it out of the field. We can also notice that, in general, the density of *out* balls next to the attacking goal (i.e. points with y coordinate within the interval [-1500..1500]) is higher for the CBR approach. Thus, we can deduce that at least the attackers were definitely aiming at scoring more times that the attackers with the reactive approach.

Table 6.2 summarizes the defender's performance during the experimention. It shows the average and the standard deviation of the number of times the defender either touched the ball or kicked it per trial. We can see that in general the defender playing against the reactive approach had more chances for reaching or taking the ball than when playing against the CBR approach. Furthermore, in the last scenario, it even doubled the average. The higher aver-

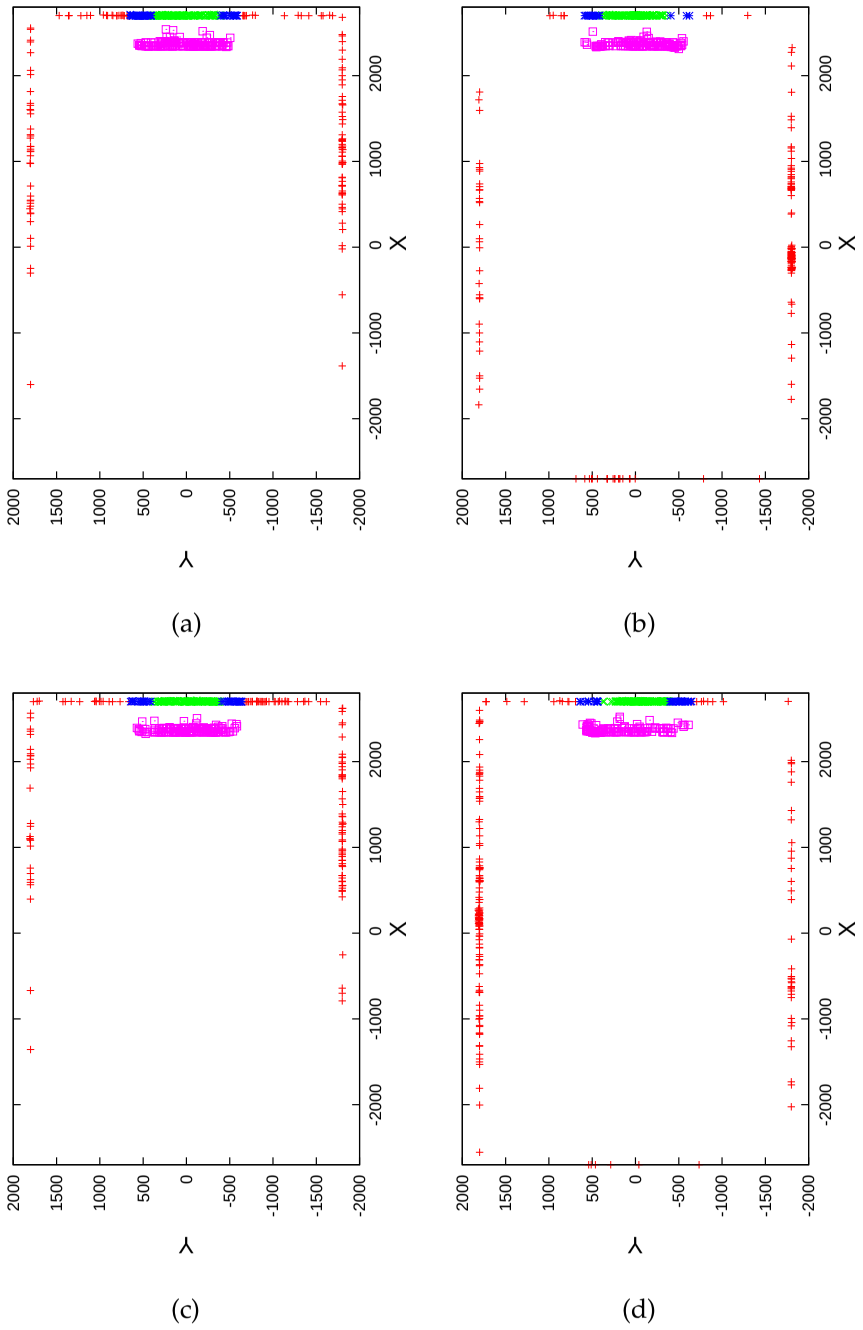


Figure 6.7: Ball classification outcome (simulation): (a) and (b) correspond to Scenario 1, while (c) and (d), to scenario 2. Left figures result from the CBR approach performance, while right figures, from the reactive approach performance. Red crosses (+) represent *out* balls; green x's (×), *goal* balls; blue stars (∗), *to goal* balls; and pink boxes (⊡), *blocked* balls.

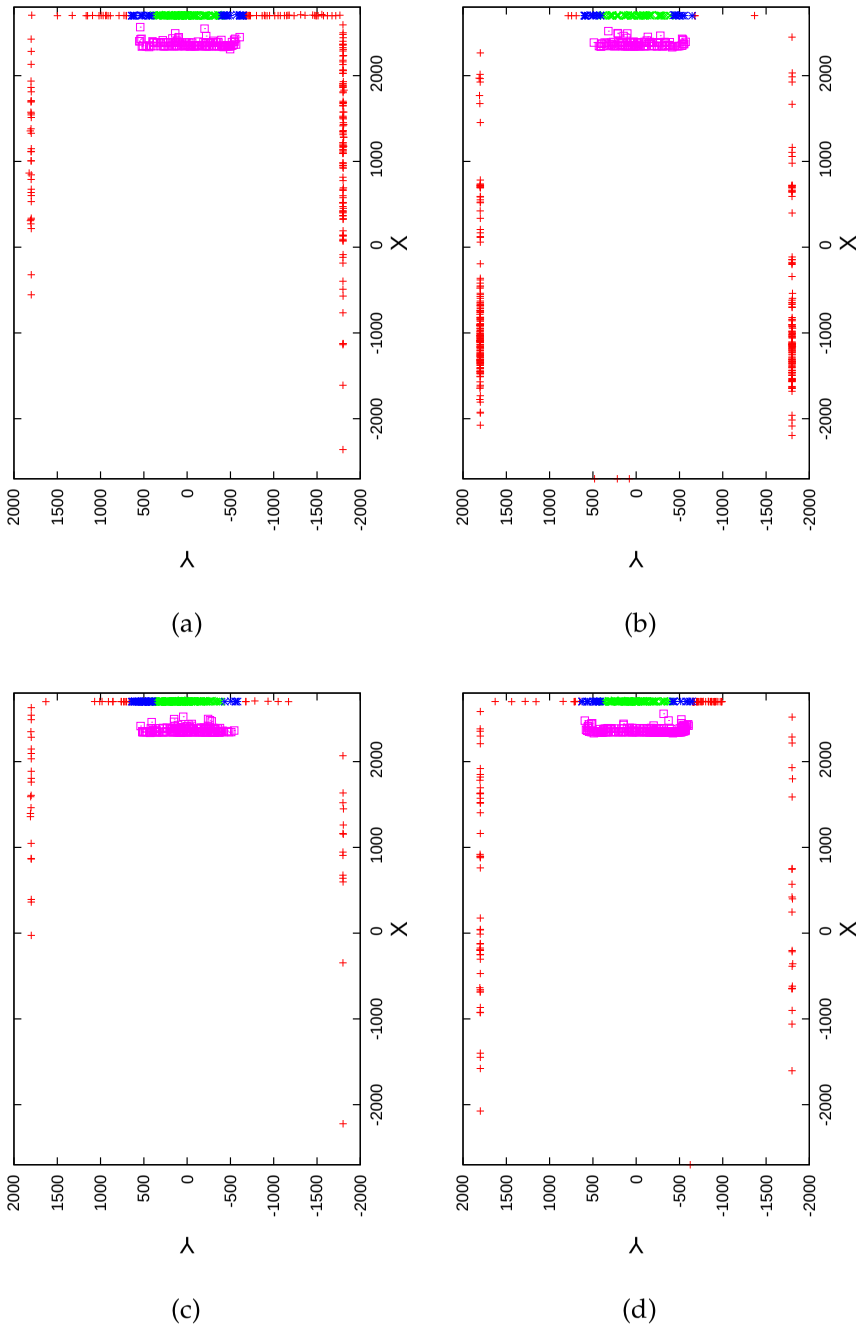


Figure 6.8: Ball classification outcome (simulation): (a) and (b) correspond to scenario 3, while (c) and (d), to scenario 4. Left figures result from the CBR approach performance, while right figures, from the reactive approach performance. Red crosses (+) represent *out* balls; green x's ($\times$), *goal* balls; blue stars ($*$), *to goal* balls; and pink boxes ($\boxdot$), *blocked* balls.

	scenario 1		scenario 2		scenario 3		scenario 4	
robot	A	B	A	B	A	B	A	B
AV	3.3	16.0	7.2	26.2	2.8	18.6	16.5	21.6
%	17	83	21	79	13	87	43	57

Table 6.3: Average and percentage of the backing up times per robot (A or B) and scenario.

	scenario 1		scenario 2		scenario 3		scenario 4	
case	single	mult	single	mult	single	mult	single	mult
AV	3.9	5.0	4.1	6.0	3.6	5.3	2.4	2.5
%	44	56	41	59	40	60	49	51

Table 6.4: Average and percentage of "single" and "multiple" cases used during the experimentations.

age values for both approaches correspond to the first three scenarios since in these scenarios the ball is located further from the goal compared to the fourth scenario. Hence, the chances for the defender to steal the ball are higher since the distance the attacking robots have to travel to reach the goal is longer.

In order to show the degree of the collaboration among robots we computed two more measures in this experimentation set. As we described in Section 6.2.1, the reactive approach provides the robots with a simple coordination mechanism: while a robot possesses the ball, the second robot performs a default behavior to avoid interfering with the first one. Thus, in general, during a trial the robot starting the first action (e.g. get the ball and kick it) moves with the ball while the second one is backing up. Once the action ends, both robots will try to get near to the ball, but the chances for the second robot to arrive first are lower since it had previously moved away from the ball. The first robot instead, has more chances to get the ball first, while the second robot will have to back up again and again. For each trial, we counted the number of times each robot backed up. Table 6.3 shows the average and percentage of the number of times the robots backed up per trial. As we can see, except for the last scenario, the percentage of times that robot A backs up is significantly lower compared to robot B. Hence, we can conclude that in general, because of the strategy used, robot A (the robot that gets the ball first) acts individually without integrating robot B (the other) in the task.

Since the reactive and CBR approaches are very different, we cannot apply the "number of backing ups" measure to the latter one. Therefore, to demonstrate collaborative behavior with the CBR approach, we counted the number of reused cases that implied a single robot in the solution, or more than one (in this work two robots) during the experiments. We label "single" cases to those cases with only one robot in their case description, and "multiple" cases to those cases with more than one robot. The percentage of the type of cases used and the average per trial is detailed in Table 6.4. As we can observe, in general, half of the time (or even slightly more) the robots retrieve multiple cases, i.e. cases where an explicit pass between two robots takes place. This is probably due to the fact that the robots start their performance in the middle of the field and with a defender in between them and the goal. In this situation a

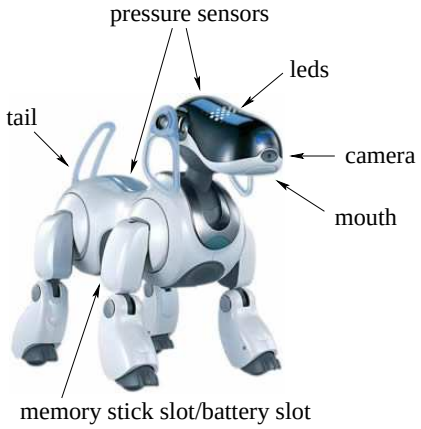


Figure 6.9: Sony AIBO ERS-7(M2-M3) robot description.

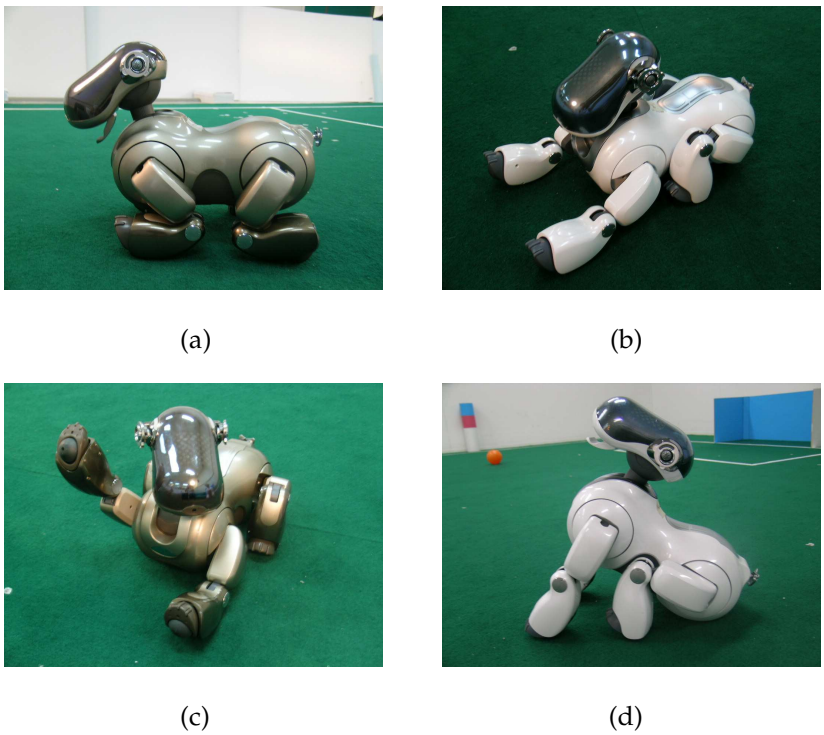
cooperative strategy (multiple cases) is more useful since the robots can work together to get closer to the goal. Once they get near the penalty area, it makes more sense to try to score individually (simple cases), and not to have passes between teammates.

6.4 Real Robot Experiments

After the successfull results in simulation, the next step is to test similar experiments, but this time with the real robots. We must keep in mind the difficulties that arise when dealing in a real environment, increasing the complexity of the problem to solve. More precisely, the uncertainty in the incoming information (robots' perception), which has not been taken into account in the simulated environment. Hence, with this second experimentation stage we aim to verify hypothesis 2, i.e. the ability of the approach presented in this work to handle uncertainty, as well as to reinforce hypotheses 1 and 3, already supported in the previous experimentation stage.

6.4.1 The Robots

The robots we have used in this experimentation stage are four Sony AIBO ERS-7 robots (one M2 robot, and three M3 robot). The AIBO robot is a four-legged robot with a dog shape (Figure 6.9). The dimensions of the robot are 180×28×319 (in mm, width×height×length). A camera is located in its nose with a field of view of 56.9° wide and 45.2° high. It has 18 PID joints, each with force sensing: 3 joints per leg (elevate, rotate, knee), 3 joints on the neck (tilt, pan, nod), 2 joints on the tail (tilt, pan) and 1 joint on the mouth. Its internal CPU is a 576MHz processor with 64MB RAM. As we can see, it is a very limited processor and therefore requires the implementation of fast and simple algorithms. The programs are copied to a Memory Stick that is inserted in the robot. When the robot is turned on, it loads all the information from the Memory Stick and starts moving autonomously. The robots communicate through a standard wireless network card (802.11b wireless ethernet). The robot has 26



(a) (b)

(c) (d)

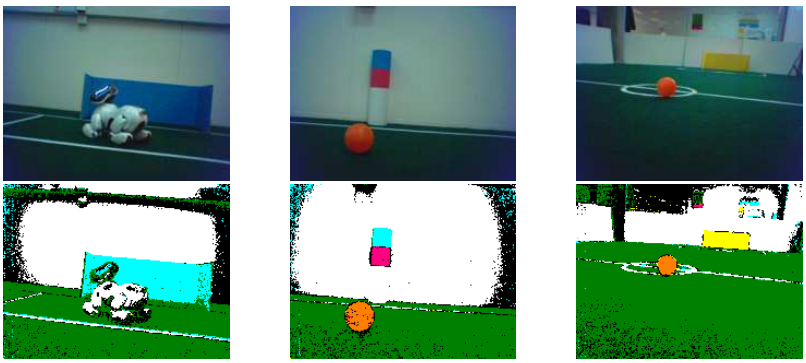
Figure 6.10: The robots: (a) Fang (Mud), (b) Boira (Mist), (c) Terra (Soil) and (d) Nata (Cream).

independent leds on its face which are useful for debugging the robot's behavior. It also has four pressure sensors (three on its back and one on its head) employed to modify the robot's behavior (pause, resume, reset, etc.). Pictures of our four robots are shown in Figure 6.10.

Vision Issues

The vision system of the robots is in charge of identifying and localizing the objects in the environment, i.e. ball, markers, goals, lines and other robots. Hence, a robot is capable of knowing its own position on the field (its localization is based on the observed markers positioned along the field) and deriving the locations of the objects in its field of view accordingly. Because of computational limitations the vision processes must be fast and simple. Therefore, the robustness of the vision system is not guaranteed. The main efforts of the designers are focused on rapidly detecting the ball and markers on the field, which are the most important objects for the robot to perform its task, i.e. move the ball towards the goal. Hence, although there is an attempt to also identify and localize opponents, in fact, the robots can hardly know where the opponents are with a minimum degree of accuracy. Figure 6.11 illustrates some examples of images extracted from the robot vision system, both original images (RGB) and after the segmentation process.

The purpose of this research is to study the performance of the approaches,



(a) (b) (c)

Figure 6.11: Top: original images (RGB), and bottom: segmented images. (a) Cyan goal and a robot. (b) Marker and ball. (c) Yellow goal and ball in the center of the field.

and not to improve robustness to the perception system. Since the opponents locations are fundamental for the experimentation we present in this work, to evaluate both approaches independently from vision issues, the robots from the opponent team report their positions to all the robots on the field through the network (the same way the robots from the same team do so).

We must also mention that during the experimentation with the CBR approach, after every cycle (i.e. retrieving and executing the case) all robots stop for 5 seconds in order to update their localization on the field with lower uncertainty and thus, increase the accuracy of the case retrieval. Otherwise, the performance of the CBR approach would be degraded due to visual issues, misleading the overall evaluation of the system.

6.4.2 Results

Since working with real robots is harder than in simulation (it is unfeasible to reproduce with the real robots the volume of experimentation done in simulation), for this second part of the evaluation we only used the third and fourth scenarios. As mentioned before, we believe these are more interesting than the first two scenarios because the defender is located in front of the ball, blocking the first movement the attacker could perform. Hence, the attacker needs to apply some strategy to avoid the defender and not to lose the ball.

We performed 30 trials per approach and per scenario, 120 trials in total. Next we evaluate both scenarios separately discussing for both approaches: first, the approach performance; second, the ball classification outcome; and finally, the defender's performance.

Scenario 3

- *CBR approach performance*
  After observing the 30 trials performed by the robots, we sketch the general behavior of the CBR approach in Figure 6.12. As we can observe,

given the initial positions of the robots, the first action is to perform a pass to avoid the defender (Figure 6.12a). Hence, robot A moves towards the ball to start the pass, while robot B moves towards the front to receive the ball. Meanwhile, the defender (robot D) remains on its line of action facing the ball. As the pass takes place, the defender moves to a better position to continue blocking the ball. Since robot A has ended its sequence of actions (gameplay) it performs the default behavior, maintaining a close distance to the ball, but without going after it. When robot B receives the ball, it performs a kick towards the middle line (Figure 6.12b). The first case reuse ends. The next case consists in moving the ball forward in order to move it closer to the attacking goal. Hence, as robot A is closer to the ball, it is in charge of reusing alone the second case, while robot B moves next to the ball towards a better position executing the default behavior. Meanwhile the defender tries to reach the ball as well (Figures 6.12c and 6.12d). Finally, the last case is retrieved, which once again consist in having a pass between robots A and B to avoid the goalie (robot G). Hence, robot A moves to take the ball, while robot B waits for the pass (Figure 6.12e). Once it receives the ball, it kicks towards the goal (Figure 6.12e).

The sequence detailed above is a perfect execution, where the attackers manage to score and the trial ends. Unfortunately, because of the high imprecision of the action executions, the performances of the trials varied from one to another retrieving different cases (thus, executing different actions) to overcome the altered sequence. The critical points where a modification of the ideal execution occurs are:

- during a pass (Figures 6.12b and 6.12f): the pass could fail because (*i*) the ball is sent to the wrong direction (usually due to wrong localization of the robots), (*ii*) the receiver does not succeed in grabbing the ball, or (*iii*) the defender intercepts the pass.
- during the adaptation of the case (Figures 6.12c and 6.12e): while the robot is moving towards the ball, the defender may reach the ball first, clearing the ball or kicking it out of the field.

- *Reactive approach performance*
  The approach only takes into account the position of the opponent for making decisions when the opponent is very close to the ball (approximately 40 cm away at most), blocking it from a forward kick. Hence, in the initial trial layout, the defender is far enough from the ball to consider it during the decision making and therefore, robot A first performs a forward kick (Figure 6.13a). In the next timestep, the ball is close enough to the defender and thus, the reactive approach includes it as an obstacle that must be avoided. Since explicit passes are not modeled in this approach, the only chance for avoiding the opponent is to dodge it, moving in diagonal (either to the right or to the left) while grabbing the ball as shown in Figure 6.13b. The opponent, in this case the defender, also moves towards the ball and both robots collide fighting for the ball. The outcome is either a success for the attacker, getting rid of the defender and kicking the ball forward, or a success for the defender, stealing the ball and clearing it.

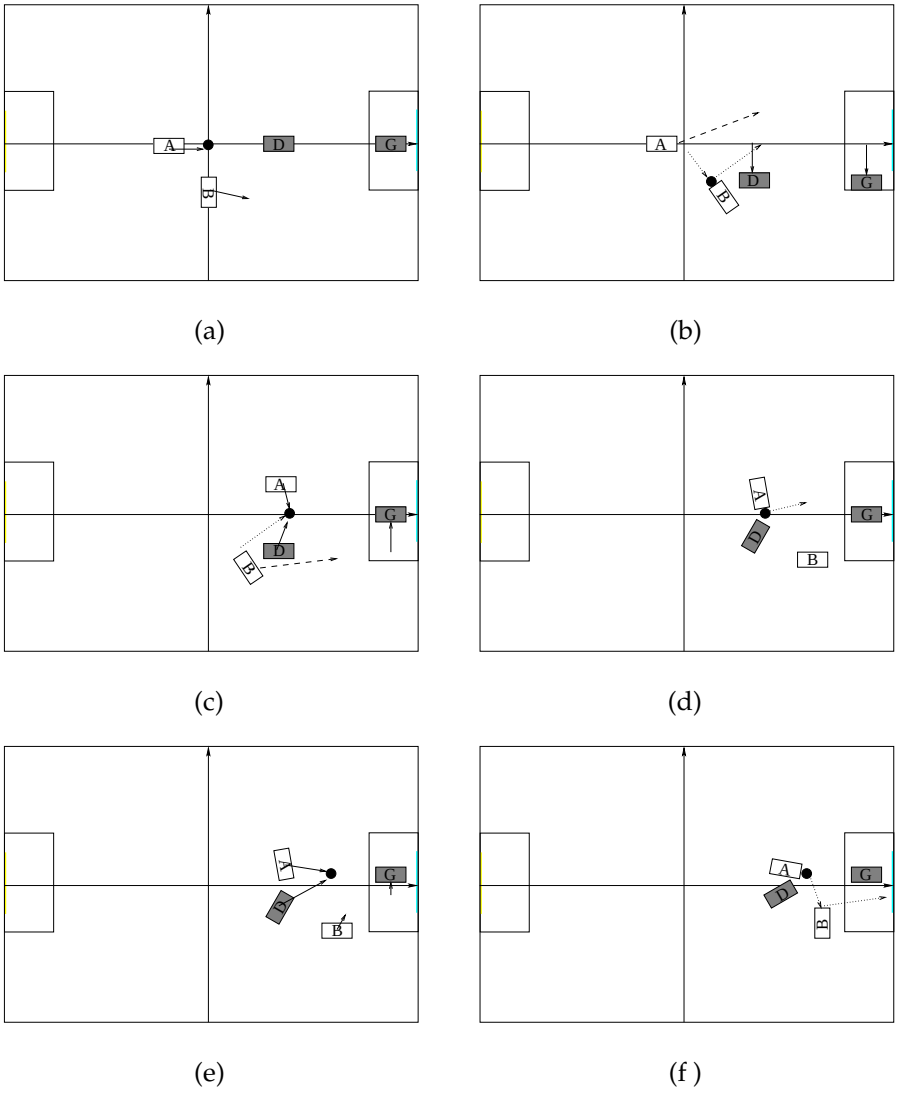
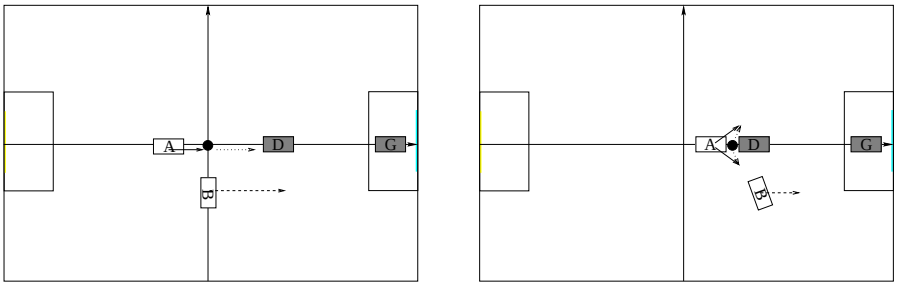


Figure 6.12: Sketch performance of the attackers using the CBR approach in scenario 3. Solid arrows represent the robots movements, dashed arrows, the default behaviors (moving next to the ball), and pointed arrows, the ball's movement.



(a) (b)

Figure 6.13: Sketch performance of the attackers using the reactive approach in scenario 3.

The overall performance of the reactive approach is the same in general, trying to move the ball close to the attacking goal, and dodging the opponent when it gets near the ball approaching from the front. At some point, the attacker reaches the attacking goal and tries to score avoiding the goalie either turning or dodging side to side.

- *Ball classification*
  The CBR approach outperforms the reactive approach. As summarized in Table 6.5 the percentage of balls *to goal* is higher for the CBR approach (30%) with respect to the reactive one (17%), as well as the percentage of *blocked* balls, i.e. 43% for the CBR approach, and 30% for the reactive approach. Hence, the chances for scoring with the CBR approach are higher, since more times the attackers reached the attacking goal, ending the trial either scoring or trying to score. This fact is also derived from the percentage of balls *out*, where we can observe that the percentage for the reactive approach (53%) even doubles the percentage for the CBR approach (27%). More precisely, as listed in Table 6.6, the number of balls *out* due to the defender's actions is higher for the reactive approach (11) with respect to the CBR approach (6).

- *Defender's ball possession*
  The chances for the defender to steal the ball are higher when the attackers use the reactive approach. Table 6.6 lists the average and standard deviation of the number of times the defender possessed the ball, i.e. either touched or kicked the ball. The average of the defender's ball possession is 2.27 in contrast to the average of 1.40 when playing against the attackers with the CBR approach. This means that in average, at least two times the defender had the opportunity to either block the ball or even worst, to clear the ball from its defending zone (the half side of the field it defends). Thus, we can state that the teamwork component in the CBR approach, more precisely the passes between teammates, are indispensable for reducing the opponent's chances to intercept the ball. This fact is also confirmed by the number of balls *out* mentioned above, where the defender kicks the ball out of the field more times when playing against the reactive approach.

scenario	approach	**ball classification (%)**				
		goal	close	out	block	to goal
3	cbr	20	10	27	43	**30**
	reactive	10	7	53	30	17
4	cbr	20	3	17	**60**	23
	reactive	30	7	30	33	**37**

Table 6.5: Ball outcome classification (real robots).

scenario	approach	**ball possession**		***out* balls**		
		average	stdev	def	att	total
3	cbr	1.40	1.16	6	2	8
	reactive	2.27	1.93	11	5	16
4	cbr	0.60	0.72	2	3	5
	reactive	1.07	0.87	5	4	9

Table 6.6: Defender's ball possesion (real robots).

Scenario 4

- *CBR approach performance*
  Similarly to the previous scenario, the first action the attackers perform is a pass between them to avoid the defender, while the latter tries to take it (Figure 6.14a and Figure 6.14b). After the first case reuse, the ball ends close to the penalty area, where the goalie is expecting it as shown in Figure 6.14c. Since the goalie is on the right side of its penalty area, it is not only blocking the ball from a front kick, but also incapacitating robot A from scoring. Hence, the only solution is for robot B to individually try to score dodging the goalie (Figure 6.14d), while the defender comes from the back trying to take the ball on time. Once again, failures during the execution can occur due to the reasons already mentioned in the previous scenario (errors during passes or defender reaching the ball first).

- *Reactive approach performance*
  In contrast to the third scenario, in this occasion the initial configuration sets the opponent close enough to the ball, so the attacker can detect it. Hence, using the dodging tactic robot A tries to avoid the defender, moving diagonally towards the left and kicking the ball forward (Figure 6.15a). Meanwhile, robot B moves towards the attacking goal, avoiding to intercept the ball. Once robot A has finished the kick, robot B can immediately go after the ball (Figure 6.15b). This action could be interpreted as a pass, although it was not really meant to be. Next, robot B is close enough to the attacking goal and alone with the goalie, and therefore, tries to score (Figure 6.15c).

  We must once again recall that the above described scenario corresponds to an ideal execution. As the results we have obtained show, most of the times the defender prevented the attackers from reaching the goal or at least, greatly difficulted their task.

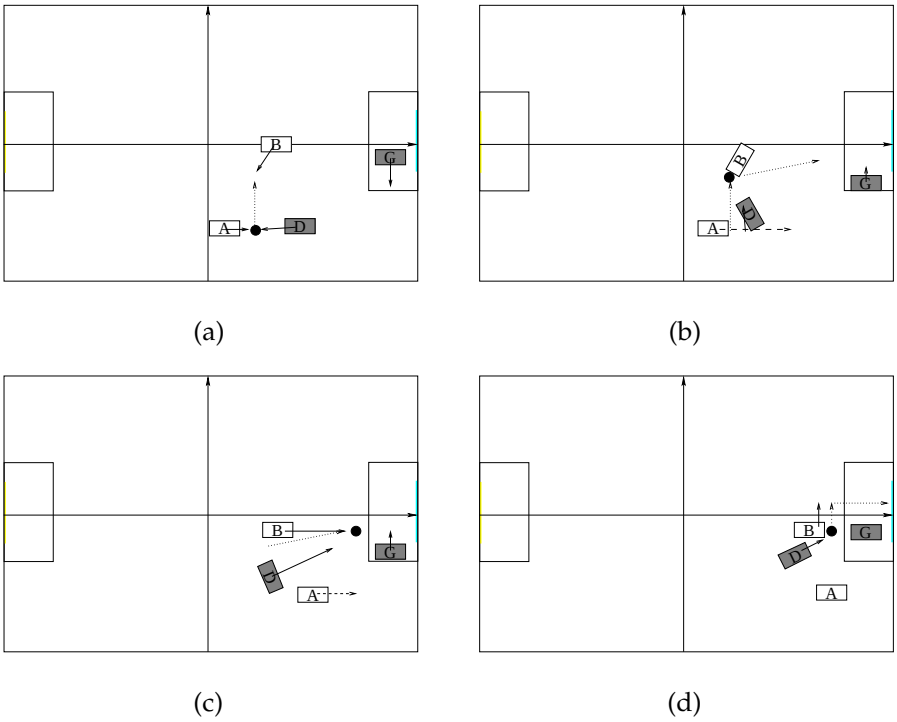


Figure 6.14: Sketch performance of the attackers using the CBR approach in scenario 4.

- *Ball classification*
  The CBR approach is not as efficient as the reactive approach. As we can observe in Table 6.5 the percentage of balls *to goal* using the reactive approach (37%) is higher than using the CBR approach (23%). However, we must also take special attention to the fact that the percentage of *blocked* balls by the goalie is much higher for the CBR approach (60%, it doubles the reactive approach). Therefore, we confirm that altough the attackers with the CBR approach did not manage to score as many goals as the attackers with the reactive approach, at least most of the times they reached the attacking goal and aimed at scoring. Moreover, as detailed in Table 6.6, while playing against the reactive robots the defender had more opportunities to kick the ball out of the field (5 times vs. 2 against the CBR approach), preventing the attackers from reaching the attacking goal.

- *Defender's ball possession*
  Similarly to scenario 3, as Table 6.6 summarizes, the average number of times the defender intercepted the ball when playing against the reactive approach (1.07) is higher than when playing against the CBR approach (0.60). As mentioned in the approach performance, the first attacker's action using the reactive approach is to dodge the defender moving forward with the ball, instead of performing a pass, as the CBR approach

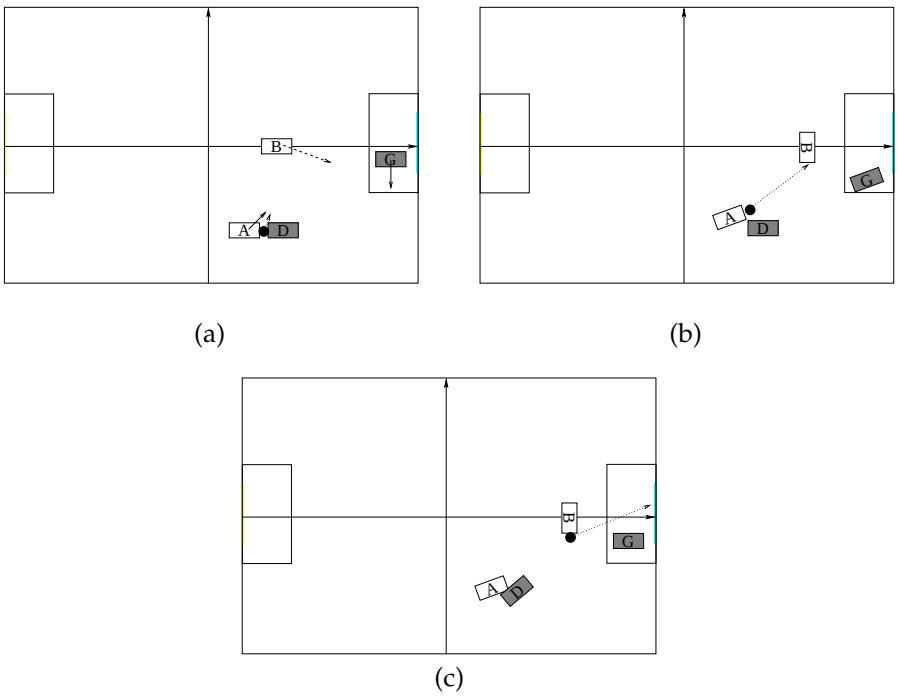


Figure 6.15: Sketch performance of the attackers using the reactive approach in scenario 4.

does. Hence, although the attacker might try to avoid the defender, most of the times, the defender manages to block its movements, forcing the attacker to lose the ball. Therefore, in average, at least one time the defender blocks the ball, complicating the task of the attacker to finally move the ball towards the attacking goal.

Further discussion on the overall performance of the approaches comparing the simulation and the real robots results is presented in the last section of this chapter.

6.5 A Trial Example

In order to complete the experimentation stage, we next detail a complete trial with the real robots, starting with the coordinator selection of the multi-robot system, continuing with the retrieval process, and finally, showing the execution of a case, i.e. the case reuse.

Figures 6.16 through 6.24 show the evolution of a trial using the fourth scenario. For each case reuse we first show the retrieved case and the paths followed by the robot. The subsequent figures show the execution sequence of the case, composed of four series of snapshots. The general description of each image is the following:

- retrieved case: similar to the figures shown in the beginning of this chapter (Section 6.1), the green circles correspond to teammates (A and B),

while squares correspond to opponents, the defender (D) and the goalie (G). The ball and opponents' scopes are depicted with ellipses, and the ball's path with trapezoids.

- path image: it illustrates the path of the robots and the ball during the case execution. As shown in the legend, green crosses correspond to robot A and blue stars, to robot B (the attackers). The defender and the goalie (D and G) are represented with a yellow and a pink square respectively. Finally, the ball is denoted by the red circle. The data is obtained from the internal beliefs of the robots, i.e. where they believe they are located on the field and what is the ball's position.
- execution sequence: each step of the case reuse is composed of three images:
  - a snapshot of the video of the trial. The brown robots correspond to the attackers, while the white ones, to the opponents.
  - the robot's internal beliefs, also called world model[1] corresponding to any of the two attackers.
  - an image extracted from one of the robots vision system (segmented images). The robots id's in the world model are 1 and 2 for the attackers, 3 for the goalie, and 4 for the defender (white or gray filled squares).

In this trial example, three cases were reused to fulfill the task. We next detail each of the steps:

1. Case 1 (Figures 6.16, 6.17 and 6.18): as observed in the first snapshot of the field, Figure 6.17 (1), the robots are located as in the initial layout of the fourth scenario. The selected coordinator corresponds to robot A, the closest robot to the ball. The world model and the segmented image were obtained from robot A. A reduced description of the problem to solve corresponds to (from robot A's perspective):

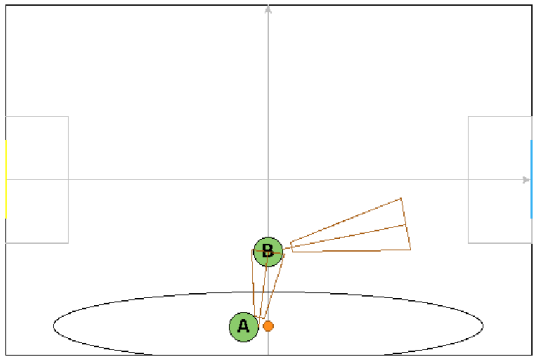
id	position (x,y)	
A	182	-1288
B	756	-79
defender	1458	-1273
goalie	2359	-163
ball	682	-1263

   The retrieved case corresponds to the case mSide (multiple-side), Figure 6.16a, which consists in a pass between both robots to avoid the defender positioned in front of the ball.

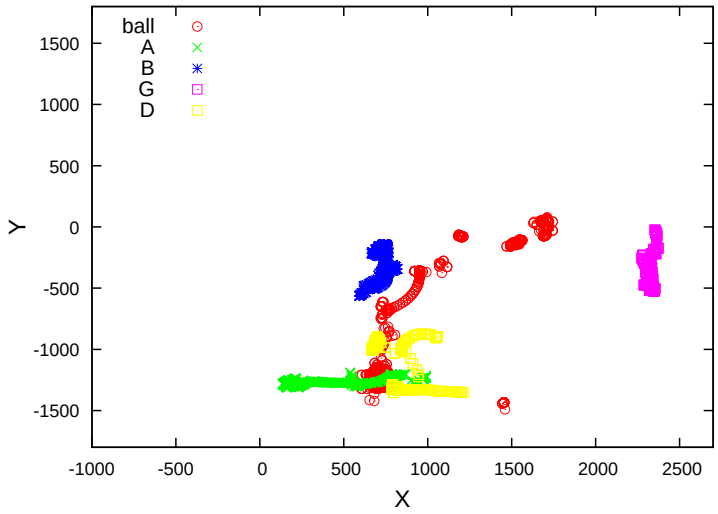
   The second row, Figure 6.17 (2), shows the robots starting the execution of the case, i.e. the case reuse. In the picture we can see that robot A is performing the pass, while robot B is waiting to receive the ball. Meanwhile, the defender has moved to intercept the pass. Next, robot B takes

[1] To obtain this information we used Chokechain2, a debugging tool implemented by the CMU team. The symbols and graphics shown correspond to features used to debug the robot's behavior. We only detail those relevant for this example.

the ball, Figure 6.18 (3), and turns to kick the ball towards the attacking goal, Figure 6.18 (4). The segmented image in this last row is taken from robot A, and illustrates the kick performed by its teammate (robot B) finalizing the execution of the case. The ball ends close to the attacking penalty area as shown in red in the path's image (Figure 6.16b).



(a)



(b)

Figure 6.16: Trial example: first case. (a) Retrieved case, (b) robots' paths.

(1)

(2)

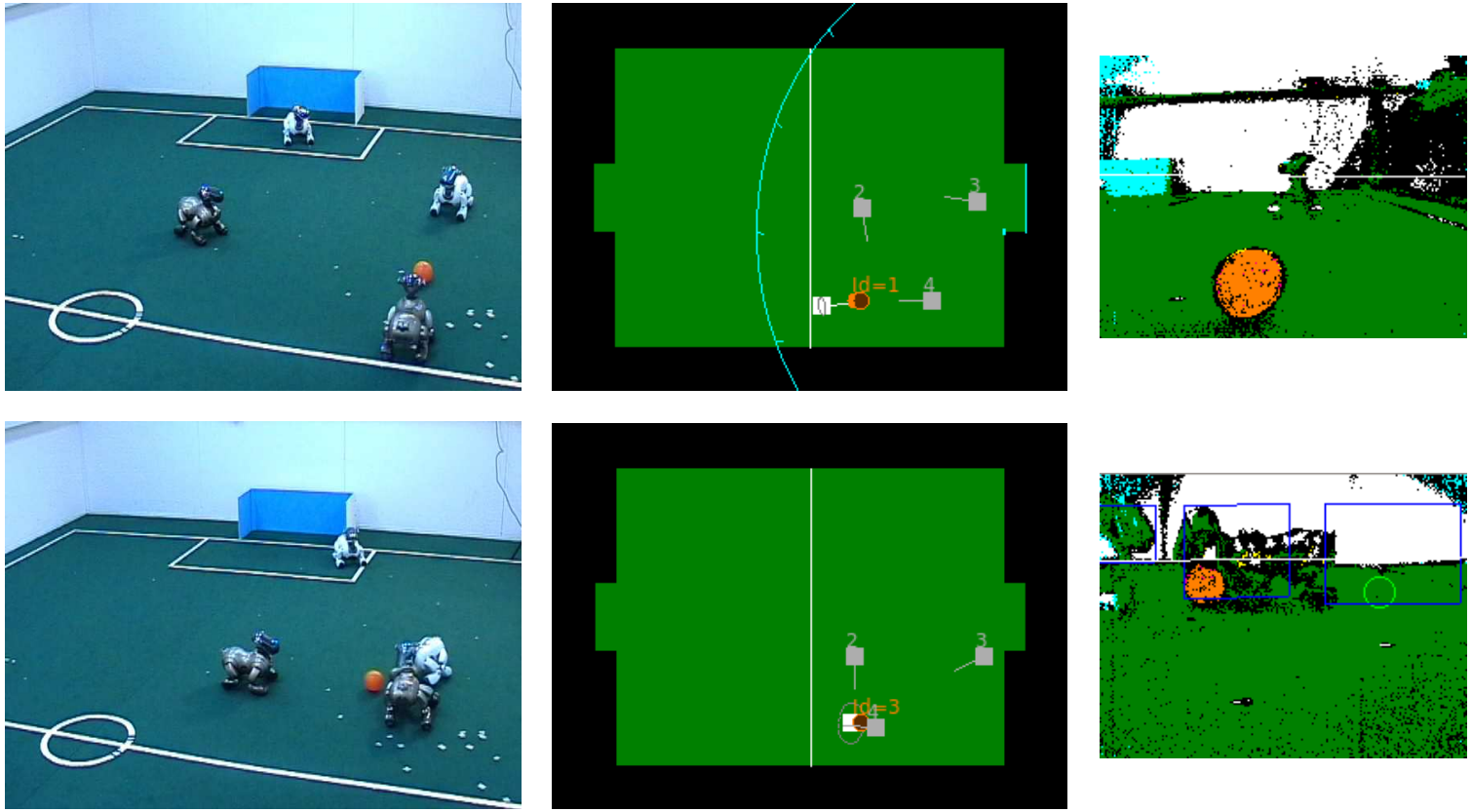
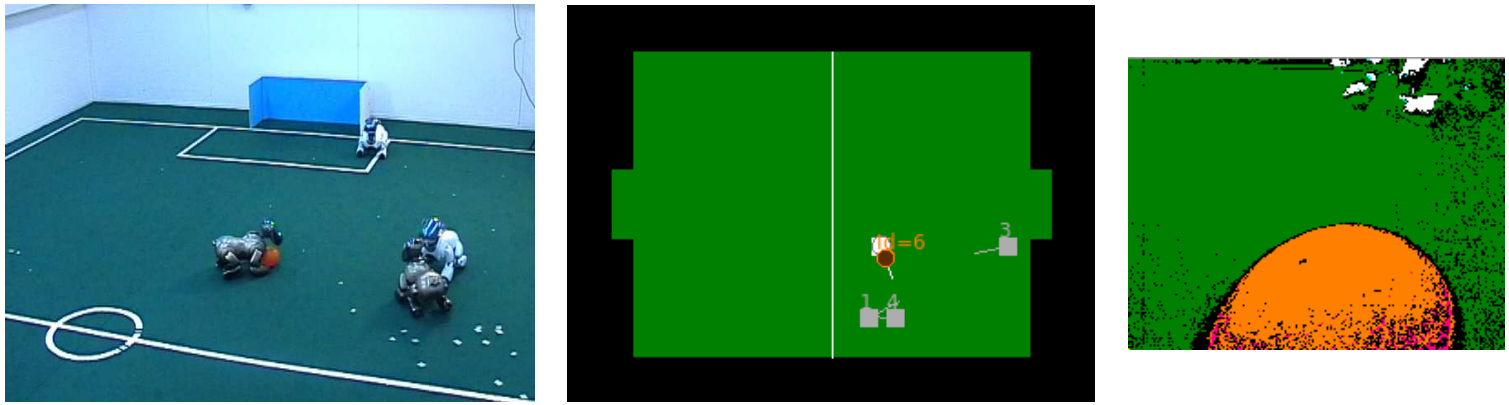


Figure 6.17: Trial example: first case (sequence 1 and 2). Snapshot, world model and segmented image.

(3)



(4)

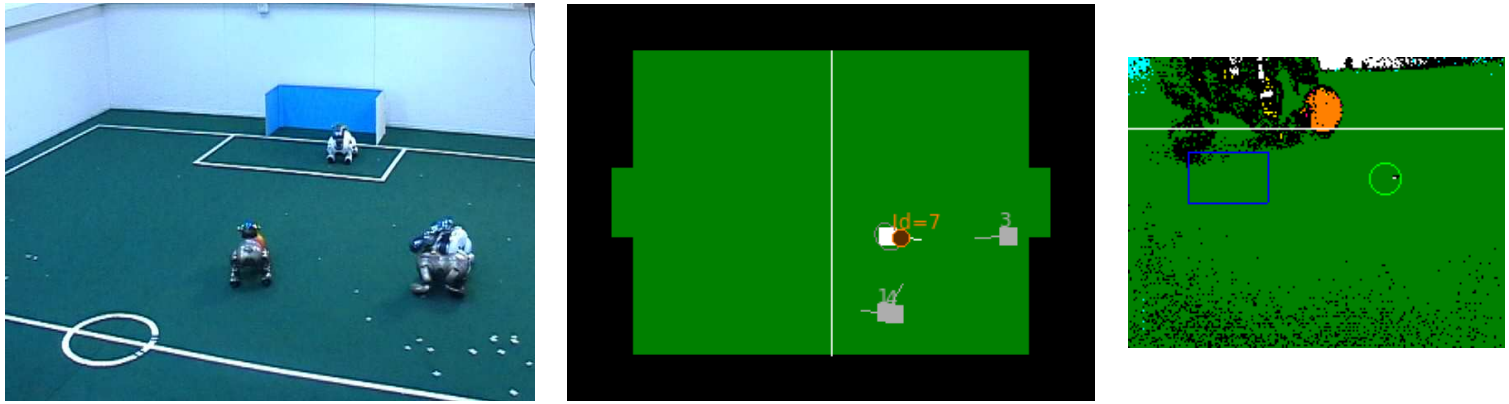


Figure 6.18: Trial example: first case (sequence 3 and 4). Snapshot, world model and segmented image.

2. Case 2 (Figures 6.19, 6.20 and 6.21): after the execution of the previous case, robot B is selected as the new coordinator. Robot A is further in the back blocked by the defender, while the goalie starts moving perpendicular to the ball to prevent a goal. The problem description corresponds to (this time from robot B's perspective):

id	position (x,y)	
A	952	-1288
B	685	-371
defender	908	-1006
goalie	2369	-140
ball	1595	-4

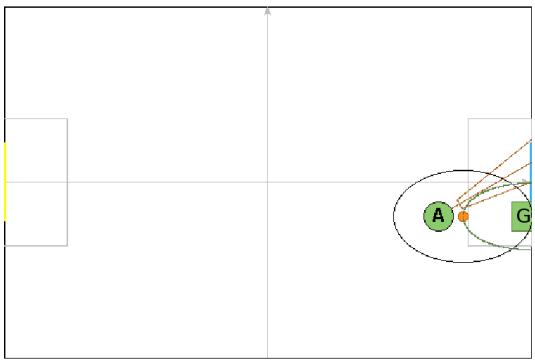
Since robot B is alone in front of the goalie, Figure 6.20 (1), case sGf (single-goalie-front) is retrieved, Figure 6.19a. Thus, the robot should take the ball and try to score alone avoiding the goalie (either turning to face an empty spot in the goal or dodging the goalie).

In the second row, Figure 6.20 (2), we can observe that robot B approaches the ball and the defender starts traveling towards that point as well. The segmented image is taken from robot A's point of view, where we can observe robot B getting closer to the ball. In the next row, Figure 6.21 (3), robot B kicks the ball while the defender tries to steal it. The defender does not manage to get the ball, although the goalie moves to block it and succeeds (segmented image in Figure 6.21 (4)). However, the trial in this occasion is not stopped, since the goalie has not kicked the ball out i.e. it just blocked the ball.

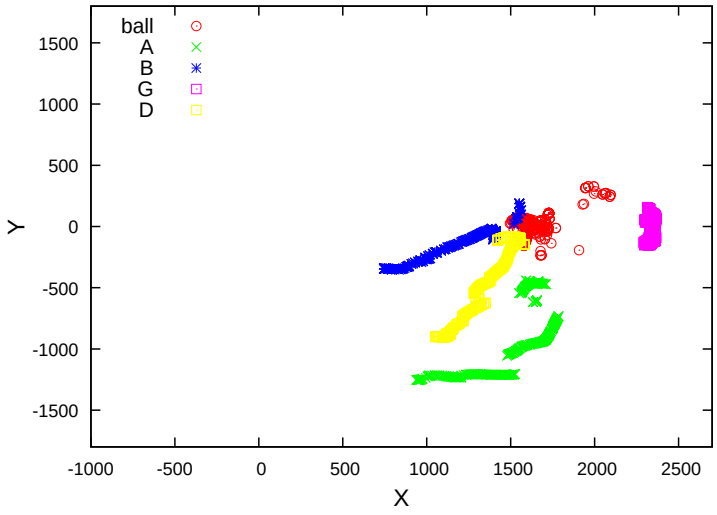
3. Case 3 (Figures 6.22, 6.23 and 6.24): in this final step, robot B is selected as the coordinator once again, although as we see next, the case is reused only by robot A. Thus, given the current state of the world:

id	position (x,y)	
A	1656	-363
B	1522	114
defender	1630	-328
goalie	2374	0
ball	1952	-42

The retrieved case corresponds to case sFront (single-front), Figure 6.22a, where no opponent is considered. Although from our perspective (human) we can clearly see in the first picture, Figures 6.23 (1), that the goalie is in front of robot A and therefore, the case sGf (as above) would be more appropriate, we must recall that a high degree of uncertainty is present within the robots' beliefs. In fact, if we observe the picture in the Figure 6.23 (2), it turns out that the goalie moves too much towards its left, leaving the goal free for a short period of time (the segmented image illustrates a free gap in the attacking goal). Thus, the case execution continues, Figure 6.24 (3) and (4), and in spite of the opponents' efforts to prevent the goal, finally robot A manages to score.



(a)



(b)

Figure 6.19: Trial example: second case. (a) Retrieved case, (b) robots' paths.

(1)

(2)

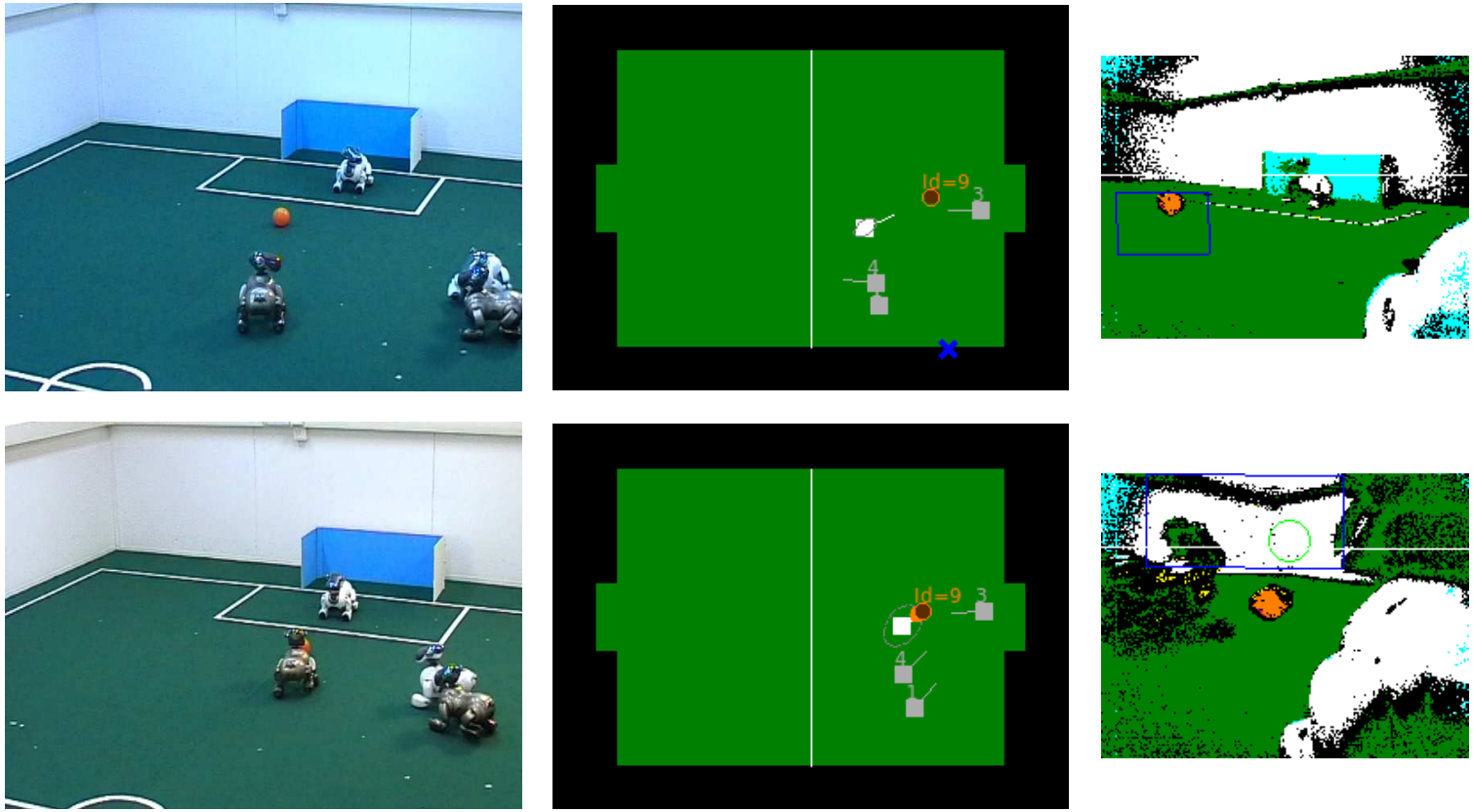


Figure 6.20: Trial example: second case (sequence 1 and 2). Snapshot, world model and segmented image.

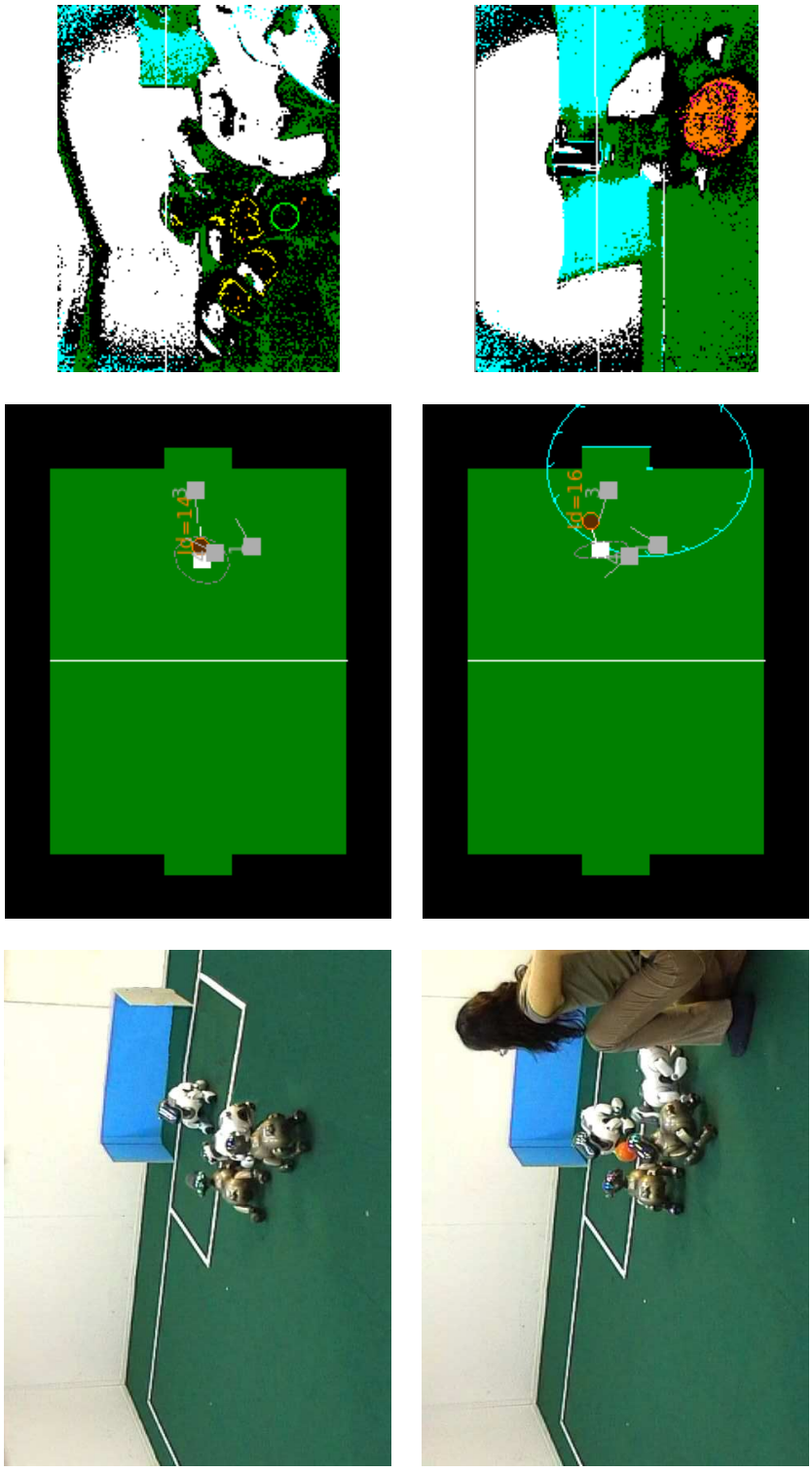
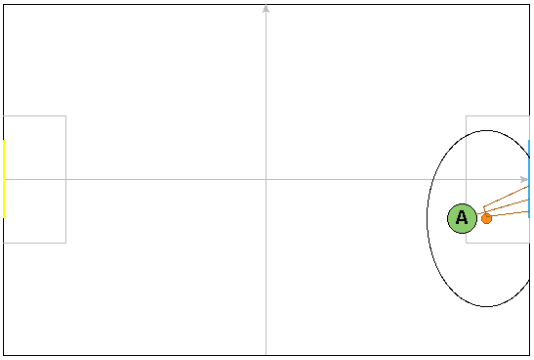
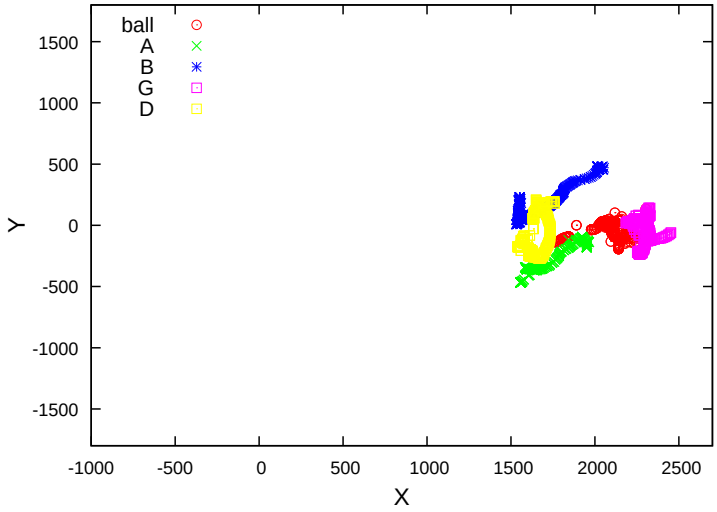


Figure 6.21: Trial example: second case (sequence 3 and 4). Snapshot, world model and segmented image.



(a)



(b)

Figure 6.22: Trial example: third case. (a) Retrieved case, (b) robots' paths.

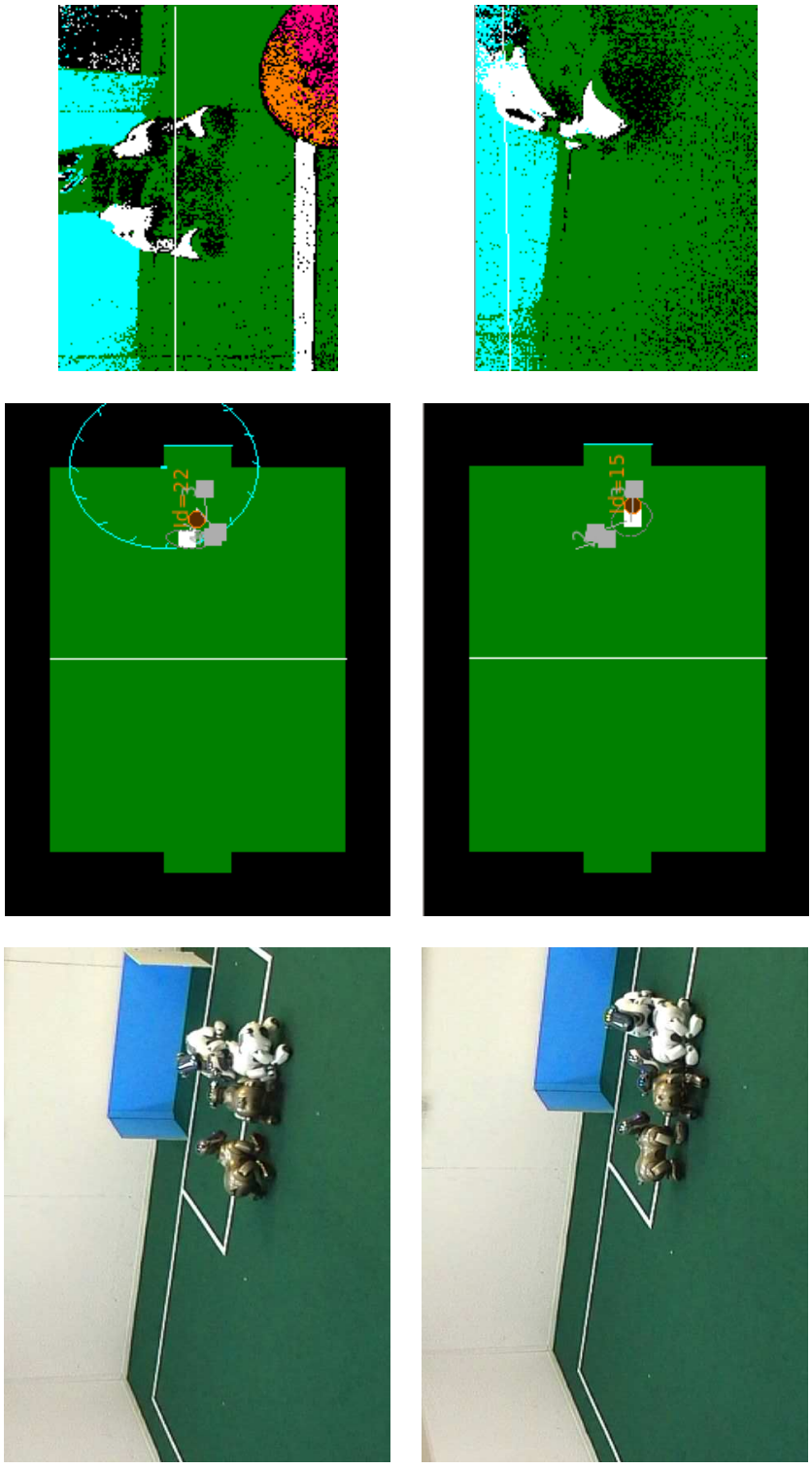
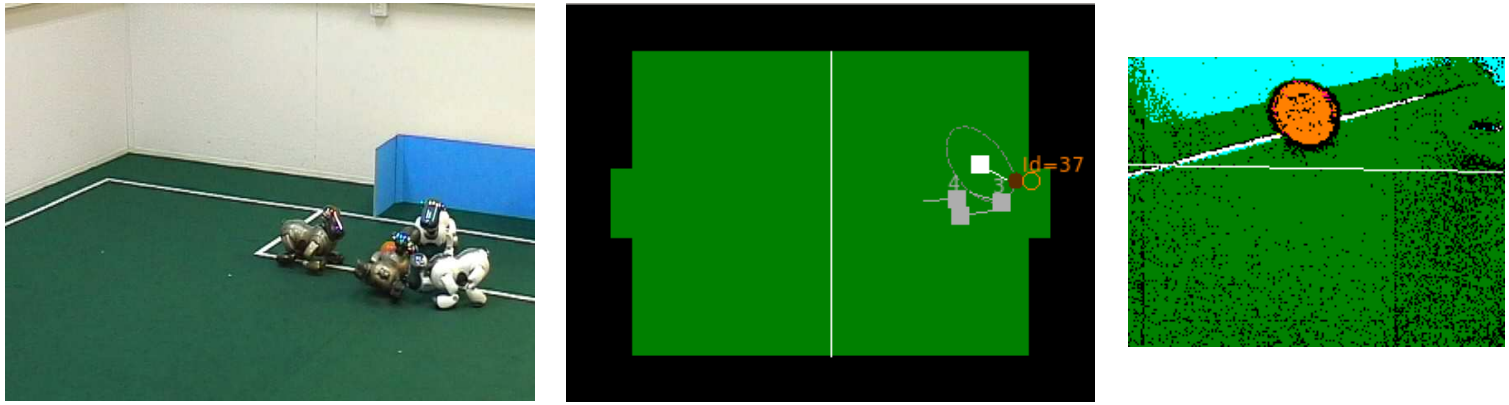


Figure 6.23: Trial example: third case (sequence 1 and 2). Snapshot, world model and segmented image.

(3)



(4)

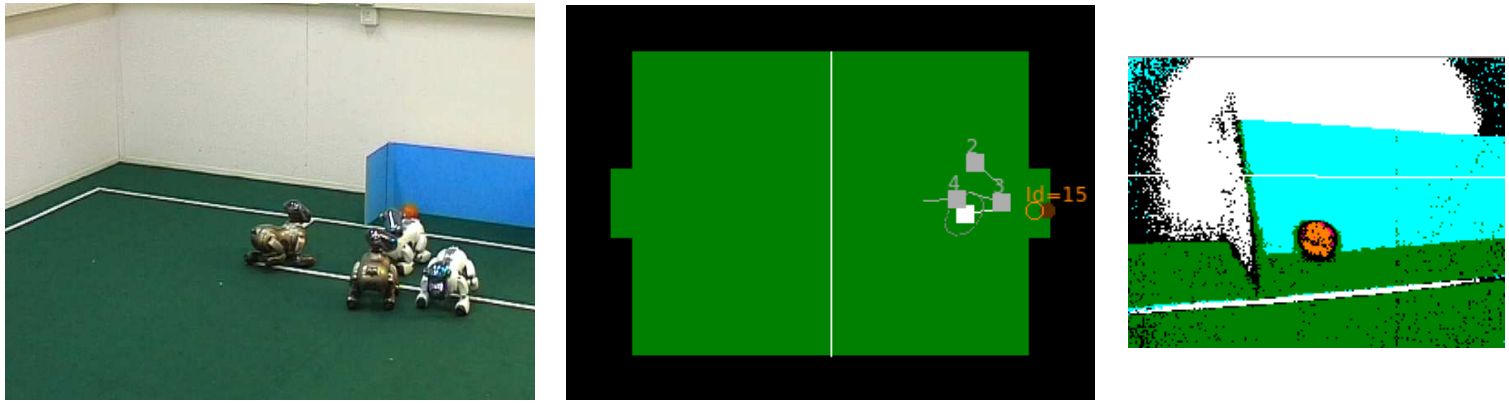


Figure 6.24: Trial example: third case (sequence 3 and 4). Snapshot, world model and segmented image.

6.6 Discussion and Future Work

In general, analyzing both results obtained in simulation and the ones obtained with the real robots, we can confirm that the Case-Based Reasoning approach indeed improves upon the region-based approach, not only on achieving a higher percentage of *to goal* balls, but also, achieving a lower percentage of *out* balls. More precisely, the third scenario results with the real robots confirms the results obtained in simulation. In the fourth scenario, once again the average of balls *out* is higher for the reactive approach, which confirms that the defender had more chances to take the ball.

However, in the last scenario, the reactive approach achieved a higher percentage of balls *to goal* compared to the CBR approach. We must point out that comparing the ideal executions of both approaches (Figures 6.14 and 6.15) we can easily observe that the reactive approach is more efficient on faster moving the ball towards the attacking goal, i.e. with two kicks the robots can score. On the contrary, the attackers with CBR approach need at least three kicks to complete the same task. Hence, the chances for the goalie to move towards a better position to block the ball also increase, as confirmed in the percentage of *blocked* balls by the goalie (60% for the CBR approach vs. 33% for the reactive approach). These results also support that at least the CBR approach had more chances to get closer to the attacking goal, i.e. succeeded on avoiding the defender in the first step (Figure 6.14b), while with the reactive approach, the attacker's first action was blocked most of the times (Figure 6.15a). Otherwise, as the ideal sequences shows, the attackers would have had even more opportunities to try scoring, considerably increasing the percentage of *to goal* balls.

We must analyze a last issue. Comparing the results obtained for the fourth scenario in simulation and with the real robots, we can see that in simulation, the CBR approach outperformed the reactive approach. Hence, no matter that the number of kicks is higher, the CBR can still improve the reactive approach. However, we must always have in mind that the high uncertainty in the perception for the robots is not present in the simulated environment. If we take a look back to Figure 6.14c, we can observe that after reusing the first case, the ball stops near the penalty area. Although the goalie is not allowed to leave its action region (the penalty area), due to the high uncertainty in the robot's world model, the goalie may believe that the ball is within its action region, and therefore, try to grab it. In the simulated environment this situation never takes place, unless the ball is actually within the penalty area. Hence, while in the real world the trial could end with the goalie clearing the ball, i.e. *block* ball in the statistics, in simulation the attacker would probably be able to score, i.e. a *to goal* ball.

This fourth scenario situation, where the attackers are in front of the opponent's goal, verifies that in this kind of domains (high uncertainty, highly dynamic and real time response requirements) to solve critical situations it is useful to have a reactive strategy rather than a deliberative strategy. When the players are in front of the goal, there is no need of reasoning about interesting plays. Instead, being focused on trying to score as fast as possible is the best strategy. We must also remember that the defenders are forbidden to enter the penalty area, and thus, the opponent team has less opportunities to take the ball. The goalie is the last obstacle to achieve the goal. Thus, acting fast

is crucial. However, a deliberative strategy outperforms a reactive one for the remaining situations, i.e. in the middle, back, sides and corners of the field where the opponents have more chances of stealing the ball.

Regarding the defender's performance, we have confirmed that using the CBR approach is a better strategy to avoid the defender stealing the ball because of the explicit passes between teammates. The reactive strategy almost doubles the chances for the defender to steal the ball compared to the CBR approach.

In conclusion, we believe we have indeed verified all three hypotheses presented in the beginning of the chapter. First, both experiments in simulation and with the real robots confirm that applying Case-Based Reasoning techniques is a feasible alternative to procedural programming; second, the real robots experiments reinforce the ability of the CBR approach to handle uncertainty; and third, again, both experiments encourage the advantages of performing a cooperative strategy for joint tasks within dynamic adversarial domains, as the robot soccer presented in this work.

As future work, we are interested in studying in depth a combined strategy integrating both approaches, i.e. a pure deliberative strategy with a reactive one. We believe that because of the domain features, the reactive component must always be part of the overall strategy to solve critical situations. Hence, by combining both approaches we can benefit from their advantages.

Chapter 7

Conclusions and Future Work

In this chapter we review the contributions presented in this dissertation. We also summarize future research lines to improve the presented approach.

7.1 Summary of the Contributions

Designing the decision-making engine of a team of robots is a challenging task, not only due to the complexity of the environment where the robots usually perform their task, which include uncertainty, dynamism and imprecision, but also because the coordination of a team must be included in this design. The robots must be aware of other robots' actions to cooperate and to successfully achieve their common goal. Besides, decisions must be made in real-time and with limited computational resources. In this thesis we have proposed a Case-Based Reasoning system for action selection in a team of robots within the robot soccer domain.

We next go through the contributions presented in this dissertation and briefly summarize the conclusions drawn from each chapter:

Case definition and Case Base description

A case represents a snapshot of a game, i.e. the description of the environment, the actions the robots should perform, i.e. the solution description, and the case scope, i.e. general domain knowledge. An initial set of cases has been manually created. When the system loads the case base, a larger set of cases is automatically derived through spatial transformations. The complete case base is divided in two sets to reduce the search space during retrieval.

Assessing case similarity

When comparing the current problem to solve with the cases in the case base, we first compute their similarity based on what we call the *non-controllable* features. The values of these features cannot be directly modified by the system. Different similarity functions have been proposed depending on the features domains. The overall similarity results from the aggregation of the individual similarities. Next, we compute the cost of adapting the current problem to the case using the *controllable* features.

The idea underlying this measure is that the robots of the team (not the opponents of course) can move to better positions in order to increase the similarity between the state of the world and the evaluated case. The last measures to compute involve the positions of the opponents. The aim is to evaluate whether the case is applicable or not even though the problem to solve and the case have been considered similar enough. Thus, a case is applicable if the trajectories the ball follows when applying the actions indicated in the solution of the case are free of opponents. Finally, the opponent similarity measure reinforces the similarity degree. All this process is conducted through a filtering mechanism. Hence, whenever a case fails in any of the filtering stages, it is immediately withdrawn and the procedure evaluates the next case in the case base, thus reducing the time invested in the search.

Filtering mechanism

After the filtering process takes place, a set of candidate cases are obtained. A sorting algorithm ranks the candidates based on several criteria. In this work we have presented five criteria and three different sorting algorithms. We have performed empirical evaluation in simulation to verify that the retrieval process is suitable for the work presented here, and also to determine the most efficient sorting algorithm among the three proposed.

Multi-robot architecture, coordination and case reuse

After a case is retrieved, the next step is to reuse it. Hence, we have introduced a multi-robot architecture and a coordination mechanism to execute the actions indicated by the retrieved case in a cooperative way. The multi-robot system is composed of two types of robots: the *retrievers* and the *executors*. As their names indicate, the retrievers are capable of retrieving cases, i.e. they incorporate a CBR module in their internal architecture. The executors instead, wait for the retrievers to indicate the case to reuse. Only one case can be executed at a time. Therefore, a coordinator is selected to retrieve the next case to reuse.

Once the coordinator informs the rest of the team (retrievers and executors) about the retrieved case, they all first move towards their adapted positions to start the execution of the case. During the case reuse, any of the robots may abort the execution if, based on its beliefs, it finds out that the case is not applicable anymore. When the case reuse ends, the process starts again, selecting a coordinator, retrieving a case, reusing it, and so on.

Supervised learning for acquiring the scope of cases

In this thesis we have also included a first attempt to automate the adaptation and acquisition of the case-based reasoning system's knowledge with respect to the scope of a case, i.e. the case coverage. To this end, a two-steps algorithm has been presented. The first step grows the initial scope of a case until approximating the expected one (based on the expert's knowledge), while the second one, makes it converge. We have presented three different function policies to this end, and evaluated them

through simulation and with a real robot. We have also included a simple mechanism to create new cases when no case is found.

Empirical evaluation of the approach

Finally, we have evaluated the retrieval and the reuse step both in simulation and with real robots comparing the proposed approach (CBR approach) with the region-based approach (reactive approach) presented by the Carnegie Mellon team, CMDash. The scenarios consisted of two vs. two games, where the attackers played using either the CBR or the reactive approach. We have implemented the behaviors for the defender and the goalie, which were used against both types of attackers.

The results showed that the CBR approach not only outperformed the reactive approach in general, scoring more goals, throwing fewer balls out of the field and decreasing the defender's ball possession, but also encouraged the team to behave in a cooperative way, having passes between the attackers when possible. The experiments also demonstrated that in this kind of domains (high uncertainty, highly dynamic and real time response requirements) to solve critical situations it is sometimes useful to have a reactive strategy rather than a deliberative strategy where acting fast is crucial. Thus, we believe that the combination of both strategies is essential to obtain an effective robot behavior.

7.2 Future Directions

The presented work introduces a complete framework for the action selection problem in a team of robots, starting from the decision-making until the coordinated execution of the selected actions. As concluded in the previous section, and observing the successful results obtained through the experimentation, we can claim that the goals of the thesis have been achieved, while the proposed hypotheses have also been verified. However, and as expected, improvements and open issues are still pending. We next review the open challenges proposed through this dissertation.

Retrieval Step

The cost function should model not only the distance the robots have to travel to reach their adapted positions, but also the possible obstacles the robots may encounter in their paths. However, it must be taken into account that the cost may vary from decision time, i.e. during retrieval, to execution time, i.e. during reuse, since the obstacles are other robots that are constantly moving. Other parameters such as robot's orientation or velocities can also be considered when defining the cost function.

In order to increase flexibility in the applicability measure, we could make use of the fuzzy representation of the free path and the opponent similarity functions.

The evaluation of the candidate cases could be extended, including not only the problem description, but also an analysis of the solution description. This way we could discriminate between similar cases whose problem descriptions are equal, but with different actions. To this end, the

outcome of the reused case should be included in the case description so that different solutions can be compared.

It would be also interesting to include for each case description, a list of most likely cases to be retrieved in the next cycle of the CBR. This way patterns of case executions can be obtained for further analysis, such as the evaluation of the team behavior or prediction of future states.

Reuse Step

The selection of the case to reuse could be improved considering the proposals of each retriever robot. Thus, a negotiation mechanism should be introduced so the robots could bargain for selecting the most suitable case to reuse among the retrieved ones.

Some of the available negotiation mechanisms we can find in the literature, among others, and that should be studied to determine their viability, are:

- *voting mechanisms*, where the retrievers would vote for a case or a set of candidate cases, and the most voted case would be selected;
- *bidding mechanisms*, where each retriever indicates along with the proposed case, a bid representing the confidence on how suitable the retrieved case is given its current internal beliefs (we must recall that uncertainty in the robot's beliefs is always present in different degrees); or
- *argumentation mechanisms*, which are far more complex, since the robots (agents) must exchange arguments for or against the proposals submitted for discussion. However, this approach might not be feasible in domains where acting fast is crucial.

Revise Step

Including this step in the current approach is fundamental if we expect the system to automatically improve its performance as well as to adapt and to learn new situations encountered through time. However, because of the nature of the domain, and more precisely, the continuous property of the domain, the design of this latter step is very challenging. In order to revise the reused case, at least the following questions should be analyzed:

- when to consider that a potential case has *started*, i.e. identify the initial situation;
- when does it *finish*, i.e. which are the actions to store as part of the solution of a case; and,
- how to evaluate the *outcome* of the performed actions to determine if the case is useful or not for achieving the robot's goals (i.e. positive or negative feedback).

Some previous works have been already presented in the past addressing these issues [51, 37, 27], and should be studied in detail in order to adapt their ideas to the robot soccer domain.

As discussed in the experimentation chapter, we should also study in more detail the combination of the two types of strategies, deliberative and reactive, in order to benefit from the advantages of both. Having a deliberative strategy is fundamental for making decisions from a high level point of view, considering the complete state of the world, as well as past history or future predictions of the state evolution. However, a reactive strategy is also essential when fast response is required to solve critical situations.

Finally, the parameters used in the approach, such as thresholds, could be modified on-line in order to alter the behavior of the retrieval and the reuse step. Thus, different cases would be retrieved modifying the team strategy. Moreover, these parameters could vary based on the time and score of the game. For instance, consider a situation where few minutes remain for ending the game and the team is losing. It would be then desirable that the team switched to an offensive strategy. We could achieve this through two means: we could design a specific set of cases for each type of strategy, and let the retrieval step to select the most suitable type of case as presented in this dissertation; or we could vary the parameters of the approach such that cases that would be initially withdrawn, would now be candidates because the measures' thresholds have been altered. In the case of the free path function for instance, a case is discarded if an opponent is within the ball's trajectory(whose width is parametrized by a given threshold). If we alter the value of this latter parameter, we could reduce the width of the trajectory. Hence, the opponent that initially occupied part of the path, would not be considered within it anymore, and therefore, the case would be a next candidate. Similarly, the ball's scope and the opponents' scope can be also modified on-line, and as a consequence, the retrieval process of the CBR system is modified as well, which in turn, alters the robots' behavior.

Bibliography

[1] A brief history of robocup, http://www.robocup.org/overview/23.html.

[2] A. Aamodt and E. Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39 – 59, 1994.

[3] G. Adorni, S. Cagnoni, M. Mordonini, and M. Piaggio. Team/goal-keeper coordination in the robocup mid-size league. In P. Stone, T. R. Balch, and G. K. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Computer Science*, pages 279–284. Springer, 2001.

[4] M. Ahmadi, A. K. Lamjiri, M. M. Nevisi, J. Habibi, and K. Badie. Using a two-layered case-based reasoning for prediction in soccer coach. In H. R. Arabnia and E. B. Kozerenko, editors, *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications*, pages 181–185. CSREA Press, 2003.

[5] M. Ahmadi and P. Stone. Instance-based action models for fast action planning. In U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, editors, *RoboCup 2007: Robot Soccer World Cup XI*. Springer Verlag, Berlin, 2008. To appear.

[6] R. C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[7] I. Asimov. *I, robot*. Gnome Press, USA, 1950.

[8] R. Berger and G. Lämmel. Exploiting past experience - case-based decision support for soccer agents. In *KI 2007: Advances in Artificial Intelligence*, volume 4667 of *Lecture Notes in Computer Science*, pages 440–443. Springer, 2007.

[9] C. Bustamante, L. Garrido, and R. Soto. Fuzzy naive bayesian classification in robosoccer 3d: A hybrid approach to decision making. In *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, pages 507–515. Springer, 2007.

[10] K.-Y. Chen and A. Liu. A design method for incorporating multidisciplinary requirements for developing a robot soccer player. In *Proceedings of the Fourth IEEE International Symposium on Multimedia Software Engineering*, pages 25–32. IEEE Computer Society, 2002.

[11] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

[12] R. T. Committee. *Sony Four Legged Robot Football League Rule Book*, December 2004.

[13] M. H. DeGroot and M. J. Schervish. *Probability and Statistics*. Addison-Wesley, 2001. 3rd. Edition.

[14] A. Drogoul and A. Collinot. Applying an agent-oriented methodology to the design of artificial organizations: A case study in robotic soccer. *Autonomous Agents and Multi-Agent Systems*, 1(1):113–129, 1998.

[15] Y. Duan, Q. Liu, and X. Xu. Application of reinforcement learning in robot soccer. *Engineering Applications of Artificial Intelligence*, 20(7):936–950, 2007.

[16] S. Dutta, B. Wierenga, and A. Dalebout. Case-based reasoning systems: From automation to decision-aiding and stimulation. *IEEE Transactions on Knowledge and Data Engineering*, 9(6):911–922, 1997.

[17] R. Fikes and N. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[18] G. Fraser and F. Wotawa. Cooperative planning and plan execution in partially observable dynamic domains. In *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Computer Science*, pages 524–531. Springer, 2005.

[19] B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[20] K. Haigh and J. Shewchuk. Geometric similarity metrics for case-based reasoning. In *Case-Based Reasoning: Working Notes from the AAAI-94 Workshop*, pages 182–187. AAAI Press, 1994.

[21] K. Haigh and M. Veloso. Route planning by analogy. In M. Veloso and A. Aamodt, editors, *Case-Based Reasoning Research and Development*, volume 1010 of *Lecture Notes in Computer Science*, pages 169–180. Springer-Verlag, 1995.

[22] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[23] Z. Huang, Y. Yang, and X. Chen. An approach to plan recognition and retrieval for multi-agent systems. In *Workshop on Adaptability in Multi-Agent Systems (AORC 2003)*, 2003.

[24] K. G. Jolly, K. P. Ravindran, R. Vijayakumar, and R. S. Kumar. Intelligent decision making in multi-agent robot soccer system through compounded artificial neural networks. *Robotics and Autonomous Systems*, 55(7):589–596, 2007.

[25] A. Karol, B. Nebel, C. Stanton, and M.-A. Williams. Case based game play in the robocup four-legged league part i the theoretical model. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 739–747. Springer, 2004.

[26] H.-S. Kim, H.-S. Shim, M.-J. Jung, and J.-H. Kim. Action selection mechanism for soccer robot. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, page 390. IEEE Computer Society, 1997.

[27] Z. Kira and R. C. Arkin. Forgetting bad behavior: Memory management for case-based navigation. In *International Conference on Intelligent Robots and Systems*, volume 4, pages 3145–3152, 2004.

[28] A. Kleiner, M. Dietl, and B. Nebel. Towards a life-long learning soccer agent. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Computer Science*, pages 126–134. Springer, 2003.

[29] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1973.

[30] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA, USA, 1993.

[31] S. Konur, A. Ferrein, and G. Lakemeyer. Learning decision trees for action selection in soccer agents. In *ECAI-04 Workshop on Agents in dynamic and real-time environments*, 2004.

[32] M. Kruusmaa. Global navigation in dynamic environments using case-based reasoning. *Autonomous Robots*, 14(1):71–91, 2003.

[33] K. Lam, B. Esfandiari, and D. Tudino. A scene-based imitation framework for robocup clients. In *Workshop on Modeling Others from Observations (AAAI 2006)*, 2006.

[34] A. D. Lattner, A. Miene, U. Visser, and O. Herzog. Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 118–129. Springer, 2006.

[35] J. Lee, D. Ji, W. Lee, G. Kang, and M. G. Joo. A tactics for robot soccer with fuzzy logic mediator. In *Computational Intelligence and Security*, volume 3801 of *Lecture Notes in Computer Science*, pages 127–132. Springer, 2005.

[36] M. Lenz and H.-D. Burkhard. Case retrieval nets: Basic ideas and extensions. In G. Gorz and S. Holldobler, editors, *KI-96: Advances in Artificial Intelligence*, volume 1137 of *Lecture Notes in Computer Science*, pages 227–239. Springer Verlag, 1996.

[37] M. Likhachev and R. C. Arkin. Spatio-temporal case-based reasoning for behavioral selection. In *International Conference on Robotics and Automation*, volume 2, pages 1627–1634. IEEE, 2001.

[38] Y.-S. Lin, A. Liu, and K.-Y. Chen. A hybrid architecture of case-based reasoning and fuzzy behavioral control applied to robot soccer. In *Workshop on Artificial Intelligence, 2002 International Computer Symposium (ICS2002)*, 2002.

[39] R. López de Màntaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. L. Maher, M. T. Cox, K. Forbus, M. Keane, A. Aamodt, and I. Watson. Retrieval, reuse, revision and retention in case-based reasoning. *Knowledge Engineering Review*, 20(3):215–240, 2005.

[40] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler. Co-evolving soccer softbot team coordination with genetic programming. In *RoboCup-97: Robot Soccer World Cup I*, volume 1395 of *Lecture Notes in Computer Science*, pages 398–411. Springer, 1998.

[41] C. Marling, M. Tomko, M. Gillen, D. Alexander, and D. Chelberg. Case-based reasoning for planning and world modeling in the robocup small size league. In *IJCAI-03 Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating*, 2003.

[42] A. Micarelli, A. Neri, S. Panzieri, and G. Sansonetti. A case-based approach to indoor navigation using sonar maps. In *International IFAC Symposium On Robot Control SYROCO*, 2000.

[43] A. Miene, U. Visser, and O. Herzog. Recognition and prediction of motion situations based on a qualitative motion description. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2004.

[44] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[45] T. Nakashima, M. Takatani, M. Udo, H. Ishibuchi, and M. Nii. Performance evaluation of an evolutionary method for robocup soccer strategies. In *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 616–623. Springer, 2006.

[46] O. Obst and J. Boedecker. Flexible coordination of multiagent team behavior using htn planning. In *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 521–528. Springer, 2006.

[47] J.-H. Park, D. Stonier, J.-H. Kim, B.-H. Ahn, and M.-G. Jeon. Recombinant rule selection in evolutionary algorithm for fuzzy path planner of robot soccer. In C. Freksa, M. Kohlhase, and K. Schill, editors, *KI 2006: Advances in Artificial Intelligence*, volume 4314 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2006.

[48] K.-H. Park, Y.-J. Kim, and J.-H. Kim. Modular q-learning based multi-agent cooperation for robot soccer. *Robotics and Autonomous Systems*, 35(2):109–122, 2001.

[49] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[50] A. Ram. Indexing elaboration and refinement: Incremental learning of explanatory cases. *Machine Learning*, 10(3):201–248, 1993.

[51] A. Ram and J. C. Santamaría. Continuous case-based reasoning. *Artificial Intelligence*, 90(1-2):25–77, 1997.

[52] M. A. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, and R. Ehrmann. Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer. In P. Stone, T. R. Balch, and G. K. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Computer Science*, pages 367–372. Springer, 2001.

[53] T. Rofer, T. Laue, and M. W. et al. Germanteam robocup 2005. Technical report, Universitat Bremen, Humboldt-Universitat zu Berlin, Technische Universitat Darmstadt, Dortmund University, 2005.

[54] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986.

[55] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[56] R. C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, New York, NY, USA, 1983.

[57] H. L. Sng, G. S. Gupta, and C. H. Messom. Strategy for collaboration in robot soccer. In *Proceedings of the The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA)*, pages 347–354. IEEE Computer Society, 2002.

[58] T. Steffens. Adapting similarity-measures to agenttypes in opponent-modelling. In *Workshop on Modeling Other Agents from Observations at AAMAS 2004*, 2004.

[59] R. Stepp and R. S. Michalski. Conceptual clustering: Inventing goal-oriented classifications of structured objects. *Machine Learning: An Artificial Intelligence Approach*, 2:471–498, 1986.

[60] P. Stone, M. Sridharan, D. Stronger, G. Kuhlmann, N. Kohl, P. Fidelman, and N. K. Jong. From pixels to multi-robot decision-making: A study in uncertainty. *Robotics and Autonomous Systems*, 54(11):933–943, 2006.

[61] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[62] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[63] A. Tews and G. Wyeth. Multi-robot coordination in the robot soccer environment. In *Proceedings of the Australian Conference on Robotics and Automation*, pages 90–95, 1999.

[64] C. Urdiales, E. J. Prez, J. Vzquez-Salceda, M. Snchez-Marr, and F. S. Herndez. A purely reactive navigation scheme for dynamic environments using case-based reasoning. *Autonomous Robots*, 21(1):65–78, 2006.

[65] D. Vail and M. Veloso. Dynamic multi-robot coordination. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, volume 2, pages 87–100. Kluwer, 2003.

[66] B. van der Vecht and P. U. Lima. Formulation and implementation of relational behaviours for multi-robot cooperative systems. In *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Computer Science*, pages 516–523. Springer, 2005.

[67] M. Veloso, P. E. Rybski, S. Chernova, C. McMillen, J. Fasola, F. vonHundelshausen, D. Vail, A. Trevor, S. Hauert, and R. Ros. Cmdash05: Team report. Technical report, Carnegie Mellon University, 2005.

[68] M. M. Veloso. *Planning and Learning by Analogical Reasoning*, volume 886 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[69] U. Visser and H.-D. Burkhard. RoboCup: 10 Years of Achievments and Future Challenges. *AI Magazine*, 28(2):115–132, Summer 2007.

[70] J. Wendler and J. Bach. Recognizing and predicting agent behavior with case based reasoning. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 729–738. Springer, 2004.

[71] J. Wendler, S. Brggert, H.-D. Burkhard, and H. Myritz. Fault-tolerant self localization by case-based reasoning. In P. Stone, T. R. Balch, and G. K. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Computer Science*, pages 259–268. Springer, 2001.

[72] J. Wendler and M. Lenz. CBR for Dynamic Situation Assessment in an Agent-Oriented Setting. In *AAAI-98 Workshop on CaseBased Reasoning Integrations*, 1998.

[73] C.-J. Wu and T.-L. Lee. A fuzzy mechanism for action selection of soccer robots. *Journal of Intelligent Robotics Systems*, 39(1):57–70, 2004.

[74] K. Yoshimura, N. Barnes, R. Rnnquist, and L. Sonenberg. Towards real-time strategic teamwork: A robocup case study. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Computer Science*, pages 342–350. Springer, 2003.