Team play among soccer robots

Master's thesis Artificial Intelligence
University of Amsterdam

Matthijs Spaan
April 5, 2002

Abstract

The main question discussed in this thesis is how to create robotic soccer players which collaborate and play as a team. First we present an approach for coordinating a team of soccer playing robots which is based on the idea of dynamically distributing roles among the team members and adds the notion of a global team strategy (attack, defend and intercept) to existing approaches. Utility functions are used to estimate how well suited a robot is for a certain role, taking into account the time the robot expects to need to reach the ball and the robot's position in the field.

Second contribution of this thesis is a way to use the assigned role in a robot's selection of the next action. We model the planning of future actions using Markov decision processes and the role of the robots determines its action space and influences its reward function. The latter uses four soccer heuristics: whether the ball is in one of the goals, ball possession, position based role evaluation and strategic positioning.

These approaches have been developed for use in the Dutch RoboSoccer team Clockwork Orange, which participates in the RoboCup middle-size league. Empirical results from the RoboCup 2001 tournament are presented demonstrating the value of actively adding a notion of team play.

Acknowledgements

First of all I would like to thank Frans Groen for his guidance and support, both during the stage of developing ideas and software as well as during the writing of this thesis. Nikos Vlassis has been more than helpful during the second half of the graduation project.

Graduating on a topic in a team of soccer robots is not possible without the support of the team behind the team. I would like to thank all the people who formed Clockwork Orange on the road to RoboCup 2001, in particular the Amsterdam crew consisting of Bas Terwijn, Raymond Donkervoort and Jeroen Roodhart, for the many hours spent in good spirit and collaboration.

The years of education have been enlightened by the company of the Golden Boys, and during the graduation period specifically by Wouter Caarls, Erik de Ruiter and Barry Koopen who have shared their opinions and insights on numerous topics.

Furthermore I would like to thank my neighbor Emrys van der Loos for having to bear with my daily struggles concerning robots and writing.

I am grateful to my parents and sister for their support and encouragement throughout these years.

Preface

Chapters one and two are a result of a collaboration between Raymond Donkervoort, Bas Doodeman and myself.

Voetbal is simpel. Het is echter moeilijk om simpel te voetballen.

— toegeschreven aan Johan Cruijff

Soccer is simple. However, it is hard to play simple soccer.

— attributed to Johan Cruijff

Contents

Chapter 1

Introduction

Imagine being a robot. Not a fancy bipedal human-like walking robot, but a simple two wheeled one with the appearance of a trash can. Imagine your only view on the world is from a camera mounted on top of your body that is only able to look straight ahead. You can't do anything but rotating your left or right wheel and folding out your aluminum leg. Finally, you don't have the luxury to lean back and relax while a human operator takes some decision about what to do next, no, you have to figure it out all by yourself. Your masters expect you to play soccer with your robot friends and beat the other team. Luckily you can speak with your friends to tell each other where the ball is and discuss strategy and tactics. Sometimes however you're not sure where you are on the field which complicates things considerably.

How would you go about? This little story captures the essence of the problems one is faced when designing an autonomous robotic soccer player which has to coordinate its actions with its teammates. We will start by generalizing the concept of a robot to the one of an agent, consider the case when multiple agents have to work together in an environment followed by a description of the domain used for this thesis. We conclude the chapter with an outlook on the rest of the thesis.

1.1 Agents

We define the term agent as just about anything that can perceive its environment through sensors and can act through actuators. Examples of agents are humans, animals but also robots and certain communicating software programs ("softbots") . The agents that will be described in this thesis are robots, autonomous agents acting in the real world. The sensors a robot uses to perceive its environment can be devices such as cameras, sonars or laser range finders. The actuators can be all sorts of things like motor driven or pneumatic devices. For an agent to be autonomous it also has to reason about its environment before acting upon it. This reasoning typically happens in the software part of a robot. Figure 1.1 shows a diagram of an autonomous robot and its environment.

There are numerous applications for autonomous robots, both industrial and domestic. Industrial robots can perform tasks like the assembly of indus-

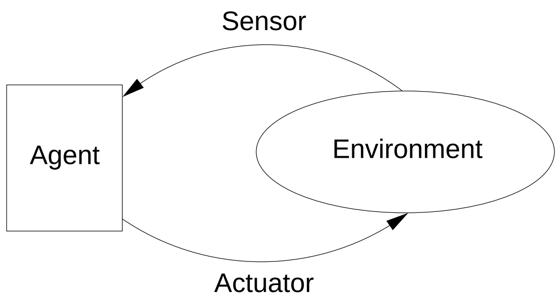


Figure 1.1: An agent interacting with its environment.

trial products, intelligent transport systems, bringing medicine to patients in hospitals and guarding buildings. Domestic robots can make life easier by performing tasks such as mowing the lawn, cleaning the house or assisting the disabled. Robots can also be used in situations which are too dangerous for humans: rescuing people from burning buildings, finding people under debris after an earthquake or sweeping mine fields.

1.2 Multi-agent systems

Multi-agent systems are systems in which multiple agents act in the same environment. An interesting situation occurs when there is no central control, all agents get their information from the environment from their own sensors or through communication with other agents, and act upon their perception of environment with their own individual actuators. The behavior of the entire system depends on the behavior of the separate agents. Communication between the agents plays an important role in multi-agent systems, both in sharing information from the individual sensors and in communicating behavior in intended actions. There are several different reasons to prefer a multi-agent system approach over the single-agent approach:

- Multi-agent systems are easier to extend by adding a new agent, making them much more scalable than single-agent systems.

- Multi-agent systems can be very robust, if one of the agents breaks down the others will continue their job and can possibly still fulfill their task.

- Multi-agent systems can, by all having their own sensors, form a more complete and accurate model of their environment than single-agent systems which often only have one perspective at a certain moment of time. However this means sensor fusion is needed to form this model since all the data has to be combined in the right way.

- Some applications consist of parts that keep information hidden from the other parts and have different goals. This problem would be impossible to handle with a single-agent system. Every part of the application has to be handled by a separate agent who can communicate with each other to solve the problem. An example is an e-commerce setting in which agents have to bargain for selling or buying certain services.

The task of the developer of a multi-agent system is to decide in which way to organize the system. First he should try to decompose the problem into smaller subproblems that could be solved by separate agents. Then he will have to decide on the representation of the domain, the architecture to use and how to program the interaction between the agent and the behavior of the individual agents. The behavior of an individual agent is determined by a so called 'decision policy' which uses the inputs from the sensors to determine the action of the agent. The architecture of the multi-agent system determines how the different agents cooperate. There are different possible architectures varying in complexity and suitability for certain applications.

Centralized single agent control A central controller collects the information from the sensors of the individual agents and from these inputs selects the actions for each agent. The great advantage of this approach is that the best cooperative policies for the agents will be found. However the system isn't very robust, if the central control breaks down the entire system will stop working. Also the action space in the central controller will be huge, making the system very complex.

Policy sharing To reduce the complexity of this system, policy sharing can be used. Each agent now uses the same policy making the agents homogeneous. A great disadvantage of this approach is that it leaves no room for specialization of certain agents for specific problems.

Task schedules By computing in advance a schedule of all the tasks the agents have to perform we can build a system that has very low complexity and is very fast. However when the environment changes the schedule has to be computed all over again. This architecture is therefore not suited for dynamic environments.

Local agents In this system each of the agents will have to find its own policy with limited interaction only, thus reducing the complexity of the system dramatically. But because of the very limited range of view each agent has it will be very hard to find a good cooperative behavior. If there are many dependencies in solving parts of the problem this will create difficulties.

Hierarchical systems Instead of completely dividing the problem in local parts, we could also use a more hybrid approach. In hierarchical systems some independent parts of the problem are handled by individual agent whereas parts with dependencies will be handled by multiple cooperative agents. However this approach requires some sort of central control which could break down, making the system less robust.

Shared global world models The information the agents receive about their environment is shared with the other agents to form a combined global world model, upon which each agent determines its actions.

Before deciding on which architecture to use in our multi-agent system we have to describe our domain more carefully. However, from the listed architectures a central control type of architecture can be labeled as unsuitable beforehand. Central control is not suited for robust multi-agent systems as it has a single point of failure. Robots in general and soccer robots in particular are prone to hardware failures so the system should degrade gracefully. Another disadvantage of central control is the fact that it does not scale well as the central unit has to cope with a growing number of clients.

In order to be able to make a well-founded decision on a multi-agent architecture we need to take a look at the possible domains in which an agent can be used. As mentioned above, autonomous agents can be used in many different applications. To make it easier to determine which architecture we should use for any specific problem, we should have a way of describing the domains. To do this we can use a number of features:

static or dynamic Does the environment in which the agents have to act change while the agents are in it and do the acts of the robots influence the environment?

communication Is communication between the agents possible or even desirable? In some applications communication between agents is necessary, but in others it is better not to use it since it is liable to interference.

discrete or continuous Is it possible to represent the states of the agent in a discrete manner or is the problem such that it has to be dealt with in a continuous way?

cooperative or competitive Do all the agents in the system have the same goal or do some have different or even conflicting goals?

completely or partly observable Is the environment completely or only partly visible to the system?

dynamic or static agents In some applications the agents have a fixed place in the environment while in others they can move, introducing the risk of colliding with objects or even with each other.

dependent or independent Is it possible for the robots to fulfill their separate tasks individually or do they have to work together to complete a certain part of the problem?

homogeneous or heterogeneous Are all the robots in the system the same or do they have a different design?

The robotic soccer domain this thesis deals with can be characterized in the features described above: both environment as well as the agents are dynamic, communication is an option and variables are likely to be continuous. The multi-agent system is both competitive as well cooperative as two teams play against each other while the robots in the same team share a common goal. The environment is only partly observable for a single robot and the observations contain a certain degree of error. The robots should try to work together, but strictly speaking the domain is independent as one single robot can score goals. The multi-agent system is heterogeneous as there are two different teams. A single team can be both homogeneous or heterogeneous.

Having described the environment using the features mentioned above we can now choose which of the suggested architectures to use. Some architectures will be unsuitable for certain types of problems, while they work perfectly for other types of problems. For instance, in applications in which navigation of the robots plays an important role, which are dynamic and continuous and contain dynamic agents, the global world model architecture would be preferable. Task schedules can be used to divide problems in subproblems and are also useful for many static problems. Policy sharing can only be used in situations where homogeneous agents are adequate. Hierarchical systems are especially useful in situations where agent behavior has to be coordinated.

We have chosen to design our system around a shared global world model, which simplifies team coordination as each robot can reason about a global view of the world. Sharing the local world models enhances the completeness and accuracy of the global world model. Each robot itself can be viewed as a multi-agent system, designed as a hierarchical which enables us to have several autonomous software modules in our architecture, each responsible for its own task. A detailed description of our software architecture is presented in the next chapter. More information about multi-agent systems in general can be found in [15, 18].

1.3 The RoboCup project

In the year 1997 the Robot World Cup Initiative ([32, 43] for more information) was started as an attempt to improve AI and robotics research by providing a standard problem in which a wide range of technologies can be integrated and examined. The standard problem that was chosen is a game of soccer which contains most of the important elements that are present in a real world multi-agent applications. There are different robots that have to work together toward a common goal, the domain is continuous and dynamic, there are opponents whose behavior will not be fully predictable and because of the competitive element of the game it is necessary to act sensible and fast. This together with the fact that the game offers a constricted controllable domain and is entertaining and challenging makes it an ideal test-bed for multi-agent collaborating robotics research. To keep the game as close as possible to the real game of soccer, most rules used in human soccer are also used in robot soccer. To achieve the goal of an autonomous team of soccer playing robots, various technologies have to be incorporated including control theory, distributed systems, computer vision, machine learning, communication, sensor data fusion, self localization and team strategies. In order to do research at as many different levels as possible several different leagues exist.

Simulation League In this league teams of 11 virtual players compete. The players in these teams can be either homogeneous or heterogeneous and have various properties such as dexterity and pace. Since there is only a limited amount of uncertainty in the information the teams have about the game, compared to the other leagues it is relatively easy to generate a complete and accurate world model, although communication is limited. This enables the teams to concentrate on cooperative team behavior and tactics. The University of Amsterdam also participates in this league [6].

Small-Size Robot League The small-size league (F180) is played on a table-tennis table-sized field. Each team consists of five small robots. A camera above the field is used to get a complete view of the game, which is send to the computers of the teams on the side of the field. From this image a world model is constructed using the color coding of the ball and the different robots. Using this world model the actions of the different robots are determined and send to the robots. Since the world model is complete and quite accurate research here focuses on robot coordination, team behavior and real time control. The games in this league are typically very fast and chaotic.

Middle-Size Robot League In the middle-size league (F2000) teams consisting of four robots, sized about 50×50×80 cm, compete on a field of about 10 meters long and 5 meters wide. The main difference with the small-size league is that there is no global vision of the field, all the sensors and actuators for perceiving and acting in the game are on-board. All robots will have to form a model of the world using only their local sensors and the information which they receive from the other robots. Besides individual robot control and generating cooperative team behavior, key research issues here are self localization, computer vision and fusion of the sensor data. This is the league in which the Dutch robot soccer team Clockwork Orange, which will be described in the next chapter, participates. The specific rules for this league will be described in the next section.

Sony Legged Robot League On a field, slightly larger than the small-size league, teams of three Sony AIBO's (the well-known robotic toy dog) compete. These robots walk on four legs, and are thus the first 'step' toward a league of biped humanoid robots. Since every team uses the same robots, the only difference between the teams is in the software.

Humanoid League Starting in the Fukuoka 2002 RoboCup, this league will consists of teams of biped humanoid robots.

Since the first RoboCup Event held at the International Joint Conference on Artificial Intelligence in Nagoya, Japan in 1997, there has been a Robot Soccer World Cup each year: Paris 1998, Stockholm 1999, Melbourne 2000 and Seattle 2001. Also an increasing amount of regional competitions are organized, as there were the European Robot Soccer Championships in Amsterdam 2000, the Japan Open in Fukuoka 2001 (also host of the 2002 World Cup) and the German Open in Paderborn 2001 (in which Clockwork Orange also participated). The number of teams attending in the different leagues of RoboCup has increased dramatically since the first World Cup (from 40 teams in 1997 to about 100 teams from about 20 different countries in 2001). Over the years the games also started getting more attention. The number of spectators has increased from 5000 in Nagayo 1997 to 20,000 in Seattle 2001. Also an increasing amount of media attention, several different large international newspapers and television stations have reported on the RoboCup event, including well known Dutch newspapers De Volkskrant and De Telegraaf. These results show that the initiative has succeeded in its goal of attracting more attention to and improving AI and robotics research by providing an entertaining and challenging application. And maybe this will lead to the robotics researchers ultimate goal of, by the

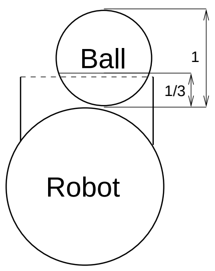


Figure 1.2: The ball handler rule.

year 2050, building a team of robots that can beat the human world champion soccer team.

1.4 The rules of the middle-size league

The league in which Clockwork Orange participates is the middle-size league. Unlike the Sony legged league, the teams in the middle-sized league are free to choose the type of robot, sensors and actuators. However there are some restrictions. The robot may not have a configuration in which its projection on the floor does not fit in a 60×60cm square. A robot should also have a configuration (with all its actuators retracted) in which its projection on the floor fits into a 50×50cm square. The robot may not be any higher than 80cm but should be at least 30cm (so it is large enough to be perceived by other robots). The ball handling mechanism should be build in such a way that it is always possible for an opponent to steal the ball. Therefore a ball handling mechanism may not include the ball for more then 1/3 of the ball's size (see figure 1.2). It also is prohibited to fix the ball to the body by using some kind of ball holding device. These rules also make it more challenging to let the robot turn and dribble with the ball. The robots should all be completely black and are supposed to wear some kind of marker which is either magenta or cyan, depending on the team the robot is in. This makes it possible for the robots and the audience to see which team the robot is in. The robots should also carry numbers making it possible for the referee to tell them apart.

The ball that is used is a standard FIFA size 5 orange ball. The field is green and may vary in size between 8 and 10 meters in length and between 4 and 7 meters in width. The lines are white and closely resemble the lines on a human soccer field. There is a line in the middle of the field, with a dot in the middle, from which the kickoff is taken, with a 1 meter wide circle around it, which must be empty apart from the taker of the kickoff and the ball when starting a game. There are penalty dots on which the ball will be positioned during a penalty shoot-out and there is a goal-area on either side of the field. The field is surrounded by a wall, so the ball can't leave the field. There are two 2-meter wide goals, a blue one and a yellow one, making it possible for the robots to distinguish their own goal from the opponent's goal by color.

The rules of the game are comparable to but not completely the same as those of human soccer. If all robots would stay in line in front of their own goal it would be impossible for the opponent to score a goal. Therefore only one

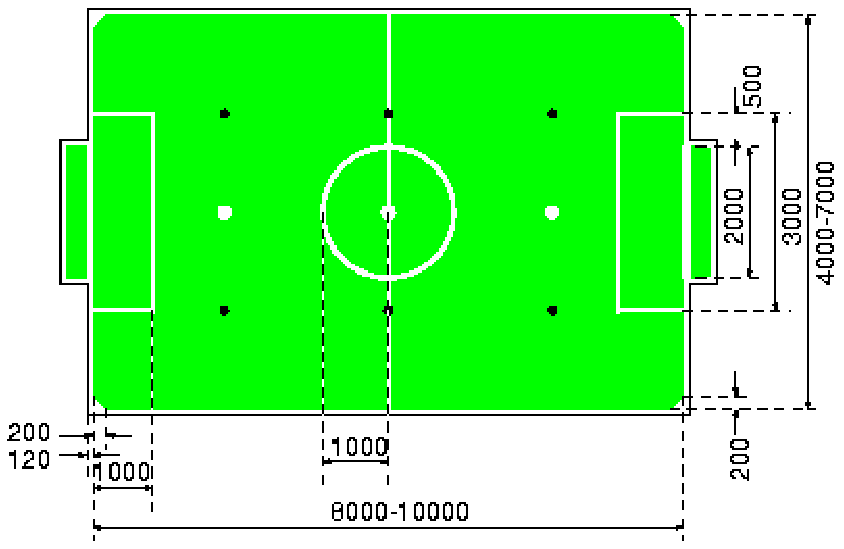


Figure 1.3: The sizes and lines of a robot soccer field.

robot in each team, which must be designated as goalkeeper, may permanently stay in the team's own goal area. Any other robot may not stay in its own goal area for more then 5 seconds. Also only one robot at a time may enter the opponent's goal area and may stay there for no more than 10 seconds and is only allowed there if the ball is also in the goal area. This should prevent obstruction of the opponent's goalkeeper and by doing that scoring goals in a unguarded goal. As in human soccer, a robot will receive a yellow card when charging an opponent. The game is stopped and the opponent gets a free kick. When a robot receives a second yellow card this is considered a blue card and the robot must be removed from the game until the next game restart. If a robot receives a fourth yellow card this is considered a red card and the robot must leave the field for the remainder of the game. Other rules, present in human soccer, like corner kicks, throw-ins and the off side rule don't apply to RoboCup at this moment. The duration of a game is 20 minutes, divided in two halves of each 10 minutes. During a 15 minute half-time break teams can change the batteries if necessary and fix their robots. The complete FIFA and RoboCup rules can be found in [34].

1.5 Overview of thesis

The main question discussed in this thesis is how to create robotic soccer players which collaborate and play as a team. The chosen design has been put into practice in the Team skills module of Clockwork Orange, the Dutch RoboSoccer team.

We will now provide a short overview of the rest of the thesis. Chapter 2 introduces hardware, software and communication architecture of Clockwork Orange which forms the basis of the work described in this thesis. Chapter 3 discusses literature about coordinating teams of agents and the approaches other middle-size league teams have chosen on this level.

The main contributions of this thesis can be found in chapters 4 and 5. The first describes our team coordination mechanism based on the dynamic distribution of roles according to a global team strategy. The latter discusses our approach for choosing the next action based on an assigned role which will benefit the team the most. Chapter 6 discusses results obtained at RoboCup 2001, the world cup tournament in August 2001 in Seattle, USA. Finally chapter 7 ends the thesis with some conclusions and suggestions for future research.

The work described in this thesis has been carried out at the Intelligent Autonomous Systems group of the Faculty of Science of the University of Amsterdam. It builds on software developed there and on software originating from the Pattern Recognition group of the Delft University of Technology.

Chapter 2

Clockwork Orange

Clockwork Orange [11, 45] is the Dutch RoboSoccer team, named after the nickname of the human Dutch national soccer team of the seventies. It is a collaboration between the Utrecht University, the Delft University of Technology and the University of Amsterdam. The team participates in the RoboCup middle-size robot league. This year Utrecht University could sadly not contribute to the team during RoboCup 2001 because of severe hardware problems.

This chapter will give an overview of the hardware, software and communication architecture of the team in order to describe the setting of which the Team skills module forms a part.

2.1 Hardware

First of all the robots have to be introduced, as no game can be played without players. Our lineup consists of six Nomad Scouts and one Pioneer 2. Delft University of Technology and University of Amsterdam both own three Nomad Scouts adapted for the soccer game while the Pioneer 2 belongs to Utrecht University.

The Pioneer 2 (figure 2.1) from ActivMedia Robotics [1] uses a laser range finder, 16 ultrasonic sonar sensors, odometry sensors, and a camera for sensing. It uses a pneumatic kick device as actuator.

The Nomad Super Scout II (figure 2.2) from Nomadic Technologies has odometry sensors, one camera for sensing and a pneumatic driven kick device as effector. Its 16 ultrasonic sensors and its tactile bumper ring are not used for RoboCup. The specifications of both types of robots can be found in table 2.1.

A functional overview of the hardware setup of our Nomad Scouts is shown in figure 2.3. A low level motor board controls the original hardware from Nomadic Technologies while the high level computer is connected to all the custom hardware like the camera, the kick device and the wireless Ethernet. The original hardware includes the motor board, the sonars, the tactile bumper ring and the motors.

The only hardware sensors we use for RoboCup are the camera and the wheel encoders. The encoders should be able to tell the amount of rotation of each wheel from which the robot's path can be calculated. However, Nomadic Technologies decided not to return the real rotations of the wheels as one would

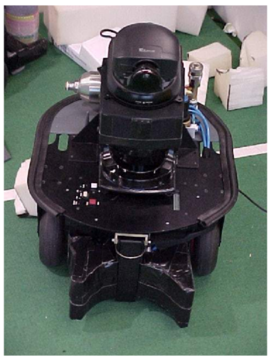


Figure 2.1: A Pioneer 2.

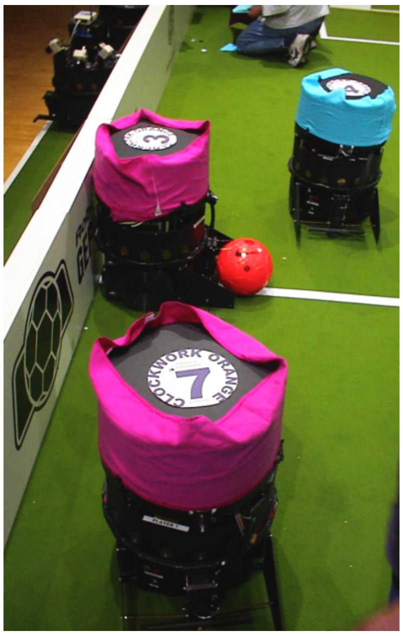


Figure 2.2: Three Nomad Scouts.

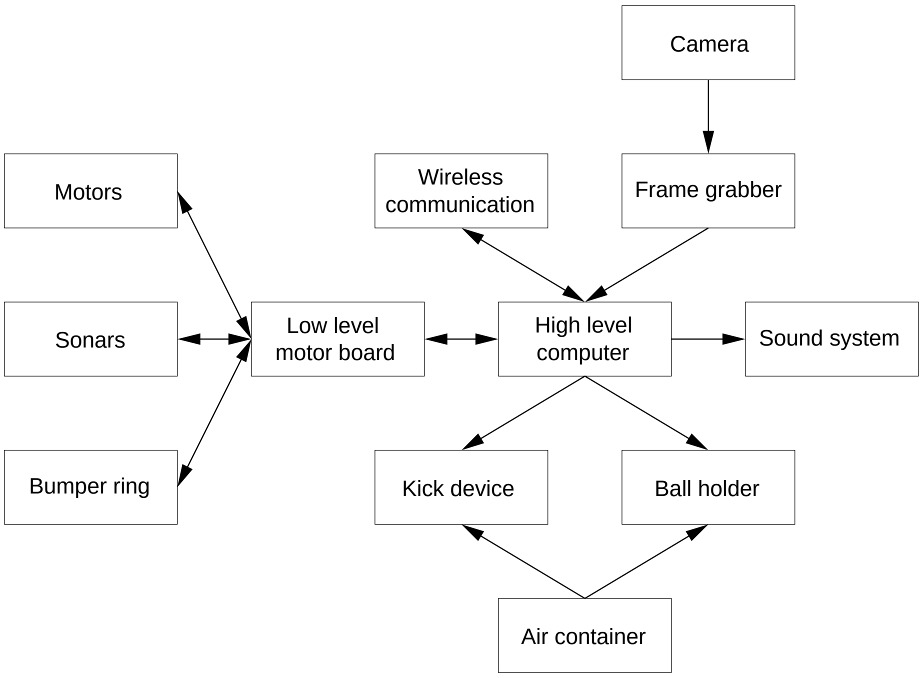


Figure 2.3: Hardware setup of our Nomad Scout.

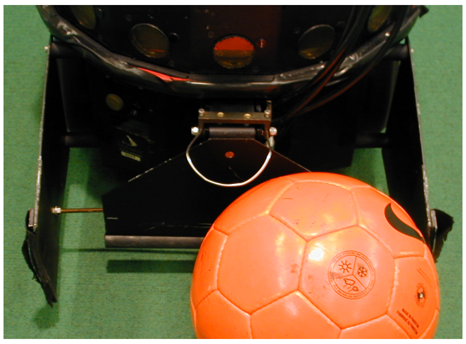


Figure 2.4: The kicker, ball handlers and the ball handling mechanism.

expect but instead the desired speed as calculated by the motor board. The motor board calculates these speeds for both wheels to get smooth accelerations and decelerations when the user sends the desired speed and heading of the robot. In practice this means one cannot discriminate between normal driving or pushing against a static object.

The **Camera** is a color PAL-camera from JAI Camera Solutions equipped with a lens with a horizontal angle of view of 81.2°. Several features are configurable on this camera, such as the auto white balance feature. The configuration can be set manually or by software through a serial port. For reliable color detection is it necessary that the auto-white balance can be disabled. We use a standard WinTV **Frame grabber** which has been set to grab 12 frames per second in 640×240 pixels YUV format at 3×8 bits color depth.

Our Nomads have been heavily customized and currently have four actuators: the wheels, the kick mechanism, the ball holder mechanism and the sound system. The **Kick device** (figure 2.4) uses compressed air from a 16 bar **Air container** on top of the robot. The kicking mechanism is located between two ball handlers and together they include the ball for 7 cm (one third of the ball diameter, as specified by the rules, see section 1.4). Our goalkeeper's kicking device is half a meter wide (the maximum allowed width), since its objective is not to handle the ball carefully but just kick it across the field. On top of the kicker mechanism resides the **Ball holder** mechanism: a small pneumatic device used to tap the ball for holding. Current RoboCup regulations have deemed this device illegal, so we don't use any more.

The **Motors** are being controlled by a Motorola MC68332 processor on a **Low level motor board**. Motor commands are only accepted five times a second which severely limits the amount of control one has over the robot. The Nomad can for instance not be allowed to drive at maximum speed because when obstacles are suddenly being detected the robot might not be able to stop in time.

The onboard sensors, **Sonars** and **Bumper ring** are being read out by the Motorola processor. The bumper ring is not used as there are two ball handlers in front of the robot, preventing the bumper ring from touching any obstacle. Furthermore, driving backwards is not recommended if your only camera is facing forward. The sonars are currently not employed as they pose a risk to the system: the potential difference when in use can cause disturbances in the

Specification	Nomad Scout	Pioneer 2
Diameter	41 cm	44 cm
Height	60 cm	22 cm
Payload	5 kg	20 kg
Weight	25 kg	9 kg
Battery power	300 watt hour	252 watt hour
Battery duration in game	1 hour	30 min
Max. Speed	1 m/s	2 m/s
Max. Acceleration	2 m/s^2	2 m/s^2
Special sensors	Camera	Laser + camera

Table 2.1: The specifications of our robots.

low motor board.

Our software runs on the **High level computer**, an Intel Pentium 233 MHz based computer on an industrial biscuit board (5.75” by 8”). Communication with the low level board is accomplished using a serial port and a PC 104 board is used to control the kicking mechanism. An onboard sound chip forms our **Sound system** together with two small speakers.

For **Wireless communication** between robots we use a BreezeCOM [9] IEEE 802.11 wireless Ethernet system with a maximum data rate of 3 Mbps giving the robot an action radius of up to 1 km outdoors. It uses the 2.4 GHz band and finds a frequency which is not yet fully utilized by means of frequency hopping. During tournaments the BreezeCOM system proved very reliable in contrast with the WaveLan system which suffers heavily from interference of other wireless networking devices in the 2.4 GHz band.

2.2 Software architecture

Not only do there exist two different types of robots in our team, Delft / Amsterdam and Utrecht also have chosen different approaches in intelligent agent design. Being a heterogeneous team both in hardware and software makes the team skills task more challenging, since you do want to act as one team instead of two sub teams. The team has has not been heterogeneous this year due to hardware problems of the Pioneer 2, but the team coordination mechanism should be designed in such a way it can coordinate heterogeneous teams.

Utrecht University’s Pioneer 2 uses an extended version of the subsumption architecture [10] in which particular behaviors such as *Get_Ball*, *Dribble*, and *Score* compete for controlling the robot. All behaviors can react in correspondence to the world model or to direct sensor data provided by the camera and laser range finder.

The Nomad Scouts operate on a hybrid architecture which looks like a classical, hierarchical approach but whose units have a high degree of autonomy.

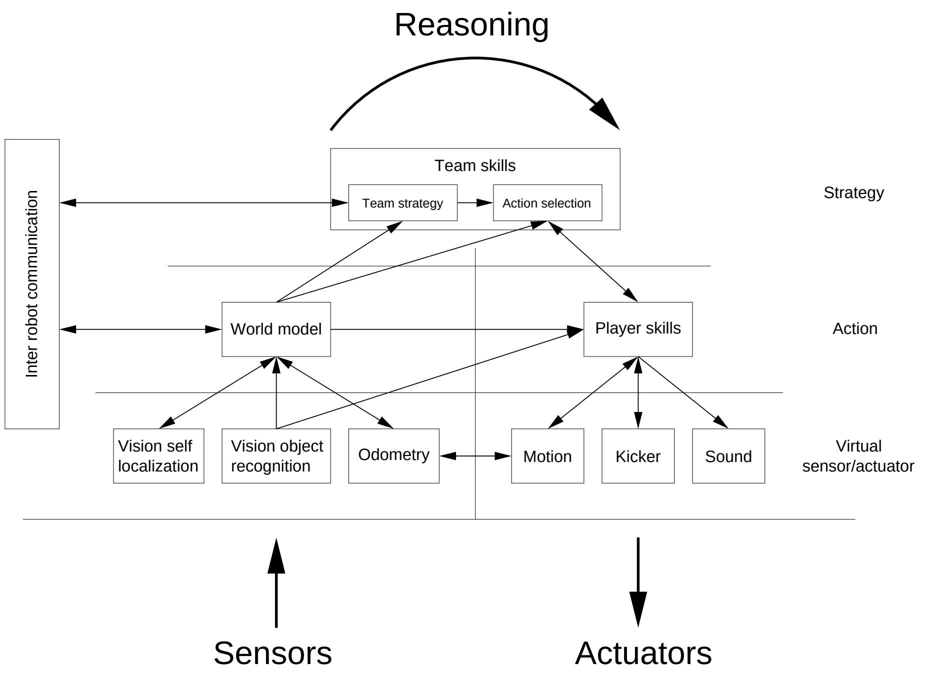


Figure 2.5: Functional architecture decomposition.

Figure 2.5 depicts a functional composition of this architecture. It can be divided in three levels:

Virtual sensor/actuator level Here reside the virtual sensors and the actuator interfaces to the hardware.

Action level Control of the local robot on this level, as well as combining information from the virtual sensors in a world model.

Strategy level Determining the team strategy and the next action of the robot with respect to his teammates is handled on this level.

A good example of an autonomous unit is the Player skills module. It is a reactive unit as the arrow toward it from Vision object recognition shows, which allows it to do real-time collision avoidance.

One should keep in mind the architecture has been designed a few years ago ([26], short version appeared in [28]) and evolved during time. Master's students from two different universities each worked on their part of the project. A lot of effort has to be made to coordinate the different modules. The functional decomposition in figure 2.5 describes the architecture used for RoboCup 2001, given some minor violations introduced under pressure of oncoming tournaments.

2.2.1 Virtual sensor/actuator level

The lowest level in our software architecture is the link between software and hardware. These virtual sensors and actuators usually don't communicate directly with the hardware but use device drivers from the operating system as intermediates. We use RedHat Linux 6.2 as operating system and development environment.

The **Odometry** module is a virtual sensor which keeps track of the motion of the robot. It gets it data from the motor board[1] and estimates the odometry error of it. The University of Michigan Benchmark test as described in [7] has been run to estimate the systematic error, which increases with the traveled distance. For a typical Nomad this error turned out to be 20 cm after driving a 3 m square (1.7%).

The camera supplies the data for our two other virtual sensors: object recognition and self localization. All vision processing uses 24 bit color images of 320 $\times$ 240 pixel resolution, half of the original height. This reduction is necessary because of the limited processing power available. The camera and vision system on each robot have to undergo a lengthy calibration procedure every time lighting conditions change.

Vision object recognition [27] extracts objects like the ball, the goals and robots from these camera images. This is done via color detection in a pie piece in UV space. In the YUV space Y is the intensity and the UV space describes the color. In this space we take the point of the white color as the center. Going out of the center increases the saturation of the color. Due to specular reflection the saturation of an object can vary, so to determine its color irrespective of saturation we describe the colors by pie pieces.

Size and position of these objects are estimated [27] and this information is passed on to the World model and to the Player skills module. The latter is also notified when a large portion of an image is white, which usually indicates the robot is standing in front of a wall. The Player skills module can react by taking appropriate measures to avoid hitting it.

As knowing your own position is crucial for constructing a global world model we also use the camera for self localization. **Vision self localization** [25] uses the lines on the field and the goals. The self localization mechanism involves a global and a local method. The global method first splits the screen into multiple regions of interest and finds straight lines in each of these. These are matched to the world model giving an estimate of the position. Because of the symmetry of the field multiple candidates are found. We use Multiple Hypothesis Tracking to follow all the candidates over time, updating them for our own movement.

The local method [4] is used to verify these candidates and to correct for changes. We check all the candidates, verifying their heading and distance to the goals and the overall result given to us by the local method. The loop is repeated until one candidate remains, which is used to update our position.

Our self localization mechanism requires the robot to move around because otherwise candidates cannot be discarded. For this reason the Team skills module is notified when we've lost our position. To give an idea of the performance of the self localization: during a total period of more than three hours our position was known 49% of the time. Further results can be found in section A.5.

Driving is controlled by the **Motion** module which communicates motor commands to the low-level processor of the Nomad. These motor commands are for example the desired speed or acceleration of the left and the right wheel. The **Kicker** module controls the pneumatic kick mechanism used for shooting

[1]The Motion module passes this data to the Odometry module, since only the Motion module can communicate directly with the low level motor board.

at goal. The **Sound** module plays sounds on request of other modules for entertaining and debugging purposes.

2.2.2 Action and strategy levels

On top of the virtual sensor/actuator level resides the action level, in which local control of the robot as well the building and maintaining of a world model takes place. The strategy level of the software architecture controls the team strategy and action selection.

The main reactive and autonomous component of our architecture is the **Player skills** module [3]. It tries to fulfill the desired actions of the Team skills module while at the same time keeping in mind its other behaviors, which have a higher priority. These are the collision avoidance behavior and the license to kill behavior. The collision avoidance behavior makes sure a robot does not run into obstacles such as robots and walls. If a robot has the ball and sees a large portion of the enemy goal the license to kill behavior makes it shoot at it. These two reactive behaviors get their information directly from the Vision object recognition system. Other actions include simple *Goto(x,y,φ,v)*, *ShootAtAngle(φ)* and *Seek(ball)* but also more sophisticated actions like *DribbleToObject(theirGoal)* or *GotoObject(ball)* are available. The Player skills module notifies the Team skills module when an action has been completed or aborted, in which case it specifies the reason.

Below a summary of the available actions is given, in which *x,y* is a position, φ is an angle, v is a speed and *object* is either *ball*, *yellowGoal* or *blueGoal*:

Turn(φ), rotate the robot around its axis until it reaches the desired heading, relative to the world. Also turns relative to the robot can be specified.

TurnToObject(object), turn to face the *object*.

Shoot(), kick the ball straight ahead at maximum force.

TurnShoot(φ), kick the ball at specified angle. This is accomplished by driving while turning followed by shot. If the robot threatens to lose the ball while turning it will shoot it at that time.

Goto(x,y,φ,v), move to a specified position while avoiding obstacles. Either desired heading or desired speed at end point can be requested. Forward and backward motion is supported, but since the robot can only detect obstacles in front of it driving backwards is not recommended during normal operation.

GotoObject(object), if *object* is *ball* move toward the ball and try to control it, a maneuver which will be called “chasing the ball” throughout this thesis. If *object* is one of the goals move toward it until the robot is one meter away from them (you usually don’t want it to actually drive across the goal line).

Seek(object), keep turning until the robot sees the requested object.

Dribble(x,y,v), carefully drive to the requested position trying to keep control over the ball. If the robot loses the ball don’t immediately declare the action a failure but try to regain it. Dribbling with the ball is very difficult due to the shape restrictions and the limit of five motor commands per second.

DribbleToObject(object), dribble toward one of the goals. The Player skills module keeps a memory of the relative heading it last saw the goals, in order to be able to find them again.

To enable a distributed form of control and improve robustness, the **World model** [17] (summarized in [19]) is also distributed. Each of the robots in our team locally maintains a world model. This model consists of the dynamic objects in the game (i.e. the robots and the ball) and the static objects (the goals). Each of these objects has static features such as its color, shape and classification and more dynamic features such as its position, speed and the estimated uncertainty of the object's position estimate. Given the observations of the various objects in the game done by the virtual sensor modules it is up to the world model to combine them into an as complete and accurate view of the world as possible.

Since we want to construct an absolute model, which enables us to distribute it among our teammates (and thus facilitating cooperative team behavior), it is necessary that the robot knows its own position. Two sensors are used to estimate this position. The first is the odometry sensor, the second is the vision self localization. The odometry sensor continuously keeps track of our position. It however, as explained before, will lose its correctness after driving long distances or after collisions. So from time to time the vision self localization is used to reset this position. It is up to the world model to combine the information of these two sensors in order to maintain a good estimate of the position. The vision-based self localization will typically come with some delay, so the world model has to keep a history of prior position estimates, enabling it to combine the vision-based estimate with the odometry information at the time the vision-based self localization was done. The new combined position estimate will be the weighted average of the two positions, where the weights are given by the uncertainty estimates. This will result in a position shift from the original estimate, which can be used to adjust all the later odometry positions, giving us an as accurate as possible estimate of the robot's position at the current time.

The other objects in the game are being detected by the vision system. Each observation will come with information about the shape or color which is detected (e.g. an orange ball or black robot), and the position of the observed object relative to the robot. The world model will, given its own position, convert the ego-relative position of the observed object to a world-relative position. Next it will try to match the observation to any of the known objects in the world model. This is done by checking whether there is an object that matches the observation's shape within the uncertainty region. When an observation is finally matched to an object, it will be inserted in the object's observation history. Using the linear least squares algorithm the world model tries to fit a linear function through this list of observations. Extrapolating this linear function enables us to do an accurate estimate of the object's position at the current time, while filtering out noise. It also enables us to estimate the speed and heading of the observed object.

The world model keeps a list of the objects which are being tracked (i.e. the objects have been observed no longer than 4 seconds ago) and a list of objects which are known to be in the game but which aren't tracked anymore.

This prevents the world model from having to throw away valuable information about the objects while at the same time keeping a model without any false or outdated information.

Due to its limited field of view and the fact that parts of the field will be occluded by the objects on them, the robots will often perceive only a small part of the game. Having an absolute world model enables us to share it with the other robots in the team. Sharing the local world models and combining these will enable us to increase the accuracy and completeness of the world models of all the robots in our team. For instance, when the ball is perceived by only one of the robots, sharing the world models will result in all the robots knowing where the ball is.

One of the other features of the world model is the ball possession protocol. When a robot detects that the ball is in its possession (i.e. the ball lies between its ball handlers) a notification is send to the world model, which will record that the robot has the ball and will send a message to all the other robots in the team telling them the team has the ball. Detecting whether the opponent team has the ball is slightly harder since our vision system does not enable us to detect whether the ball is in the opponent's ball handling mechanism. For now we assume an opponent has ball possession if the ball has been within its reach for some time.

The advantage of an absolute shared world model is that team coordination is greatly simplified. The task of the **Team skills** module is twofold: it coordinates team strategy and it chooses the next action the Player skills module should execute. These subtasks are described in chapters 4 and 5 respectively.

2.3 Communication

Communication between modules is handled by a message passing system. The system is based on UDP and uses Linux kernel message queues. In a *message passing system* processes communicate by sending each other messages. A *message* is an atomic unit with a message type and a string of bits containing data. The message is sent from one module to another using a method of delivery. In a message-based paradigm the abstraction is that there are direct connections between modules that serve as conduits for messages. These conduits do not necessarily have to exist on physical level. The routing for a message that travels from one module to another occurs at hardware and lower software levels.

2.3.1 The message passing system

In the message passing system [51] we use, each module has a message queue. For every module, all incoming messages are stored in their message queue in FIFO order. When a module is ready to process a message, it will issue a request for the next message on its queue and process it. This process continues until all messages are processed or the module terminates. Message queues allow a module to send information when desirable without having to wait until the other party is ready to receive. Without some form of buffering, modules would be subjected to a blocking wait if the other party isn't ready. Such a situation can easily lead to a communication deadlock for the involved modules. The message passing system has been designed with shared memory capabilities in mind, but so far none have been implemented. Both the intra robot communication as well as the **Inter robot communication** is based on message passing for information exchange.

Communication between modules in different layers of the hierarchy can only be initiated by the module which is in the higher layer. This ensures that control always resides with the higher module. In communication between modules in the same hierarchical layer, any of the modules may initiate communication. Three different types of communication were defined in our message passing system to accommodate for downward and upward communication flows: Orders, Questions, and Triggered notifications. Orders are intended for the downward communication flow. Questions and Triggered notifications should be used for the upward communication flow.

Orders Downward communication through the hierarchy is established by using the order communication type. The *order* type is typically used when a module asks another module to perform a specific action. The orders type is used to send orders toward the actuators. For instance, Team skills action selection can send orders to the Player skills module (e.g. *DribbleToObject(object)*), which in turn can send orders to the Motion module. It should however not be possible for the Motion module to send an order to the Player skills module (which resides on a higher level in the hierarchy).

Two types of communication are used for upward communication though the hierarchy: questions and triggered notifications.

Questions If a higher level module is interested in information of a lower level module, the *question* type will be used to ask for the information. The lower level module will answer the question. Upward communication through the hierarchy is less trivial than downward. If it is not allowed for lower level modules to initiate communication with higher level modules it is a bit tricky to use event-like constructions for receiving sensor information. Clearly, questions may be suitable to receive periodic information, but since they cannot be initiated by lower level modules they can not be used for this purpose. Also, each Question resembles one piece of information, so one piece of information requires two messages: one question and one answer. This introduces double latency and bandwidth, which is no restriction to use it for sporadic or irregular information exchange. It is, however, for periodic information exchange. It would not make sense to use the question type for periodic information, because the question would be repeated periodically, which is a waste of bandwidth.

Another communication type used in the upward communication flow which more or less solves these issues is the triggered notification type.

Triggered notifications The *triggered notification* type can be seen as a single request for a particular type of information from a higher level module to a lower level module which honors the request by sending the information to the higher level module, either periodically or when new information is available. If the higher level module is no longer interested in the type of information the lower level module sends, it simply requests the lower level module to stop sending.

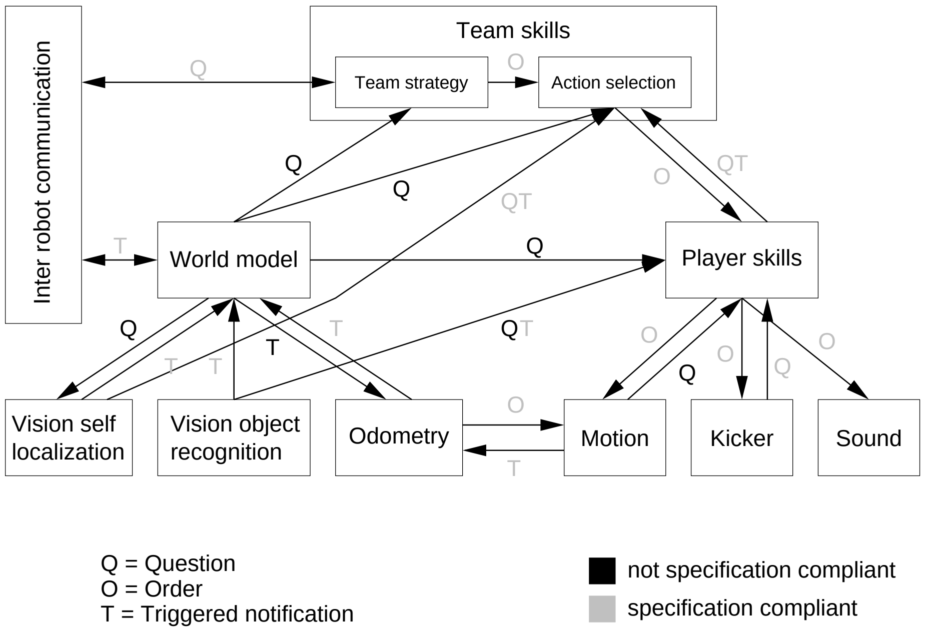


Figure 2.6: Communication details. The directions of the vertices correspond to the directions of the information flow.

The mechanism we use for the triggered notification communication type resembles a subscription mechanism in some aspects. A *subscription mechanism* is based upon the so-called push strategy: as soon as a producer of data has new data available it will publish the data so it is locally available to subscribers, regardless whether a subscriber needs that particular instance of data on that particular moment. When the moment arrives the subscriber needs the data, it simply reads the data from its local buffer and continues processing. The main advantage is that when the data is needed it has already been transferred over the network, and is locally available to the subscriber. An important difference between the triggered notification and a true subscription mechanism is that the latter has anonymous publishers/subscribers. In the mechanism we use the publisher and subscriber know where the information comes from. Therefore our architecture lacks the increased modularity which architectures based upon a true subscription mechanism have.

2.3.2 Deviations from the communication specification

Given the types of communication and knowing communication can only be initiated by the highest module of two communicating modules, it should be easy to identify which type of communication is used for a vertex in figure 2.5. Unfortunately this is not the case. In figure 2.6, a more detailed picture of the communication types used in the architecture is given, in which a distinction is made between the use of communication types whose usage corresponds to the specification of the communication types as given in the documentation of the message system [51] and the use of communication types whose usage does not correspond to it. Most of the non-compliant usage of the types, is the usage of the question type for periodic information, which can for instance be seen in communication between the World model and the Team skills module.

A more serious type of deviation is the reverse of the initiation of communication which occurs in the communication between the Vision self localization and the World model. The initiation of communication by a lower module may seem very innocent, but in an architecture in which control is arranged hierarchically, it is very important control does reside with the higher module. If communication is initiated from below, the higher module has no control over the information it receives, and therefore is (at least partially) dependent on when and if the lower module is willing to send information. It becomes unclear which of the two modules is in control over the other, this is not a problem if the modules are part of the same hierarchical layer. If however two modules from different layers are involved it should be - metaphorically speaking - always be clear who is captain on the ship.

Chapter 3

Related work

Before designing and implementing the Team skills module one should define its exact task and carefully examine existing literature related to this module. For the Team skills module this results in literature regarding cooperation in multi-agent systems and a look at the approaches other middle-size teams have chosen on this level. To enable a reasonable evaluation of these approaches and related work the demands RoboCup middle-size league places on the Team skills module have to be enumerated.

3.1 Team skills task

After introducing the software architecture of Clockwork Orange in the previous chapter now is the time to specify the exact job of the Team skills module. If you look at figure 2.5 on page 14 its only responsibility is guiding the robot by means of the Player skills module.

Choosing the next action in our sense-plan-act architecture requires some reasoning about which action is preferable, not only with respect to the immediate reward but also taking into account the longer term reward a sequence of actions might have. However, in a dynamic environment such as RoboCup with unpredictable adversaries it is not possible to accurately predict more than let's say the coming 30 seconds of the game. This can mean a low reward on the short term might be preferably to a higher reward on a longer term.

Another complicating factor in the action planning problem is the fact a robot not only has to deal with opponents and the ball but also with his teammates. For instance, a robot should be able to predict that one of his teammates will go after the ball and take this into account in its own action planning: it might make itself more useful by falling back to defend instead of also pursuing the ball, usually leading to a clash.

This calls for a form of team play, the level on which robot soccer starts to resemble human soccer. Team play should ascend the level of action planning (since the world is so unpredictable) and try to incorporate higher level strategy elements like formations and common sense. For example, it is a good thing to have at least one field player close to his own goal area ready to defend it by intercepting any oncoming enemy attacker, even though you cannot exactly predict at what time in the future this will happen. However, it is common knowledge that in games like soccer one should prepare for a counterattack.

So a need for action selection leads to a need for action planning which leads to a need for team coordination. Before looking at approaches available to tackle this team coordination problem we will first give a more detailed account of the nature of the RoboCup middle-size league domain from a multi-agent system design point of view.

3.2 Requirements on the Team skills module

Next we'll describe requirements the environment places on the Team skills module, in order to be able to judge approaches found in literature on their merits for this particular task.

- In the RoboCup middle-size league there are only four agents in a team, of which the goalkeeper doesn't actively participate in team play since it has a different mechanical design (its camera is rotated 90°). The methods chosen should also be able to deal with the fact that not always all three field players are in play due to referee decisions or hardware failures.

- Between team members is no struggle for resources like the ball or free space, they all cooperate to achieve a common goal. There is no need for negotiating who may take possession of the ball and try to score. For a robot it should not matter whether it or a teammate scores, an aspect in which robot soccer can differ from human soccer. The free space resource should also be carefully divided, the team should spread out as opposed to clinging together.

- An important aspect of the game is that cooperation is not strictly necessary for success. If the team would only consist of a single player it could still be able to achieve the goal of the game: score more goals than your opponents. However, team play can increase the chances on success, even if it is only used to make sure a robot stays back defending the goal. Indeed, team play should first of all be used to coordinate who will get the ball and try to score since all robots going after the ball at the same time is very counterproductive. This concept has been dubbed "herd ball" in youth soccer: all children chase the ball in one big group without any kind of position play.

- We want decentralized control instead of one robot (or an off-field computer) telling everyone what to do. This is hopefully more robust since it lacks a single point of failure. Distributed control is also a step closer to a generic multi-agent system as central control does not scale very well.

- Communication is possible and necessary for two reasons. The Utrecht robot will probably never run the same Team skills module as the others, so in order to create non-trivial team behavior communication must be used. In a perfect world communication between team members with the same software would not be necessary. Everyone has the same knowledge regarding the world and therefore can know each team member's internal state. Unfortunately, the world models of each robot will have inconsistencies (see [17] for an evaluation). Using communication will improve robustness.

- However, the system should not depend on communication as it could fail during matches. The team needs to have a form of graceful degradation: if only one robot remains active it should go after the ball and try to score.
- Real world systems should take into account the fact their sensors are rather imperfect, which makes precise knowledge about its own position and the position of other objects not trivial. The sensor readings are also likely to be continuous instead of discrete.
- The RoboCup environment is very dynamic, so the robots should be able to respond quickly to changes in their view of the world. The ball can be at the opposite end of the field in a matter of seconds, not only due to robots rushing with the ball but also because most teams employ some kind of kicking device.

3.3 Team coordination frameworks

Quite a number of people have thought about agents interacting, and the next sections summarize some of these methods for team cooperation. Parker [41] discriminates between two families of robot cooperation approaches: "swarm" and "intentional" cooperation. Swarm cooperation focuses on the emergent cooperative behavior in large groups of robots, each of which individually has limited capabilities. Cooperation is a side-effect of the behavior which governs the individual robot. The behavior is a collection of simple control rules which work directly on the sensory input without the aid of global world model.

When robots achieve purposeful cooperation built on their individual higher level capabilities the cooperation is called intentional. It reflects the design philosophy in which the robots explicitly communicate and coordinate actions with teammates. These systems can be divided in a perception, control and actuation part. They are allowed a global world model which simplifies the team coordination.

Which type of cooperation is desirable depends on the application. Swarm cooperation is typically used for large numbers of homogeneous agents all performing more or less the same task. Applications in which the size of the environment is large relative to the robot's size are suited for swarm cooperation. When the application requires several distinct tasks to be performed under time constraints intentional cooperation is more suited.

We view RoboCup middle-size league as an application in which intentional cooperation is the logical choice for a cooperation model. We will see that other teams such as the Trackies don't agree: "A deliberative approach to the team strategy acquisition seems useless in such a dynamic and hostile environment." [46]. In the middle-size league however one should try to prevent more than one robot going after the ball at the same time as they obstruct each other, which would call for techniques covered by intentional cooperation. Several tasks such as defending and attacking are required, the number of robots is very limited, timing is crucial for success and the robots can be heterogeneous. Therefore we will describe several team coordination frameworks which are based on the intentional cooperation model, after which their applicability in RoboCup will be discussed.

3.3.1 Joint Intentions Theory

Cohen and Levesque's joint intentions theory [13, 14, 39] is based on the notion that joint action by a group of agents is more than the union of the simultaneous individual actions, even if those actions are coordinated. It is an extension of their belief-goal-commitment model of mental states [12] in which intentions are specified as internal commitments to perform an action while in a certain mental state. Intention is modeled as a composite concept specifying what the agent has chosen and how the agent is committed to that choice. A joint intention is a joint commitment to perform an action together. In a more strict form one can say that a team of agents jointly intends, relative to some escape condition, to do an action iff the members have a joint persistent goal relative to that condition of their having done the action and, moreover, having done it mutually believing throughout that they were doing it.

A persistent goal to achieve p can be defined relative to some escape condition q. An agent has such a persistent goal iff it believes three things: p is currently false, it wants p to be true eventually and it is true (and it knows it) that the second condition will continue to hold until it comes to believe either that p is true, or that it will never be true, or that q is false. A joint persistent goal extends this definition by also requiring that they mutually believe to each have p as a weak achievement goal. A agent has a weak achievement goal relative to q and with respect to a team to accomplish p if either of these conditions holds: the agent has a normal goal to accomplish p or the agent believes that p is true, will never be true or is irrelevant, but has as a goal that the status of p be mutually believed by all team members.

To summarize: when a team jointly commits to achieving p, they mutually believe that they each have p as an achievement goal. However, as time passes they can't be certain each still has p as a goal, since one of them may have noticed on his own that the goal is finished and is busy letting the whole team know. So if a team member discovers that a goal is true, impossible or irrelevant but that this is not mutually known it will be left with a persistent goal to make the status of the goal mutually known.

The question remains how joint intentions lead to individual ones. This is similar to the way a joint persistent goal leads to individual goals among the team members. Two types of multi-agent action are being considered: those that arise from more basic actions performed concurrently and those that are formed from a sequence of more basic actions. Agents who jointly intend concurrent actions also individually intend to do their parts as long as the joint intention is still operative. This means that a team member can drop a commitment if it notices that another member cannot do its share.

3.3.2 SharedPlan Theory

SharedPlan theory was originally developed by Grosz and Sidner [22] to model the collaborative behavior demonstrated by humans or other agents in dialogues.

Together with Kraus Grosz [20, 21] extended this model to facilitate the construction of agents that are fully collaborative but also provide a framework for the intentional structure of dialogue.

To have a collaborative plan for a complex action in a team of agents four conditions must hold. First of all, the team has a mutual belief of a recipe which can be partial. Each agent has individual intentions that the action be done and has intentions that his team members succeed in their subactions. The fourth condition is that individual or collaborative plans for the subactions are known. With each action a set of recipes for accomplishing that action is associated. A recipe can include actions at different levels of abstraction and the parameters of an action need not be completely specified. The actions in a recipe can be complex actions or *basic level actions.* The latter are executable in the appropriate situation and not further defined. For the complex actions there will be other recipes, thus creating a complete recipe tree. The basic level actions are always restricted to one executing agent whereas the complex actions can be multi-agent.

A SharedPlan requires two kinds of intentions. An agent can *intend to* do some action or he can *intend that* some proposition holds. Intending-to commits an agent to means-ends analysis and further down the road to acting, while intending-that is used for things like contemplating about subplans or helping teammates. An agent can only have an intention-to attitude toward an action which it can execute itself. But an agent could for instance intend-that a team member does a certain action.

Intending-that is a major property of SharedPlan theory. It helps to avoid adopting intention-to's that conflict with those arising from the team's plan and it can lead to helpful behavior. Intention-that's also require agents to communicate their progress to the rest of the team. Furthermore they facilitate the coordination and integration of subplans.

3.3.3 Generalized Partial Global Planning

Generalized Partial Global Planning (GPGP) is a domain-independent framework for coordinating small teams of agents developed by Decker and Lesser [16, 37, 38]. Each agent constructs its own local view of the activities (task structures) that it intends to pursue in the near future, and the relationships among them. This view can be augmented by information from other agents in his team, which makes the view no longer entirely local. Individual coordination mechanisms help to construct these partial views, and to recognize and respond to particular task structure relationships by making commitments to other agents. The commitments are inter-agent contracts to perform certain tasks at certain times. GPGP uses a worth-oriented approach for describing the environment, in which a goal is not black or white but rather has a degree of achievement associated with it.

The coordination mechanisms of GPGP operate on task structures. A task structure contains the following five pieces of information: first the top-level goals of an agent, one or more possible ways they could be achieved (these are trees whose leaves are basic action instantiations called methods), a quantitative definition of the value of a task or method in terms of how it affects objectives, furthermore task-task relationships that indicate how basic actions or abstract task achievement affect tasks elsewhere in the task structure and finally a task-resource relationship that indicates how the execution of a task affects the state of the resource which it uses during execution and vice versa. Originally there were five coordination mechanisms in GPGP: communicate non-local views, communicate appropriate results, avoid redundancy and handle hard and soft coordination relationships extending between tasks at two agents. These five basic mechanisms form a domain-independent set for basic agent teamwork.

The notion of commitment used is similar to the one of Cohen and Levesque [12]. The commitments result in more coherent, coordinated behavior by specifying the effects of other agent's actions on the tasks an agent will execute, when they will execute and where their results will transmitted. The commitments tend to fall into three categories [38]: *deadline* commitments are contracts to perform work by certain deadline, *earliest start time* commitments are agreements to hold off on performing a certain tasks until a certain time has passed, and *do* commitments are agreements to perform certain tasks without a particular time constraint. GPGP can be extended by specifying these types of commitments a priori, which could cut down on the coordination overhead.

3.3.4 Flexible teamwork

Tambe presents an implemented general model of flexible teamwork called STEAM [48]. It is based on joint intentions theory but also borrows from the SharedPlan theory. STEAM uses joint intentions as the basic building blocks of teamwork, but as in the SharedPlan theory team members build up a complex hierarchical structure of joint intentions, individual intentions and beliefs about other team member's intentions.

The new concept in STEAM are team operators (reactive team plans). When agents select a team operator for execution, they instantiate a team's joint intentions. As with individual operators, team operators also consist of precondition, application and termination rules. When an operator is selected the executing agent (which can be just one agent, a team or subteam) is determined, and hence if it is a individual or team operator. Each agent maintains it own private state and team state, which describes the mutual beliefs of the team. It is also possible to participate in subteams, in which case the subteam's state also has to be maintained. After a team operator has been established as a joint intention and is being executed by the team, it can only be terminated by updating the team state.

STEAM also provides mechanisms for monitoring team performance, which can improve the quality of teamwork. For each team operator monitoring conditions are specified which can be used to determine achievement, impossibility or irrelevancy of the operator. These specifications are based on the notion of a role. A role is an abstract specification of the set of activities an individual undertakes in service of the team's overall activity. When an agent changes or cannot perform its role it should announce it. If this agent is critical to achievement of the team operator according to the monitoring conditions STEAM will start replanning. If the replanning doesn't succeed and the failure turns out to be a *critical role failure* then the team should be reconfigured.

Tambe has extended STEAM to also accommodate persistent teams [49]. RoboCup for instance is a domain in which the teams can be persistent, provided that no team member will become disabled or be replaced during one half of a game. Roles can now be of two types: persistent roles, the long-term assignments of roles to individuals or subteams and task-specific roles, shorter-term assignment of roles based on the current task and situation. STEAM-L (STEAM with lookahead) tries to maximize long-term expected team utility. The lookahead reasoning is modeled as a Markov decision process (MDP) in the state space of a team's future states. A state in the state-space is an agent's model of the team's overall state and includes the mutual beliefs and the relevant private beliefs of other team members. At a state, an agent simulates the execution of one of the available operators, which correspond to the actions in the MDP. Each action has a cost associated with it. The MDP's states are the future states of the team, state transitions correspond to the team's current and future executable operators or actions and transition probabilities are the probabilities of going from one state to another after an action. This can all be represented in a decision tree.

3.3.5 Joint Responsibility

Joint Responsibility is an implementable model of teamwork proposed by Jennings [24] that is also based on joint intentions theory. This model has been implemented in a system called GRATE* which incorporates generic knowledge of cooperation and situation assessment. On top of the concept of joint goals joint recipe commitments are defined.

Individual recipe commitment means that every agent should try to follow the common recipe (remain committed), and do the actions it has agreed to undertake unless: the outcome of a recipe step is already available, the recipe does not lead to the accomplishment of the joint goal and is thus invalid, a step of the recipe cannot be executed (i.e. due to some malfunction) or one of the agreed actions has not been performed properly. Joint recipe commitment specifies that when an agent drops his individual recipe commitment for one of those four reasons he should let the team know. The team can then determine if the recipe has to be abandoned or can be adjusted to still accomplish the joint goal.

Joint responsibility requires that all team members execute the common recipe adhering to the principle of joint recipe commitment and that all the agents mutually know what they are doing while they are doing it. There is a difference between dropping a joint recipe commitment or dropping a joint goal. In the latter case the joint action is clearly over. But after abandoning a joint recipe commitment because one of the team members becomes uncommitted there is still work to be done. For instance, a rearrangement in the sequence of actions which yields the same result can be tried. Or if a step in the recipe has failed it might be retried.

3.3.6 Discussion

Building a system on top of one of the frameworks mentioned above is not desirable for coordinating our team of robots. They are much too elaborate for a team of three agents with only two top level goals (score goals and prevent opponent goals). Furthermore, it would not be feasible due to time constraints, since it would have to be done in the limited period of time of about half a year (the time allotted for a graduation project). However, a number of good ideas can be extracted from them.

When you apply the notion of joint intentions to the soccer domain you could identify a team strategy as a joint persistent goal. The joint persistent goal could be “score a goal” or “prevent the other team from scoring”. Each robot has a duty to let his teammates know if the team strategy has become irrelevant (e.g. when ball possession changes). From this joint intention individual intentions can be derived, which could be viewed as a robot wanting to attack or defend. Tambe calls such an individual intention a role, which is the building block of many of approaches chosen by other middle-size league teams described in the next section.

STEAM-L offers an interesting solution to the action selection problem in which it searches through the space of the future team states. This kind of lookahead reasoning requires knowledge about what type of action a teammate will take next. The *intend to* and *intend that* attitudes from SharedPlan theory can be applied in this process. For instance, a robot intends to fulfill a defending role while it intends that a teammate will try to score a goal.

A concept which pops up in several of these frameworks is monitoring yourself or team members. A robot should be able to detect when it cannot play its role (e.g. because it’s stuck between the wall and an opponent for example) and for instance incorporate this knowledge in its utility for a certain role estimate.

3.4 Middle-size league teams

It is necessary to look at the approaches on team coordination and action selection other middle-size teams have chosen. It takes quite some time before a team is ready to think about team coordination. Some other teams have been competing longer and already have experience on this level.

We have singled out several teams which have been active on the team coordination level. All teams except ART (which has been split up into several smaller teams last year) have competed at the 2001 world championships in Seattle. Table 3.1 introduces shortly the general makeup of each team. Listed are the institute(s) responsible, the robot base on which their robots are built and the sensors of each team.

Table 3.2 shows the discriminating features of their team play approaches: type of cooperation mechanism, whether or not the team is homogeneous, if the robots employ communication and if there is central unit in their architecture. Goal keeping robots weren’t considered when answering the question regarding homogeneity: they usually have hardware adapted for their task and their control software is built for speed, not cooperation. With central unit we do not mean the human(s) remotely starting and stopping the team but more the question whether the team’s approach is completely distributed or contains a central component.

Next are a series of short descriptions of the team skills approach chosen by each team.

3.4.1 CS Freiburg

Extended behavior networks form the basis for Freiburg’s action selection [40]. Their players distribute roles amongst themselves, namely an active, support and strategic role [52, 53]. The active robot tries to get the ball, a support player

Team	Institute	Robot base	Sensors
CS Freiburg	Albert-Ludwigs-Universität Freiburg	Pioneer 1	laser range finder, camera
ART	Univ. di {Parma, Padova, Genova, Roma "La Sapienza"}, Politecnico di Milano	Custom, Pioneer 1	camera, sonars, infrared sensors, omnidirectional camera
CoPS	Universität Stuttgart	Nomad	laser range finder, camera, sonars
AGILO	Technische Universität München	Pioneer 1	camera
Ulm Sparrows	Universität Ulm	Custom (Sparrow-99)	Camera, sonars, infrared sensors
GMD	GMD (Fraunhofer-Gesellschaft)	Custom	360° panning camera, gyroscope, infrared sensors
Trackies	Osaka University	Custom	camera, omnidirectional camera
Clockwork Orange	Univ. of Amsterdam, Delft Univ. of Technology, Utrecht Univ.	Nomad, Pioneer 2[1]	camera, laser range finder[1]

1 Not available for RoboCup 2001.

Table 3.1: General comparison between several middle-size teams.

Team	Cooperation type	Homogeneous?	Communication?	Central unit
CS Freiburg	dynamic roles	yes	yes	world model[1]
ART	dynamic roles	no	yes	none
CoPS	dynamic roles	yes	yes	none
AGILO	predefined plans	yes	yes	none
Ulm Sparrows	predefined plans	yes	no	none
GMD	dynamic behaviors	yes	yes	none
Trackies	"swarm" learning	yes[2]	no	coach
Clockwork Orange	dynamic roles	no[3]	yes	none

1 No central unit on team play level, but the world model is maintained on an off-field computer.

2 Only hardware is homogeneous, software is different per robot.

3 But the team was homogeneous on the road to and at RoboCup 2001.

attempts to assist by positioning itself appropriately and a strategic player occupies a defending position. Each robot determines its utility to pursue a certain role and communicates it to its teammates. After comparing with the utilities it receives a robot chooses his role.

3.4.2 ART

Roles are also distributed among the players of ART [2]. The roles are a main attacker which demands ball possession, a supporting attacker and a defender. A zone in the field (attack, defense or field center) is associated with each role. The protocol for distribution of the roles among the players is based on two utility functions. Every cycle each robot computes both utility functions based on its perception of the world and sends the results to its teammates. The robot with the lowest result for the first utility function becomes the main attacker while the remaining two field players compare the value of the second utility function. The one with the lowest value takes the role of supporting attacker and the other one becomes defender.

3.4.3 CoPS

The architecture of CoPS [35] is structured in a reflexive, a tactical and a strategical layer. The reflexive layer is responsible for sensing and interacting with the hardware. Inside the tactical layer a distributed shared world model is maintained and actions are generated. The strategical layer decides on long term strategies like attacking or defending and allows for inter robot negotiations. CoPS uses a dynamic role assignment protocol [30] very similar to the one of CS Freiburg.

3.4.4 AGILO RoboCuppers

The default selection of an appropriate action is based on the action with the highest expected utility in the current situation [44]. To take into account a strategic assessment and the intentions of teammates AGILO developed a robot soccer playbook. The playbook is a library of plan schemata for team play which are triggered by certain opportunities, for example the opponent team leaving one side open. A high level controller monitors the game in order to detect these opportunities and decide whether or not to start a certain plan.

3.4.5 Ulm Sparrows

The Ulm Sparrows [33, 50] focus on building strong individual players before attempting cooperative team play. They want to share their higher level software layers in a simulation league and a middle-size league team. Coordinated team play in soccer is viewed as emergent behavior which can be achieved without explicit communication. Each agent has a database of stored cooperative play patterns. If from its perceived environment an agent can match itself and at least one other teammate in such a pattern, the pattern is executed.

3.4.6 GMD

GMD [8] developed a mathematical model for robot behaviors called Dual Dynamics. It combines a behavior based approach with a dynamical systems representation of actions and goals. It allows for smooth changes between behavior modes which result in reactive, fast and natural motions of the robots. Team coordination is achieved by distributing shared variables of several behavior systems between the teammates.

3.4.7 Trackies

The Trackies [46] have a fundamentally different approach from the teams described above: they use cooperative behaviors without any planning from which cooperation in the dynamic environment should emerge. A learning approach [47] in team strategy is chosen in which a coach can coordinate a combination of players each of which has a fixed policy. The coach tries to estimate the opponent's team strategy, assuming they have the same choices in roles and keep their team strategy fixed. The coach can switch the fixed policies in its own teammates to explore the opponent's strategy. After having explored enough the coach will choose the combination of roles for his own players which exploit the other team's strategy the most.

3.4.8 Discussion

CS Freiburg, ART and CoPS all use a similar dynamic role system, which uses the concept of rating your own utility for each role and communicating the results. This resembles the approach of Tambe in STEAM and can be mapped on joint intentions theory. Advantage of a dynamic role system is its applicability in a wider context than RoboCup: one could imagine the same concept being applied to search and rescue or reconnaissance multi-agent systems.

The playbook approach of AGILO and Ulm in which the robots choose from a number of predefined plans resembles SharedPlan theory. The predefined plan is the recipe which each robot knows, and all the individual or collaborative plans for the subactions are known. In a way this approach looks more suited for American football as for soccer. An American football game consists of a long sequence of dead-ball situations interleaved with short executions of predefined patterns.

GMD has a rather novel method to creating cooperative team behavior. They approach the problem from a dynamical systems perspective instead of the deliberative, intentional view of the role assignment and predefined plans methods. The Trackies also believe a deliberative approach is not desirably in such a dynamic and hostile environment. However, their learning approach assumes a very simple makeup of the opponent team: no communication, the team is built on the same roles as your own and their strategy is fixed during a match.

3.5 Discussion

For our team coordination we have chosen to extend the dynamic role assignment approach chosen by several other middle-size league teams. It is a form of intentional cooperation, much more flexible than choosing from a database of predefined plans, rather general in the sense the idea can be applied to other domains and it seems a natural way similar to human soccer to coordinate your soccer players.

We have added the notion of a global team strategy, which can be viewed as the joint intention of the team. With each team strategy different roles are associated, each of which has a priority to make sure the most important roles are assigned first. This is important in case not all field players are participating in the game. We hope assigning specific roles to robots will stop them interfering with each other.

The robots should distribute the roles amongst themselves, we don't want one robot or an off-field computer telling everybody what to do for reasons of robustness. Assigning roles requires a mechanism to tell whether a robot is fit for a certain role. The utility functions used by other teams seem a good idea, so we have extended them by having a utility function for each role taking into account not only the robot's distance to the ball but also its position on the field.

The action selection used in STEAM-L seems a flexible way to coordinate the individual actions while taking into account the expected actions of your teammates. This is the main benefit of having a shared world model, but one should take care not to rely on these predictions too much in such a dynamic environment. We have added a backup mode to our system to be used in case a robot doesn't know its own position.

Chapter 4

Team coordination

After choosing a team coordination approach in the previous chapter we will now discuss in detail our design for dynamically distributing roles. The main problem faced here is how to measure a robot's usefulness in a certain role.

4.1 Team strategy

The standard role distributing schemes as designed by other middle-size league teams [2, 30, 52] all seem to employ just one team strategy: *attack*. However if one defines ball possession as a prerequisite for being able to attack one can see that in a typical middle-size league not as much time as one would like is spent attacking. It usually takes a lot of time to find and obtain the ball (especially if the other team is doing its best to keep it from you) before the big rushing forward with the ball at your foot can commence. Therefore we have extended this model with two more team strategies: *defend* and *intercept*.

We have divided the soccer game in three states: either your team has the ball and *attack*s, the other team has the ball and your team *defend*s, or nobody has ball possession and your team tries to obtain it by *intercept*ing it. For simplicity we assume the game is characterized by ball possession alone. The world model tells a robot whether it, its team, the other team or nobody has ball possession or if it is unknown who controls the ball, which means none of your teammates sees the ball. The two latter cases are considered equal since they lead to the same team strategy: if nobody has the ball you chase after it and if the ball's position is not known you first have to find it and then chase it. Temporal inconsistencies regarding ball possession between the world models on each robot can lead to confusion about which team strategy should be followed as we will see in chapter 6.

The appropriate team strategy is determined by a simple finite state machine (fig. 4.1) which takes as inputs the question "Who has the ball?" and the current team strategy. A problem of a finite state machine could be that it is prone to oscillations between the team states. Since it takes some time for a team has decided what robot will fulfill which role rapid switching between states could result in the team being confused. To prevent such a unfortunate situation the protocol for deciding on team strategy and distributing roles presented in section 4.5 "locks" the team strategy during a cycle of the decision making

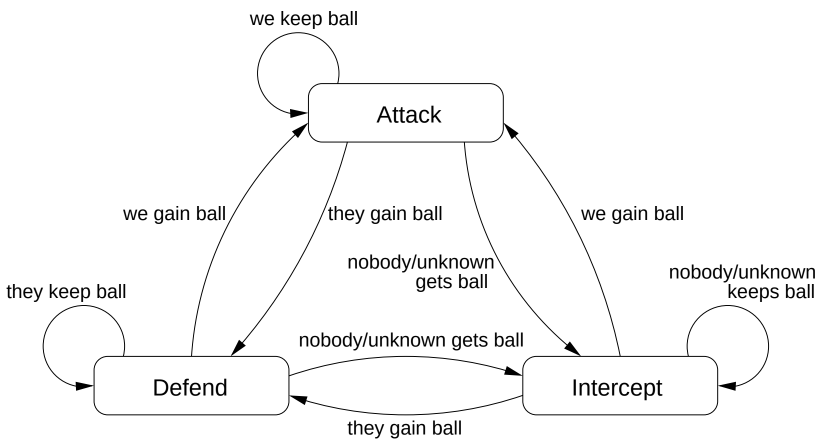


Figure 4.1: Finite state machine to determine the team strategy.

process. Problems arising from the temporary inconsistencies between world model due to network lag and processing time will be addressed at that point.

The general idea behind each team strategy will be explained next, a detailed discussion will be presented when introducing the roles in the next section.

Attack When your team has the ball you should attack, which in RoboCup middle-size league boils down to dribbling to the enemy goal and trying to shoot past their goalkeeper. "Wall passes[1]" are not yet within the mechanical capabilities of the robots, the amount of control needed for these kinds of complex actions is not present. Although even regular passing from one attacker to another one is a bit farfetched, a supporting attacker might be able to regain the ball after the main attacker made a shot on goal. As in human soccer not all field players should rush forward, but at least one player should remain back to help the goalkeeper defend.

Intercept As said before there is a difference between knowing that nobody controls the ball and not knowing where the ball is, but both these states are mapped on the team strategy Intercept. Two robots will try to intercept the ball, which means seeking it in the second case and going after it in the first case. Special care should be taken to avoid clashes of the two robots chasing the ball, the action planning mechanism of the next chapter takes care of this. Another way to avoid two robots storming at the ball at the same time is assigning only one a chase-the-ball type of role, but this would place a great responsibility on one robot. Having only one robot going after the ball is risky as it cannot detect when it is not able to fulfill its role which could result in no robot trying to gain ball possession. Anyway, there will still remain one robot behind to defend.

Defend If one of your opponents controls the ball your team should fall back on their own half of the field. One of your players will try to block the path to the

[1]A wall pass is when a player passes the ball to a team mate who plays it right back without stopping it first. The maneuver is also known as "one–two".

Strategy	#1	#2	#3	#4
Attack	DefendGoal	AttackWithBall	PassiveDefend	AttackWithoutBall
Defend	DefendGoal	ActiveDefend	PassiveDefend	ActiveDefend
Intercept	DefendGoal	InterceptBall	PassiveDefend	InterceptBall

Table 4.1: Distribution of roles associated with each team strategy.

goal while the other two[1] try to regain the ball by harassing the opponent with the ball. This gives you some depth in your defense to make sure an incoming attacker has to pass by at least two defenders before being able to take a shot at goal. Once the team has regained the ball (and will switch to team strategy Attack) a good action may be to kick the ball forward instead of trying to dribble all the way across the field.

Next step is to describe the roles for the individual robots, since they make up the team strategy.

4.2 Roles

A team strategy is nothing more than a distribution of certain roles over the available field players. We have assigned priorities to each role in a certain team strategy, since during a game not all three field players may be in play but some roles are more important than others. Robots are sometimes removed and reinserted during play due to hardware problems (injuries) or due to a decision of the referee.

Table 4.1 shows the roles for each team strategy, with #1 being the most and #4 being the least important role. The goalkeeper is always assigned role #1. If there is one field player available only role #2 will be assigned, with two field players active roles #2 and #3 will assigned etc. Role #2 is always a ball oriented role: with only one field player ready for duty you want it to go after the ball.

DefendGoal (DG) No field player will ever be assigned this role, in fact the goalkeeper is not controlled by the Team skills module but by its own simple and fast action selection module. However, it is possible to incorporate a goal keeping role in the team coordination system. This has no high priority at the moment since the goalkeeper has fundamentally different hardware: its camera is rotated 90°.

PassiveDefend (PD) A passively defending robot waits in front of its own goal for the opponent team to attack. It should try to block the path of an incoming attacker to the goal and only go after the ball when it is close. The defender should pay attention not to block the sight of the goalkeeper, but at the moment

[1]The same argument as in team strategy Intercept for having two robots going after the ball instead of one applies.

a field player cannot communicate with the goalkeeper on the team coordination level as this feature is not included in the current goal keeping module.

ActiveDefend (AD) The other defending role is ActiveDefend, whose task is to chase the ball on your own half of the field. This is the first of the ball chasing roles, whose main task it is to obtain control of the ball. This is usually accomplished by going after it under the assumption that it is just as hard to control the ball for an opponent as it is for you.

InterceptBall (IB) This role is very similar to the previous one, the difference is that InterceptBall has an offensive purpose which means a robot in this role wants to go toward the enemy goal.

AttackWithBall (AWB) Attacking usually means dribbling toward the enemy goal while avoiding defending robots and taking a shot in the most promising corner. However, when a robot gains ball possession near to his own goal line it might be more beneficiary to just kick the ball along the line to the enemy half of the field and run after it if none of his teammates can pick it up. Chances of making it all the way across the field while controlling the ball are slim.

AttackWithoutBall (AWoB) This role describes an auxiliary attacker, a robot which together with the main attacker moves toward the enemy goal, preferably on the opposite side of the field. Its task is to regain the ball if the main attacker loses it, but it should try to avoid getting in the way of its teammate.

SelfLocalization This is a special role, in the sense that its only purpose is to facilitate the self localization process: behave in a way which has a high chance of the Vision self localization regaining our position. If the Vision self localization notices our current position estimate cannot be correct (e.g. you see a goal in a direction you shouldn't) it informs the Team skills module[1]. When the robot is not performing an important task (like attacking, chasing the ball etc.) it will switch to this role until it has regained it position or found something better to do.

The distributed role approaches already in use at RoboCup usually have three roles: an attacker, a supporting attacker and a defender. These can be respectively identified with AttackWithBall, AttackWithoutBall and PassiveDefend. Adding team strategies suggests we also have to differentiate the existing roles. The added roles InterceptBall and ActiveDefend seemed necessary in our view, but one could easily argue a different set of roles or team strategies is necessary and sufficient to play a decent game of robot soccer. Regrettably we do not have the resources (two complete teams) to do experiments regarding this issue, and also a realistic simulator was not available at the time.

The current distribution of roles as depicted in table 4.1 is the parameter one can change to easily influence the overall team behavior. A human coach can decide to make the team more defensive or aggressive by shifting the priorities

[1]It is more logical for the Team skills module to get this information from the World model; the reason there is no arrow from Vision self localization to Team skills in figure 2.5 on page 14, but does appear in information flow diagram 2.6 (p. 20).

Role	Time to reach ball	Position evaluation	Ball possession
PassiveDefend		✓	
ActiveDefend	✓	✓	
InterceptBall	✓	✓	
AttackWithBall	✓	✓	✓
AttackWithoutBall		✓	

Table 4.2: Components of utility functions for each role a field player can take.

of the roles associated with each team strategy or by for instance replacing the PassiveDefend roles with ActiveDefend ones.

4.3 Utility functions

To decide which robot should be assigned which role a mechanism based on utility functions is used. Each role has its utility function which tries to measure how well suited a robot is for this role in the current state of the world. This measure can be based on three pieces of information:

- The time a robot expects it needs to reach the ball. This is important for roles that have to chase after the ball in some way or another: ActiveDefend, InterceptBall and AttackWithBall.
- How well the position of a robot is suited for the role. A evaluation mechanism for this purpose will be presented in the next section. We need this measure for the non ball oriented roles PassiveDefend and AttackWithoutBall, but it is also relevant for the ball oriented roles to fall back upon when the position of the ball is unknown.
- Whether or not a robot has possession of the ball. Only relevant for the role AttackWithBall for obvious reasons.

So not all measures are used for each role; table 4.2 summarizes this information.

We define a utility function $\mathcal{U}_{\text{role}}$ as a function which calculates an estimate of the utility of a robot for a role in a world on an ordinal scale of $[0, 1]$, on which higher means better suited for the role. The utilities calculated for different roles cannot be compared: their values are only relevant in evaluating different robots for the same role. The function has a predefined range to enable the influence of domain specific information: for instance if a robot controls the ball its utility for role AttackWithBall will be 1 to make sure it will be assigned this role. The input of $\mathcal{U}_{\text{role}}$ consists of the position $(x_{\text{player}}\ y_{\text{player}})^T$ and heading θ_{player} of the robot and the position of the ball $((x_{\text{ball}}\ y_{\text{ball}})^T)$ in this world. If no robot in the team sees the ball the position of the ball is undefined. Velocities of ball and player are not yet taken into account since their estimates coming from the world model are not reliable enough.

$$\vec{p} = \begin{pmatrix} x_{\text{player}} \\ y_{\text{player}} \\ \theta_{\text{player}} \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} x_{\text{ball}} \\ y_{\text{ball}} \end{pmatrix}$$

Function $\mathcal{T}$ returns an estimate of the time the robot at $\vec{p}$ needs to move to the ball at $\vec{b}$. The estimate is based on the shortest trajectory from robot to ball: the Euclidean distance (obtained from function D) between their centers and the smallest angle (obtained from function Φ) the robot needs to turn to face the ball. No real path planning occurs: possible obstacles between the two are ignored. We do take into account the fact whether or not the robot sees the ball with his own camera. If the robot does not, which means it is an observation from a teammate, an additional cost is added to the estimate. This cost reflects the fact that a direct observation is more reliable than an indirect one.

$$\mathcal{T}(\vec{p},\vec{b}) = g(v_d D(\vec{p},\vec{b}) + v_\varphi \Phi(\vec{p},\vec{b})), \\ \text{where} \begin{cases} g = 1 & \text{if player sees the ball with own camera} \\ g = 1.5 & \text{else} \end{cases}, \tag{4.1}$$

in which v_d is the maximum translational speed and v_φ the maximum rotational speed of the robot.

We need to convert $\mathcal{T}$ to a utility function, which means mapping it on a monotonic function which is 1 if the time to reach the ball is 0 and gradually drops toward zero when the time increases. As we only compare values of the same function to each other, any function abiding the constraints above will do. We have chosen:

$$\mathcal{T}'(\vec{p},\vec{b}) = e^{-\mathcal{T}(\vec{p},\vec{b})}. \tag{4.2}$$

The time a robots expects to need to reach the ball is only a useful component of utility functions for roles whose purpose is to obtain control of the ball: AttackWithBall, InterceptBall and ActiveDefend. For the other two roles we have to come up with a different way of estimating a robot's suitability for them: we look at the location of the robot in the field. For instance, a defender should be near its own goal area while an (auxiliary) attacker should reside close to the other goal.

Now we are ready to define the utility functions for each role. The utility function for DefendGoal $\mathcal{U}_{\text{DG}}$ is trivial since at this moment no field player will ever aspire to be a goalkeeper.

$$\mathcal{U}_{\text{DG}}(\vec{p},\vec{b}) = 0 \tag{4.3a}$$

Both passive (not ball oriented) roles have their utility solely determined by their position based evaluation. Below are the utility functions for PassiveDefend $\mathcal{U}_{\text{PD}}$ and AttackWithoutBall $\mathcal{U}_{\text{AWoB}}$

$$\mathcal{U}_{\text{PD}}(\vec{p},\vec{b}) = \mathcal{H}_{\text{PD}}(\vec{p}) \tag{4.3b}$$

$$\mathcal{U}_{\text{AWoB}}(\vec{p},\vec{b}) = \mathcal{H}_{\text{AWoB}}(\vec{p}) \tag{4.3c}$$

in which $\mathcal{H}_{\text{role}}(\vec{p})$ represents the position based role evaluation defined in (4.5) on page 41, denoting the value of the position evaluation for the role in question at position $\vec{p}$.

For the ball oriented, active roles the primary measure of utility for the role is not the position based role evaluation but how close a robot is to the ball. Only if the ball position is not known the utility function falls back on the position based evaluation for the role. We reasonably assume all robots possess the same knowledge regarding the status of the ball (tracked by the team or not), short temporal discrepancies left aside. This way no discontinuities between the two parts of the utility functions will occur. Below are the utility functions for ActiveDefend $\mathcal{U}_{\text{AD}}$ and InterceptBall $\mathcal{U}_{\text{IB}}$.

$$\mathcal{U}_{\text{AD}}(\vec{p},\vec{b}) = \begin{cases} \mathcal{T}'(\vec{p},\vec{b}) & \text{if } \vec{b} \text{ is defined} \\ \mathcal{H}_{\text{AD}}(\vec{p}) & \text{else} \end{cases} \tag{4.3d}$$

$$\mathcal{U}_{\text{IB}}(\vec{p},\vec{b}) = \begin{cases} \mathcal{T}'(\vec{p},\vec{b}) & \text{if } \vec{b} \text{ is defined} \\ \mathcal{H}_{\text{IB}}(\vec{p}) & \text{else} \end{cases} \tag{4.3e}$$

The utility function for AttackWithBall $\mathcal{U}_{\text{AWB}}$ is similar to the previous two, except that if the robot in question has control over the ball it is of course the most suitable candidate for this role. Note that $\mathcal{T}'(\vec{p},\vec{b})$ will actually never return 1 since both the ball and the robot have non zero diameter and $\mathcal{T}$ uses the centers of both.

$$\mathcal{U}_{\text{AWB}}(\vec{p},\vec{b}) = \begin{cases} 1 & \text{if player has ball} \\ \mathcal{T}'(\vec{p},\vec{b}) & \text{else if } \vec{b} \text{ is defined} \\ \mathcal{H}_{\text{AWB}}(\vec{p}) & \text{else} \end{cases} \tag{4.3f}$$

Our definition of the utility functions is not complete without a description of the position based evaluation used in them.

4.4 Position based role evaluation

For an attacking role it is easy to think of a useful utility function but this is more complicated for a defending role, since an attacker only has to focus on the ball while a defender should assume a good position waiting for things to come. Other teams [2, 30] have solved this problem in a kind of negative way: if a robot is not the attacker or the supporting attacker then it will assume a defensive role. However, as the position of a player defines it as defensive it is logical to take its position into account when evaluating its utility for a defensive role.

The position based role evaluation functions should in our opinion have the following components, some of which are inspired by potential field theory [31, 36]:

- The general flavor of the role: the difference between a defensive or an offensive role. A defender is much more useful near its own goal while an

Role	Slope	Attractor(s)	Repeller(s)
PassiveDefend	2×defensive	none	opponent 3/4 of the field
ActiveDefend	1×defensive	none	opponent 1/2 of the field
InterceptBall	1×offensive	opponent goal	none
AttackWithBall	2×offensive	opponent goal	none
AttackWithoutBall	2×offensive	opponent goal	none

Table 4.3: Description of the position based role evaluation for each role.

attacker should reside near the opponent goal. We call this component the slope of the evaluation function since it makes a visualization of the function slope from one the goal up to the other.

- A number of attractors: objects or regions whose positions it is desirable for a robot to be at. The attractors emanate a positive force which makes positions closer to them more desirable than ones further away. This is used to represent the positive influence the opponent goal has on an attacker. Attractors do not all posses the same desirability and the way their influence reaches across the field also varies.
- A number of repellers: objects or regions whose positions it is bad for a robot to be at. They are exactly the opposite of attractors with the same characteristics to enable simple combination of their influences. Repellers are used for denoting certain "forbidden" areas for defending roles: a defender for instance should not cross the halfway line so we mark the opponent half of the field as a repeller.

We believe a combination of these three criteria suffices for a role evaluation based on the position of the robot. The attractor and repeller scheme is versatile enough to be used in the action planning approach presented in the next chapter.

The interpretation of these three components for the five field player roles are given in table 4.3. The evaluation function $\mathcal{H}'$ is a combination of the three:

$$\mathcal{H}'_{\text{role}}(\vec{p}) = \mathcal{S}_{\text{role}}(\vec{p}) + \sum_i \mathcal{F}_{att}(\vec{p}, a_{\text{role},i}) + \sum_j \mathcal{F}_{rep}(\vec{p}, r_{\text{role},j}), \tag{4.4}$$

where $\mathcal{S}$ (defined in (4.8)) denotes the value of the slope for the robot at $\vec{p}$, $\mathcal{F}_{att}$ (defined in (4.9)) the attracting force and $\mathcal{F}_{rep}$ (defined in (4.10)) the repulsing force applied to the robot at $\vec{p}$.

In order for $\mathcal{H}'$ to qualify as a utility function we need to map it to a monotonically ascending function which ranges on $[0, 1]$. Since we only want to compare results of this function the exact specification does not matter. We have chosen the simplest one:

$$\mathcal{H}_{\text{role}}(\vec{p}) = \frac{\mathcal{H}'_{\text{role}}(\vec{p}) + c_0}{c_1}, \tag{4.5}$$

where c_0 is a positive constant large enough to ensure $\mathcal{H}_{\text{role}}(\vec{p})$ is always positive

$$\forall \text{role}, \vec{p} : \mathcal{H}'_{\text{role}}(\vec{p}) > -c_0 \tag{4.6}$$

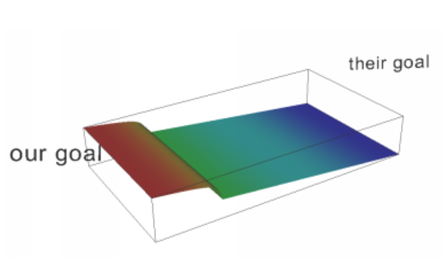


Figure 4.2: Position based evaluation for role PassiveDefend.

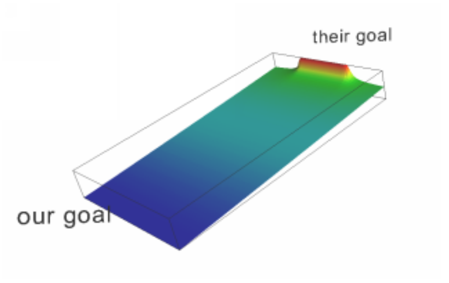


Figure 4.3: Position based evaluation for role InterceptBall.

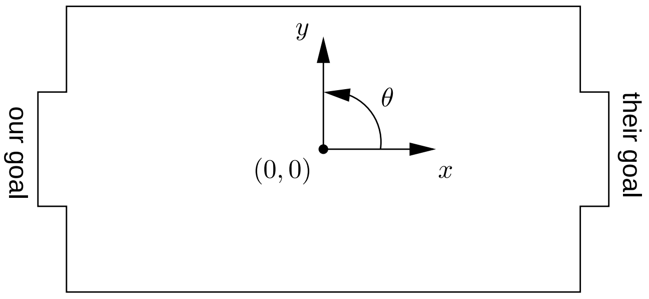


Figure 4.4: Coordinate system of Clockwork Orange.

and where c_1 is a positive constant large enough to ensure $\mathcal{H}_{\text{role}}(\vec{p})$ always ranges on $[0, 1]$

$$c_1 \geq \max_{\text{role},\vec{p}} \mathcal{H}'_{\text{role}}(\vec{p}) + c_0. \tag{4.7}$$

A visualization of the position based evaluations can be seen in figures 4.2 and 4.3 for roles PassiveDefend and InterceptBall. The evaluation has been calculated at every ten centimeters and greatly amplified for visualization purposes.

4.4.1 Slope

The slope component $\mathcal{S}$ of the position based role evaluation has two features: the direction in which it slopes and its fall between one goal and the other. The direction is up from our goal to their goal for offensive roles and down from our goal to theirs for defensive roles. The fall indicates to what degree a role is offensive or defensive: the steeper the slope the bigger the influence of the slope component.

Figure 4.4 introduces the coordinate system of Clockwork Orange, the slope depends on the x value of the position of the robot.

$$\mathcal{S}_{\text{role}}(\vec{p}) = \frac{p_x e_{\text{role}} f_{\text{role}}}{l}$$
$$\text{where} \begin{cases} e = 1 & \text{if role is offensive} \\ e = -1 & \text{if role is defensive} \end{cases}, \tag{4.8}$$

where f is a positive constant denoting the total fall of the slope and where l is a constant denoting the total length of the field.

The Slope column of table 4.3 shows whether a role is offensive or defensive and the value of f. The fall can be of default magnitude ($1\times$) or twice as big ($2\times$).

4.4.2 Attractors and repellers

As in potential field theory we employ attractors and repellers, but that is where the similarities end. Attractors are just negative repellers (or vice versa depending on your point of view) so they can negate each others effects which is intentional: for instance in the next chapter we will see that if a robot is attracted by the ball it should not be scared of approaching it by a repulsing opponent player.

Each role has its combination of attractors and repellers as specified in table 4.3. Some of the features of attractors and repellers described here may not seem necessary or useful for position based role evaluation, but they are used in the action planning process presented in the next chapter and are included here for completeness.

We want flexibility in defining attracting/repelling objects or regions, so we have defined attractors/repellers to have circular or rectangular shapes:

- Circular ones with a certain diameter for objects like robots and the ball. Distance is calculated to the perimeter.
- Points (circles with zero diameter) can be used to represent corners.
- Rectangular attractors/repellers with both a non zero length and width represent "good" or "bad" areas. Distance to a rectangle is defined as the distance to the nearest point of the rectangle.
- Lines (rectangles with zero length or width) are suited for representing one dimensional objects like walls or the goal line between the two goalposts.

The influence of an attractor or repeller should be inversely proportional with the shortest distance between them and the point under consideration. The influence is proportional to the strength of the attractor/repeller divided by the distance between it and the position of the robot. Distance is returned by function D which follows the specifications mentioned above. When a point lies within an attractor or repeller the distance is considered to be zero.

The influence at certain distance is controlled by two factors: the strength of an attractor/repeller and its significance. The strength U represents how desirable an attractor/repeller is: it is positive for attractors and negative for repellers. The influence of attractor/repeller can never exceed its strength: the influence of an attractor a is never more than its strength $U(a)$ and the influence of a repeller r is never less than its strength $U(r)$.

The second feature of an attractor or repeller is its significance ξ: the degree in which its influence can still be felt across field. The main reason for adding this feature was the idea that the influence of an object as important as the ball should be felt across the whole field while the repulsing force of an opponent can be more limited.

Attractor/repeller	Strength U
Opponent goal	3
Opponent 1/2, 3/4 of the field	-3

Table 4.4: Strength of the attractor and repeller during the RoboCup 2001 tournament.

The force of attractor a and repeller r on the robot at $\vec{p}$ are given by by $\mathcal{F}_{att}$ resp. $\mathcal{F}_{rep}$:

$$\mathcal{F}_{att}(\vec{p}, a) = \min\left(\frac{\xi_a\, U(a)}{D(\vec{p}, P(a))}, U(a)\right), \text{ where } U(a) > 0 \tag{4.9}$$

$$\mathcal{F}_{rep}(\vec{p}, r) = \max\left(\frac{\xi_r\, U(r)}{D(\vec{p}, P(r))}, U(r)\right), \text{ where } U(r) < 0. \tag{4.10}$$

The values of U in use during the RoboCup 2001 tournament can be found in table 4.4.

4.5 Assignment of roles

Our shared world model simplifies the task of assigning roles to robots. As soon as one robot notices the current team strategy is no longer applicable due to a change in ball possession it alerts its teammates by telling them the new team strategy. The robot continues by calculating its own utility for each of the roles in the new team strategy (see table 4.1) and transmits this information to its teammates. Upon receiving the new team strategy they do the same. While waiting for a certain time on the utility scores of teammates each robot calculates these scores based on its own world model. Utility scores coming in from a teammate have preference however, which introduces a level of redundancy.

After receiving utility scores of each teammate or when a timeout occurred, each robot selects its role by comparing the utility scores each team member has for each role available, starting with the most important role. The robot stays in this role until it notices the team strategy is no longer useful or until it receives a message from a teammate containing information of this kind. We try to limit the impact of oscillations between team strategies by letting the robot ignore all incoming new team strategies while still busy waiting for utility scores from teammates. To repair possible inconsistencies the distribution protocol is repeated every half a second even if the robot has not noticed a change in ball possession.

Chapter 5

Action planning

Besides coordinating the team as a whole the Team skills module has a second task: selecting the next action for a robot which will benefit the team as a whole the most. More specifically: plan a sequence of actions which are the most rewarding for the team on a longer term. To make this job easier each robot has been assigned a role as described in the previous chapter. A complicating factor however is the fact a robot does not always know with reasonable certainty its own position and heading. We deal with this problem by providing two modes of action planning: an absolute one in which the robot uses the world model and a relative one in which the robot bases its action only on information obtained locally.

We view planning as search in the possible future states of the team and Markov decision processes are a good way to represent our search problem.

5.1 Markov decision processes

Action selection can be defined as the problem of finding an optimal policy for mapping an agent's internal state to an action. Optimal is defined as maximizing a certain optimality criterion. The Markov property holds when the transition probabilities of moving from one state to another given an action depend only on the current state. So when no history of previous states is needed such a decision problem is called a Markov decision process (MDP).

A MDP can be described as a tuple $\langle S, A, T, R \rangle$ [29].

S is a finite set of states of the world
A is a finite set of actions

In a MDP setting an agent can always observe its state perfectly. Next to a set of states and a set of actions a MDP consists of a state-transition function T and a reward function R.

$$T : S \times A \rightarrow \mathit{\Pi}(S)$$
$$R : S \times A \rightarrow \mathbb{R}$$

The state-transition function gives for each state of the world $s \in S$ and each action $a \in A$ a probability distribution $\mathit{\Pi}$ over all world states for the probability of ending in a certain state s' given the agent started in state s and executed action a. The reward function gives the expected immediate reward gained for an agent for taking each action $a \in A$ in each state $s \in S$.

The goal of the agent is to maximize the future reward it gets over an infinite amount of time. As this would require infinite computational resources we have to settle with alternative optimality criteria. Two of such criteria are used: the *finite-horizon* model and the *infinite-horizon discounted* model. An agent using the first should try to maximize its reward EU_k over a finite number of time steps k

$$EU_k = \sum_{t=0}^{k} R_t$$

where R_t is the reward received at time t. Problem of this model is one has to know in advance what value of k is appropriate. The *infinite-horizon discounted* model sums the rewards over the total lifetime of an agent but discounts them using discount factor $0 < \gamma < 1$. The agent should try to maximize its discounted rewards EDU

$$EDU = \sum_{t=0}^{\infty} \gamma^t R_t.$$

The idea is that rewards received now have more value for the agent than ones gained in a more distant (and more uncertain) future. An infinite lifetime is considered but the discount factor ensures the sum is finite.

In a MDP setting an agent has a policy $\pi : S \to A$ which specifies which action to take in each state $s \in S$.

If a Markov process cannot be observed with perfect certainty the decision problem turns into a partially observable Markov decision process (POMDP) [29]. This means that by looking at our sensor measurements we cannot unambiguously determine the state of the Markov process. A POMDP extends the Markov decision process with a set of possible observations Ω and an observation function O resulting in a tuple $\langle S, A, T, R, \Omega, O \rangle$.

Ω is a finite set of observations the agent can experience of its world

$O : S \times A \to \Pi(\Omega)$

The observation function gives for each state and each action a probability distribution over all possible observations for the probability of making a certain observation given that the agent took action a and arrived in state s'.

We have briefly summarized the theory on Markov decision processes and will now describe how it can be put into practice in the RoboCup middle-size league domain.

5.2 Applying Markov decision processes to RoboCup

From the high level point of view of action selection and team coordination we regard the RoboCup domain as being a Markov decision process. A robot's current perfect view of the world combined with its role is sufficient for making decisions. Opponents are not being modeled as agents since this would be very difficult and violate the Markov property, as they react on you and vice versa. We assume they are static obstacles which have a certain probability to disrupt a robot's actions.

Compiling an accurate state-transition function for the RoboCup middle-size league domain is intractable given its continuous nature and resulting infinite number of states and possible actions. A real state-transition function should return a probability distribution over all the possible outcomes of an action which would require an large amount of time spent testing a real robot. We have simplified the function considerably by assuming each action in a certain state can have only one result, for which we dynamically estimate the probability the action will succeed.

The immediate reward of an action can be a bit unclear in soccer. The only real reward you get is at the end of a match, which is positive if your team scored more goals than the other team and negative if they scored more. This will clearly not suffice so we tried to find several characteristics of a soccer game describing the essence of it. We have designed our reward function as follows: estimate the desirability of the current world state by looking at several soccer heuristics, simulate the action on this world state to obtain a new world state, estimate the desirability of the new world state, and the difference between the two estimates is the reward.

As the robot soccer domain is very dynamic the *infinite-horizon discounted* model seems like the optimality criterion of choice. The lower the value of γ is chosen the more rewards in the longer future are discounted, so a low γ is appropriate.

RoboCup middle-size league is a partially observable domain, since one robot can not view the entire field and even if it could it would never have complete certainty about the objects on the field. We have not yet designed an observation function as this would require great effort so we assume the domain is completely observable by using the observations of teammates while ignoring the uncertainties in them. Even when the robot is in relative mode (it does not know its own position with enough certainty) we regard the domain as observable and limit the state space to the observable states. The action space has also shrunk since some actions like Move to an absolute target position (x, y) don't make sense if you don't know your starting position.

First we summarize our algorithm: it starts by generating a number of possible actions and their parameters with respect to its current role and world model. For each of these actions the probability of success is estimated. The impact on the world of an action and the expected concurrent actions of teammates and enemy players is evaluated. Now the action with the maximum expected utility can be chosen and the process can be repeated to look further ahead.

5.3 Defining a robot's state

A straightforward way to characterize a robot's state would be to start with its world model. A robot's view of the world consists of the position, heading and speed of all moving objects with the uncertainties of this data combined with information about the dimensions of the field. From the latter the position of static objects like the goals can be deduced. Drawback of including the complete world model in the robot's state is its high number of dimensions which leads to a great number of possible states to search.

We have reduced the number of states by incorporating domain dependent knowledge: four criteria based on real-life soccer experience by which a soccer game is evaluated. Together they describe the essence of a soccer game and form a measure how good a certain situation is. These four criteria will be explained in more detail in section 5.6 where they are used to evaluate the desirability of a certain state. Since some of these attributes are continuous however the space is still infinite. The robot's state consists of the following attributes:

- Whether the ball is in one of the goals. A three valued attribute: ball is in our goal, in their goal or in none of the goals.
- Ball possession. Which robot, if any, controls the ball? Also three valued: a robot in our team has ball control, a robot in their team has ball possession or none of the teams controls the ball.
- Role evaluation. Continuous attribute describing how well suited a robot is for its current role.
- Strategic positioning. Continuous measure of how desirable the robot's current location and heading are considering both moving objects and static ones like the wall.

The robot's state is not complete without certain internal parameters:

- The current distribution of roles among the field players, particularly relevant is the robot's own role.
- Whether the robot is in absolute of relative mode.
- The type of action the robot is currently executing[1]. Relevant in the sense that a continuation of a robot's current action might be preferable over a change in the type of action. For instance, when a robot is dribbling with the ball it should keep on doing that until it can take a decent shot at goal.

Not all seven components of the state definition are available at all times. When the robot is in relative mode the role evaluation and strategic positioning can not be computed and the robot will have to base its decision on the two remaining soccer heuristics, which will lead to a deterioration of the coordinated team play. Especially the robots in defending, non ball-oriented roles will perform badly since they are supposed to be at a specific area, which is hard if you don't know where you are.

5.4 Actions to consider

In a real world system such as RoboCup the number of possible actions is large, even infinite if one uses continuous variables. An action has a certain type and a number of arguments which differ per type. Table 5.1 repeats the list the

[1]This turns our algorithm into a first order Markov decision process which is converted to a zero order one at the cost of slightly expanding the state space.

Type	Argument(s)
Turn	absolute angle φ, relative angle φ, object (*ball*, *ourGoal* or *theirGoal*)
Move	absolute (position, speed) (x, y, θ, v)
Dribble	absolute (position, speed) (x, y, θ, v), object (*ourGoal* or *theirGoal*)
Shoot	absolute angle φ
Seek	object (*ball*, *ourGoal* or *theirGoal*)
Chase	object (*ball*, *ourGoal* or *theirGoal*)

Table 5.1: Overview of actions offered by the Player skills module.

actions the Player skills module [3] can offer as described in section 2.2.2, which forms the pool of actions from which the set of actions under consideration can be chosen. The Move and Dribble actions can take arbitrary target positions as arguments which enlarges the number of possible actions significantly. The number of actions under consideration at each step of the Markov decision process is the branching factor of the search, so it is clear their number should be limited as there is only a finite amount of time available for choosing the next action. If the entire action planning process takes too long to complete the information its decision is based upon will already be obsolete. Only a limited number of values for the continuous variables can be evaluated.

A robot's role should define the actions it considers, for instance a defending robot will strictly speaking not have to take into account Dribble or Shoot actions as it will not control the ball. To be on the safe side however we do include some actions which require ball possession with roles which exclude ball control to be able to handle temporal discrepancies between the real world and your assigned role. In the defending robot example it might profitable to just kick the ball across the field the moment the robot gains control over it instead of trying to dribble such a long way.

We define the set of actions A as

$$A = A_0 \cup A_{\text{role}}$$

where A_0 is a set of action relevant for all roles and A_{role} a set of actions specific to each role.

A_0 consists of the following actions:

Turn Turn actions specifying turns to an absolute heading φ in eight directions plus one turn to face the ball. Figure 5.1 visualizes the default Turn actions where the arrows denote the direction the robot will be facing. The figure shows the starting position of the robot as the circle and shows its initial heading by drawing the two ball handlers. To put everything in perspective the outline of half a soccer field is drawn.

Move Move actions lead to four (x, y) positions equally distributed on a circle whose center is the robot. The radius of the circle can be specified and

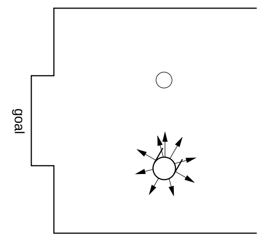


Figure 5.1: The nine Turn actions.

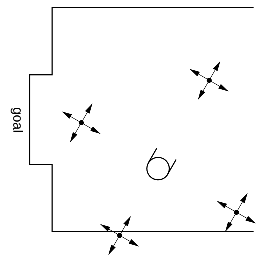


Figure 5.2: The sixteen Move and sixteen Dribble actions.

represents the time you want to look ahead. At each position four directions θ are being considered and the speed v is set to zero. The idea is that before a robot reaches the point already a new Move command to another point further away keeps it going. The sixteen Move commands are shown in figure 5.2, where the dots denote the four positions and at each dot the arrows the four headings.

Dribble Dribble actions have the same arguments as the Move actions. So figure 5.2 also shows the default Dribble actions. Note however that these two types of actions are very differently executed by the Players skills module. A robot has to drive much more careful and slower when controlling ball, especially when trying to turn with it. Dribble is an action which usually does not last very long since the robot easily loses the ball.

Shoot Shoot actions are defined on a beam spread of 135° pointing forward. Nine angles φ are being considered of which the fifth one is just kicking straight ahead. Figure 5.3 shows the Shoot actions which follow curved paths since the robot can only kick straight ahead and the Player skills module will drive a short curved path before kicking.

Seek A Seek the ball action is added to find the ball. The Seek action involves the robot turning around its own axis until it has found the desired object as the right robot in figure 5.4 demonstrates.

Chase A Chase the ball action is added to obtain the ball if necessary. The left robot in figure 5.4 approximates the effect of a Chase action.

Note that also “illegal” actions like Moves to position out of the field (e.g. in figure 5.2) or Shoots when the robot does not have control over the ball are not excluded beforehand. These actions will be eliminated afterward in the process by assigning a very low or zero probability of success (e.g. shooting not in the direction of the goal) or giving them a very bad evaluation (e.g. moving outside the field).

Each role adds to the standard set of actions A_0 a set of actions A_{role} relevant for this role. For instance the offensive roles include the powerful Dribble to *theirGoal* which should be used to approach the opponent's goal after which a nice shot can finish the attack. For the defending roles Move actions which block the path of an incoming opponent attacker could be added[1]. The set of actions

[1]...but these have not yet been implemented.

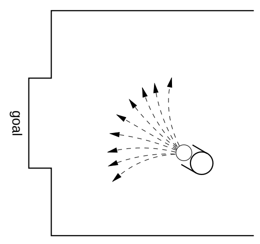


Figure 5.3: The nine Shoot actions.

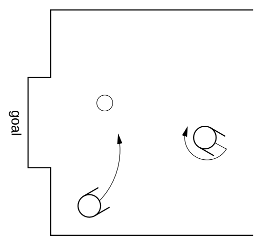


Figure 5.4: The Chase action (left robot) and the Seek action.

Role	Extra actions (A_{role})
PassiveDefend	Move actions at double radius
ActiveDefend	Move actions at double radius
InterceptBall	Move actions at double radius, Dribble to *theirGoal*
AttackWithBall	Dribble actions at double radius, Dribble to *theirGoal*, two × number of Shoot actions
AttackWithoutBall	Move actions at double radius, Dribble to *theirGoal*

Table 5.2: Actions under consideration for each role in addition to the standard repertoire.

for each role can be found in table 5.2. Each role excluding ball possession adds a number of Move actions. Their target positions lie on a circle with double radius relative to the circle radius used for the default Move actions (figure 5.2). Role AttackWithBall adds Dribble actions to the same positions and doubles the number of Shoot actions under consideration.

When a robot is in relative mode many of the actions in A don't make sense. If a robot is not sure of its own position how can it plan a path to an absolute position on the field? Relative mode requires a robot to trust only information it receives directly from its own vision system. This dramatically decreases the number of options a robot has. The default set of actions for relative mode A'_0 only contains the following actions: Turn to ball, Seek ball, Seek *ourGoal*, Seek *theirGoal*, Chase ball, Chase *ourGoal* and Chase *theirGoal*. The ball oriented roles AttackWithBall, AttackWithoutBall and InterceptBall have a set of actions specifically relevant to the role in relative mode A'_{role} consisting only of the action Dribble to *theirGoal*. For the defending roles A'_{role} is empty. No Shoot actions are available in relative mode, but the Player skills module will kick the ball if it sees a large portion of the enemy goal. It also remembers at which heading it last saw the both goals, so the Player skills module can still carry out actions like Dribble to *theirGoal* even if the robot does not know its own position by steering the robot in the direction it last saw *theirGoal*.

5.5 Moving from one state to another

Putting together a truly realistic state-transition function T for the RoboCup middle-size domain is not possible during the time allotted for this project and would not be desirable as only limited computing resources and time are available for search. We have therefore searched for ways to simplify the traditional state-transition function. Our function T does not return a probability distribution over all world states but only over one state, the one the robot is likely to end up in. This can be achieved by simply simulating the action a the robot in state s can take to obtain the new state s' and estimating how likely it is for this transition to succeed.

5.5.1 Simulating actions

It is not for the Team skills module to include a complete and realistic simulator of the RoboCup middle-size environment and the Clockwork Orange low-level software as this would require considerable effort. Such a simulator was not available at the time but there has been done work since on a simulator of the Clockwork Orange low-level software and the robot hardware [5]. The simulation should be fast and sufficient for our needs at the team behavior level for which it is not necessary to have a detailed physical model of the world and an accurate mechanical model of a Nomad Scout. We want to predict the approximate effect an action or sequence of actions has on the world so we simulate them on an abstract level.

Our simulation simply assumes that actions always succeed: if you want to move to a certain position you are transferred instantaneously without having to plan and drive a path between starting and target position. At the moment all other robots in the field are considered stationary, even though one knows the speed and heading of a teammate this data is not available for the opponents. For the moment they are considered to be a constant factor which can interrupt your actions; the next section about probabilities will put this assumption into practice. If we were to take into account a non-trivial model of the opponents we would have to incorporate aspects from game theory which lie outside the scope of this thesis.

Description of the simulation of each action type:

Turn Turn actions are simulated by setting the robot's heading to the desired one.

Move Moves result in transferring the robot to the target position and heading.

Dribble Dribble actions are Moves while retaining ball possession. When dribbling to an object like *ourGoal* or *theirGoal* just move the robot a reasonable distance toward to the object and let the robot face it.

Shoot Shoot actions always deposit the ball in the enemy goal.

Seek Seeking the ball results in finding the ball. Seeking one of the goals is simulated by making the robot face the goal.

Chase Chasing the ball is a Move where the distance between robot and ball is decreased a reasonable amount but only if the estimated time to reach the ball is not too high. Chasing one of the goals results in moving closer to them and facing them.

In our view this abstraction in the simulator meets our needs, it predicts the next state with sufficient accuracy to estimate its probability of success.

5.5.2 Estimating probability of success

The hard part of simulating the actions is judging how likely an action is to succeed. There are several ways to obtain these numbers. The best way would be to define and execute a large number of real world tests to provide insight in what factors influence success and what their exact influence is. We do not have the time to run these tests nor do we think it would be a wise way of spending one's time. Another approach to finding the probabilities of success would be to run the same tests in simulation but an adequate simulator was not available. Even in simulation these tests would take a long time. Third solution to the problem, the one we have chosen, is to make some educated guesses based on our experience with the system and the Player skills module in particular.

To eliminate actions at this stage which we know to be undesirable most action types have certain conditions under which they are removed from the process. For instance if a robot controls the ball it should dribble not move and vice versa. These kind of "wrong" actions will also be given a low reward, but eliminating them at this point is more efficient. We will now describe the heuristics for estimating the probability of success of each action type:

Turn Turn actions succeed with a fixed probability of p_{ball}, which reflects the fact there are situations where a robot can't turn. This means the probability should be lower than one. For instance when a robot is very close to the wall or another robot the ball handlers can prevent turning.

Move Move success is set to p_{move} which reflects the possibility of an opponent blocking the robot's path or target position by the time it gets there. However if a robot controls the ball the action is removed.

Dribble Estimating Dribble success is where things become interesting. When the robot was currently already dribbling it should be encouraged to keep on going by setting the probability to 1. However, if the Players skills module has aborted the current Dribble action for some reason (e.g. it has lost the ball and sees no opportunity to regain it by itself in the near future) or another teammate is closer to the ball the robot should be discouraged to dribble and the action is removed.

If the robot's current action is not a Dribble action it should first whether it has control over the ball. If not Dribble can never succeed and the action is removed. Next factor is the argument of the Dribble. If the robot is contemplating dribbling to one of the goals the probability is set to $p_{\text{dribble},0}$ if it actually has the goal in sight and set to $p_{\text{dribble},1} < p_{\text{dribble},0}$ otherwise. Not seeing your target probably means turning with the ball which can be very tricky. Probability of success of dribbling to a certain position is only dependent on the time needed to reach that point. By making it depend on the time (obtained using a function similar to $\mathcal{T}$ of equation (4.1)) we incorporate the amount of turning needed to reach the desired location.

Shoot As with Dribble the first check should be if the robot has the ball; if not the action is removed. The probability of scoring is very low and we assume it depends on two factors: how wide is the gap the robot is trying to kick the ball in and how much does the robot have to turn with the ball to reach the desired shoot heading. The first factor is depicted in figure 5.5 which shows a robot on the right ready to take a shot and in front of the goal two defending robots. The gray areas are the shoot headings blocked by the obstacles and the arrow denotes the optimal shoot heading, which is defined as the heading pointing to the center of the widest gap on the goal line between the goal posts. The second factor denotes the influence of the maneuvering necessary to get the robot to face the desired heading on the result of the following kick.

Another reason for a defender to kick the ball is to get the ball across the field toward the opponents' goal[1]. Requirement for assigning a non-zero probability of success is that the robot is facing forward to the other team's goal to prevent a robot scoring an own goal. The probability such an action succeeds increases linearly with the x position (see figure 4.4) of the robot from $p_{\text{shoot},0} \ll 1$ on half way line to $p_{\text{shoot},1} > p_{\text{shoot},0}$ on its own goal line. On the opponent's half of the field the probability is $p_{\text{shoot},0}$: if no other actions are likely to have a positive results just kicking the ball in the right direction can be a good idea.

Seek Probability of success is set to p_{seek} if the robot does not know where the ball is and the action is removed if the ball position is known to the team.

Chase If the target object of the Chase action is one of the goals the probability of success depends on the visibility of the goals[2]. If the robot sees the goal with its own vision system the probability is set to $p_{\text{chase},0}$, if not is assumed to be $p_{\text{chase},1} < p_{\text{chase},0}$. However, just as with the Dribble action if the current action also was of the Chase type the probability is set to 1.

Chasing the ball depends on whether or not the robot is closest to the ball. If one of its teammates is closer to the ball (according to function $\mathcal{T}$ of equation (4.1)) the action is removed. If the robot was not already chasing the probability is set to $p_{\text{chase},2}$. If the robot is already check whether or not the Player skills module has aborted the current Chase action, if it has set the probability to $p_{\text{chase},3}$ else to 1. The idea is you want a robot to keep on chasing until it has obtained the ball or the Player skills module has decided it cannot complete the action.

The value of the probabilities introduced above during the RoboCup 2001 tournament are listed in table 5.3. They are the parameters one can tweak to influence the behavior of the individual robot. Especially the balance between p_{move} and the pair $p_{\text{chase},0}$, $p_{\text{chase},1}$ determines how soon a robot will try to chase the ball instead of moving to a certain position. If one would like to make the robots more trigger-happy, i.d. shoot more often the Shoot probabilities $p_{\text{shoot},0}$ and $p_{\text{shoot},1}$ can be raised, although it might be more appropriate to change the reward function.

[1]The effect of such action in the simulation would still be scoring a goal though.

[2]This is one of non-compliant uses of communication as noted in the first paragraph of section 2.3.2.

Probability	Value
p_{ball}	0.75
p_{move}	0.75
$p_{\text{dribble},0}$	0.75
$p_{\text{dribble},1}$	0.25
$p_{\text{shoot},0}$	0.1

Probability	Value
$p_{\text{shoot},1}$	0.4
p_{seek}	1
$p_{\text{chase},0}$	0.75
$p_{\text{chase},1}$	0.25
$p_{\text{chase},2}$	0.75
$p_{\text{chase},3}$	0.25

Table 5.3: Value of the probabilities of success used during the RoboCup 2001 tournament.

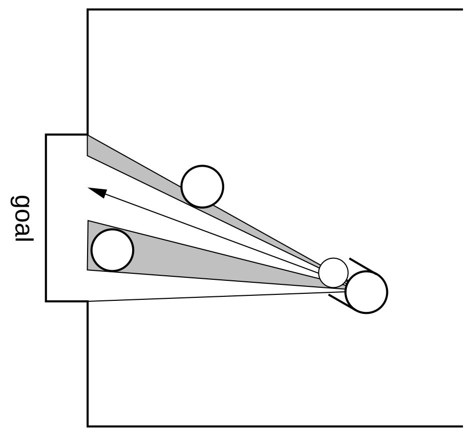


Figure 5.5: The optimal shoot angle for a given situation.

Weight	w_0	w_1	w_2	w_3
Value	4	1	1	1

Table 5.4: Value of the weights during the RoboCup 2001 tournament.

Using a truly realistic simulator one could even learn all probabilities, although for building such a simulator these probabilities are crucial. This would require a large number of real world tests.

5.6 Rewarding actions

After designing the state-transition function which tells us which state will be the result of executing a certain action we are ready to design the reward function R. We have to find a way to be able to tell how good a certain action is: to what extent will executing this action benefit the team? A straightforward solution to this problem would be executing the action in simulation using the state-transition function and try to rate the resulting state. More specifically: try to rate the improvement or decline between the current state and the future state which the action leads to.

The evaluation measure for a state is based on the four soccer related criteria in the state definition: ball in goal, ball possession, role evaluation and strategic positioning. Each has a weight associated with it, since some things are more important in soccer than others. The evaluation measure $\mathcal{E}$ is a weighted sum normalized on $[-1, 1]$:

$$\mathcal{E}(\vec{p}) = \frac{w_0\mathcal{G} + w_1\mathcal{B} + w_2\mathcal{H}_{\text{role}}(\vec{p}) + w_3\mathcal{P}(\vec{p})}{\sum_i w_i}. \tag{5.1}$$

The value of the weights during RoboCup 2001 can be found in table 5.4.

First criterion $\mathcal{G}$ is a check whether the ball is in one of the goals. The score is 1 if we scored, -1 if the other team scored and 0 otherwise. This criterion obviously has the highest weight of all, scoring a goal is the goal of the game and avoiding a goal for your opponents is also very important.

Next criterion is the concept of ball possession. This score $\mathcal{B}$ is 1 when our team has the ball, -1 when one of the opponents has it and 0 if none of the teams seems to control the ball.

Third criterion is a role evaluation. We use the position based role evaluation mechanism $\mathcal{H}_{\text{role}}$ defined in equation (4.5) of section 4.4, not the utility functions themselves which can also take into account the time needed to reach the ball. This is not desirable, a robot's position with respect to the ball's position is part of the strategic positioning.

5.6.1 Strategic positioning

The last criterion $\mathcal{P}$ concerns strategic positioning. This means staying clear of static obstacles like the walls, corners and goal areas. Walls are considered bad as the robot can bump into them and maneuvering (especially with the ball) becomes difficult. The same argument applies to the corners, but to a greater extent as turning near a corner is even more difficult. RoboCup regulations (see section 1.4) forbid a robot to spent more than a specified amount of time in one of the goal areas so they are also not a good spot to be at. An additional aspect is that the robot is either obstructing its own goalkeeper or potentially blocking a shot of a teammate.

Robots are also obstacles since two robots cannot occupy the same amount of space at the same time. Teammates are considered as much more undesirable obstacles than opponent robots since a robot should not obstruct a teammate but should not be afraid to get close to an opponent for instance in order to snatch the ball away. An important objective of team play is to not have several players crowd around a single point, say the ball, but let them spread themselves out over the field.

Strategic positioning not only means minding the obstacles and other bad influences in the field but also taking into account desirable places to go to. Currently the only desirable location is the one where the ball's at. The ball has more positive influence than any of the obstacles and its influence carries farther across the field as it is the most important object in the game.

A strategic position alone is not enough, the heading should also be taken into account. Imagine a defending robot standing just in front of its own goal area but facing its own goal, it will never see an incoming attacker and be of little use. Therefore we have included the heading of robot in our strategic positioning. Two headings are preferable in soccer: facing the ball and facing the opponent goal.

All these positive and negative influences can be easily viewed as attractors and repellers defined in section 4.4.2. Function $\mathcal{P}'$, similar to $\mathcal{H}'_{\text{role}}$ sums the attractors, repellers and the influence of the robot's heading as follows

$$\mathcal{P}'(\vec{p}) = \sum_i \mathcal{F}_{att}(\vec{p}, a_i) + \sum_j \mathcal{F}_{rep}(\vec{p}, r_j) + \mathcal{A}(\theta) \tag{5.2}$$

where $\mathcal{A}$ denotes the effect of heading θ on the strategic positioning. We will repeat equations (4.9) and (4.10):

$$\mathcal{F}_{att}(\vec{p}, a) = \min\left(\frac{\xi_a\, U(a)}{D(\vec{p}, P(a))}, U(a)\right), \text{ where } U(a) > 0 \tag{5.3}$$

$$\mathcal{F}_{rep}(\vec{p}, r) = \max\left(\frac{\xi_r\, U(r)}{D(\vec{p}, P(r))}, U(r)\right), \text{ where } U(r) < 0. \tag{5.4}$$

We have to map $\mathcal{P}'$ to a monotonically ascending function which ranges on $[-1, 1]$ to make sure the evaluation measure $\mathcal{E}$ does not exceed this range. We have chosen the simplest way of doing this:

$$\mathcal{P}_{\text{role}}(\vec{p}) = \frac{\mathcal{P}'_{\text{role}}(\vec{p})}{c_2}, \tag{5.5}$$

where c_2 is a positive constant large enough to ensure $\mathcal{P}_{\text{role}}(\vec{p})$ ranges on $[-1, 1]$

$$c_2 \geq \max_{\text{role}, \vec{p}} \mathcal{P}'_{\text{role}}(\vec{p}). \tag{5.6}$$

Attractor	Strength U
Ball	12

Repeller	Strength U
Corner	-6
Border	-3
Goal area	-3
Own robot	-10
Opponent robot	-6

Table 5.5: Strength of the attractor and repellers during the RoboCup 2001 tournament.

The strengths of the attractor and repellers during RoboCup 2001 can be found in table 5.5. Important parameters are the strength of the opponent robot repellers and the strength of the ball attractor. If the opponent robot repels more or equal to the attracting force of the ball a robot will never dare to steal the ball away from an opponent. However if the opponent does not repel at all accidents might happen and when a robot controls the ball it might dribble to a spot too close to an opponent.

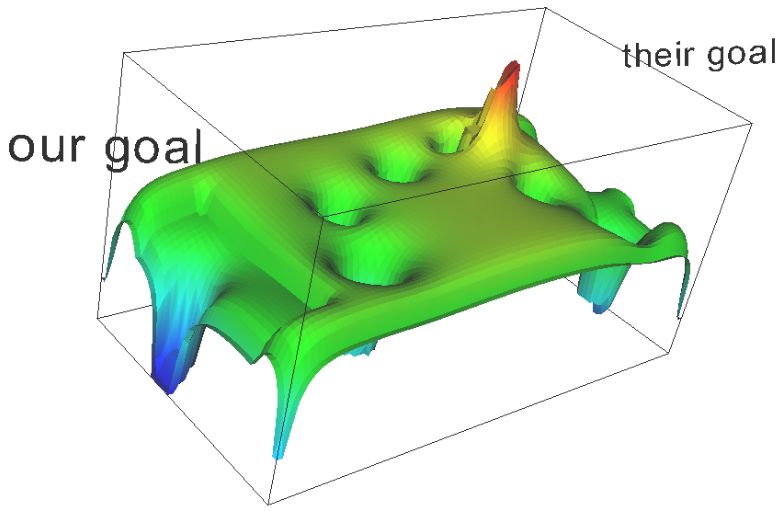
A visualization of a typical situation is shown in figure 5.6. The peak in the visualization is the location of the ball, which is not a complete circle but looks deteriorated on one side because of the opponent robot standing next to it. The difference between own robots and opponent robots can be clearly seen by comparing the influence of both goalkeepers: the negative peak of the opponent goalkeeper is less deep as the other one.

5.7 Choosing the next action

Now that we have discussed all the components of a MDP for action selection in the RoboCup middle-size league we can finish by describing the algorithm used to find the best next action. This algorithm is the policy of the robot: it maps states to actions.

The policy for a robot is simple: simulate each of the actions in set A on the current world state, evaluate the reward of the resulting world state, estimate the probability each action will succeed and calculate the utility of each action. The utility is the product of the reward and the probability of success. For each resulting world state we can repeat this process, updating the utility of the end state by multiplying the probabilities of success of the actions in the sequence leading to this end state with the reward of the end state. This is an application of the *finite horizon* criterion, which means you have to choose an appropriate horizon, which in our case depends on the amount of time already spent searching.

So our algorithm is bound by search time: if it takes too long to choose an action the decision will be based on obsolete information and thus irrelevant. Therefore we only take a small number of actions into account at each depth in the search: the n action sequences with the highest utility. Such a search method can be classified as beam search [54], where n is the size of the beam. During the RoboCup 2001 tournament n was set to 1 and the maximum search



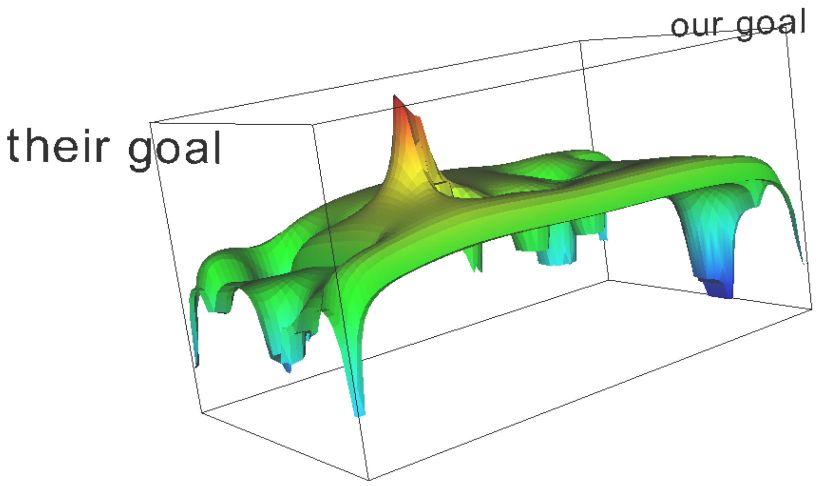


Figure 5.6: Visualization of the strategic positioning showing the static obstacles (corners, walls and goal areas), the ball and a number of robots. Both figures show the same situation viewed from two different angles.

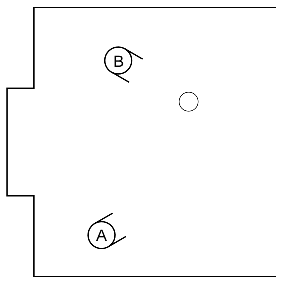


Figure 5.7: Situation in which the robots have to decide who will try to obtain possession of the ball.

depth to 1 as searching deeper was not tested at the moment.

Sharing the world model enables a robot to reason about its teammates in the same manner as it reasons about itself. Before choosing its own action a robot predicts the next action of its teammates and their effects on the world state. The robot chooses its action based on this altered world state, thus taking into account the actions teammates will probably execute. The beneficiary effects of this kind of reasoning can be illustrated using figure 5.7. Suppose robot A wants to choose a new action and both robots are in role InterceptBall. Robot A starts with predicting what action robot B will take. Robot A assumes that robot B knows it is closest to the ball. So robot A can predict robot B will choose to try to obtain control of the ball by chasing. The effect will be that robot B obtains ball possession. So in the world state robot A considers for choosing its own action the team already has ball possession and it is not necessary for robot A to chase the ball, but it can make itself more useful by moving toward the opponent goal.

The problem of the kind of I-know-you-know-I-know reasoning in our algorithm is that it requires the internal state of each robot and the world state to be common knowledge. We assume this to be the case, but Halpern and Moses have shown common knowledge is unattainable in practical distributed systems [23]. One has to make sure the system cannot end up in the state that robot A assumes robot B will chase the ball and vice versa, resulting in the undesirable situation that no robot will go after the ball. By having each robot check which robot is closest to the ball this problem is avoided. Such a problem could however occur when the potential conflict is not about chasing the ball but two robots expecting each other to move to a specific strategic position, which could result in none of them moving there. However, as long as the two robots have different roles these problems will be limited as they have different objectives and rewards.

Chapter 6

Results

To evaluate our approaches in team coordination and action planning we will present in this chapter results obtained from matches at the RoboCup 2001 tournament. More detailed information can be found in appendix A as well as some results obtained from other matches which only confirm the results presented in this chapter.

One of the problems of designing, implementing and testing a multi-agent system is the fact you need to have multiple agents available. In our case this means multiple robots, eight of them are needed to play a real game with two identical teams. In such a setting one could keep the one team fixed while experimenting with parameters or methods in the other team. For simple testing purposes however three field players will be sufficient, they can practice against static black obstacles. The University of Amsterdam owns two field players and a goalkeeper, so one of the field players of the Delft University of Technology is needed for tests. All robots and developers are brought together some time before a tournament, but during this period every software module needs to be tested. The lack of a realistic simulator has limited our opportunity to test the Team skills module and do experiments to determine good values for several parameters.

The results presented in this chapter are all obtained at the RoboCup 2001 tournament in August 2001 as that was the only opportunity to play against real opponents. Clockwork Orange also participated in the German Open 2001 tournament in June 2001, but at that stage the system was not stable yet and no usable results were obtained. Participation was still very useful as a lot of issues popped up during preparation and tournament which could be fixed for the world cup in August.

6.1 RoboCup 2001

Clockwork Orange successfully participated in the RoboCup 2001 tournament in Seattle, USA. The middle-size league consisted of eighteen teams from all over the world[1]. The team made it to the quarterfinals where it was defeated by CS

[1]Number of teams per participating country: Germany 5, Japan 3, Italy 3, Portugal 2, USA 2, Netherlands 1, Iran 1 and France 1.

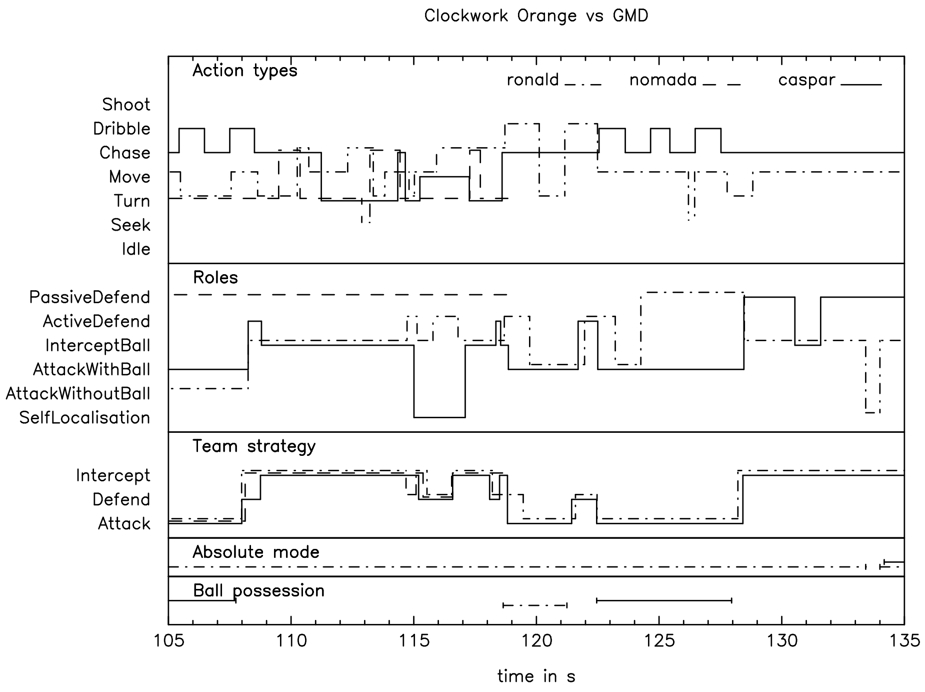


Figure 6.1: Visualization of action type, role distribution, team strategy, absolute mode and ball possession during a 30 second period of the RoboCup 2001 game against GMD. The x-axis displays the wall clock time in seconds since the start of the game. The robots involved are Ronald, Caspar and Nomada (removed from the game 118.88 seconds after game start). Their role, team strategy and action type of their current action are shown on the y-axis. When one of them has ball possession a line is drawn in the bottom part of the graph (as perceived by the team coordination mechanism, not the action selection) and when they are in absolute mode another line is drawn.

Freiburg. In total seven games were played, resulting in three victories, one draw and three losses. Results regarding team coordination will be presented from the four and a half last games. During the half-time interval in the game against Trackies the team coordination implementation was frozen for the rest of the tournament. Sadly, logs from the first half have been lost. The outcomes of all matches were:

Clockwork Orange	vs. ISocRob	3-1
	vs. CoPS Stuttgart	2-2
	vs. Trackies	0-8
	vs. Fun2maS	5-0
	vs. Artisti Veneti	3-0
	vs. GMD	1-4
	vs. CS Freiburg	0-4

6.2 Team play

As an illustration of the performance of the system we will discuss a 30 second fragment from the match against GMD demonstrating the effect of the Team skills module on team play, shown in figure 6.1. This fragment gives an example of how our coordination mechanism works under good conditions: the world model is consistent at an acceptable degree. The field playing robots in our team are called Caspar, Ronald, Nomada and Nomado and Nomadi is our designated goalkeeper. One of the field players is a substitute. Some of the claims made in this discussion cannot be derived directly from figure 6.1 but have been made on basis of investigation of the complete log-files.

At the start of the fragment Caspar has control over the ball and each robot believes the team strategy is *attack*, as they should. The team has correctly assigned the AttackWithBall role to Caspar and the AttackWithoutBall role to Ronald while Nomada is standing close to the own goal area in role PassiveDefend.

Caspar is alternating between Dribble and Chase which is a pattern that can be observed throughout all matches. During Dribble a robot can easily lose control over the ball, in which the Player skills module promptly notifies the Team skills module to take appropriate measures: switch to Chase for instance. Nomada is positioned at the left of its own goal area border while the goalkeeper Nomadi is standing inside the goal area but on the right side. So Nomada does not block the goalkeeper sight and is executing action “Turn to ball”, which is a good defensive action if the robot has reached a defensive position and has to wait for things to come.

At nearly three seconds after the start of this fragment Caspar permanently loses the ball as team coordination gets told by the world model. The team switches to team strategy *intercept* and the two attacking robots switch to role InterceptBall. Between $t = 115$ and $t = 118$ there is some confusion whether or not the opponent controls the ball which leads to oscillations between team strategies *intercept* and *defend*. Caspar switches for a couple of seconds to role SelfLocalization until it sees the ball and goes back to role InterceptBall.

At about the same time as Ronald gains possession of the ball Nomada is neatly shut down and removed from the game. This means the team has lost its defender and one of the other two should fill the gap. Ronald does so by switching to role PassiveDefend at $t = 124$, after having been in error for a short while: it was at the same time as Caspar in role AttackWithBall although it was really Caspar which controlled the ball. After it has lost the ball ($t = 128$) the team switches to *intercept* and Caspar takes on the role of PassiveDefender while Ronald switches to InterceptBall.

6.3 Distribution of team strategies over roles

To check whether or not the role distribution has functioned properly a simple sanity check is to investigate which roles have been assigned in which team strategy. For the match against CS Freiburg table 6.1 visualizes the amount of time spent in each role relative to the total spent in each team strategy. The data has been normalized in such a way that each team strategy was used for the same duration. The abbreviations used for each role can be found on page 36.

	PD	AD	IB	AWB	AWoB	SL
Attack	●	·	•	⬤	⬤	●
Defend	●	⬤	•	•	·	●
Intercept	●	•	⬤	·	·	●

Table 6.1: The portion of time spent in a certain role while in a certain team strategy. The surface area of the circles is proportional to the amount of time spent in the role relative to the total time spent in the team strategy, but the data has been normalized to make each team strategy equally frequent. The data presented is a combination of three robots gathered from the RoboCup 2001 quarter final against CS Freiburg.

Match	Attack	Defend	Intercept	Total
Trackies (2nd half)	•	•	⬤	1170 s
Fun2maS	⬤	•	●	2539 s
Artisti Veneti	●	●	⬤	2811 s
GMD	●	●	⬤	2588 s
CS Freiburg	●	●	⬤	3147 s

Table 6.2: Comparison of used team strategies during the four and a half RoboCup 2001 matches under investigation. The surface area of the circles is proportional to the amount of time spent in the team strategy type relative to the total time of that match. The data from the three field playing robots is summed.

The table clearly shows that roles exclusive for a certain team strategy (all except PassiveDefend and SelfLocalization) only are assigned in the correct team strategy, except for some minor violations. They are probably due to temporal inconsistencies when switching team strategy and roles. PassiveDefend and SelfLocalization are equally distributed over the three team strategies as one would expect: each team strategy contains an opening for PassiveDefend and the assignment of role SelfLocalization is independent of team strategy.

Results from the other three and a half matches under investigation can be found in tables A.1 through A.4. Each table looks almost identical to one of the quarter final, only the distribution of PassiveDefend over the team strategies varies.

6.4 Team strategy

As table 6.2 shows team strategy *intercept* is the most common in all but one match. The exception is the game against Fun2maS in which *attack* is dominant. This seems consistent with the outcome of the match. The outcome of the match

	PD	AD	IB	AWB	AWoB	SL	Total
caspar	●	•	●	•	·	•	1084.18 s
nomada	●	•	●	•	·	●	1010.96 s
ronald	●	•	●	•	·	•	1044.07 s
all	●	•	●	•	·	•	3139 s

Table 6.3: The portion of time spent in a certain role for each robot and all three combined. The surface area of the circles is proportional to the amount of time spent in the role relative to the total time, shown in the right column. The data presented is from the RoboCup 2001 game against CS Freiburg.

against Artisti Veneti was also positive, but one of the three goals was in fact an own goal.

6.5 Assignment of roles

From table 6.3 we can see that the AttackWithoutBall role has been assigned only a small amount of time. It is the least important role of team strategy *attack* which means it will only be used if there are three field players active and one of them has ball possession. This is not a likely situation since during matches there are periods in which one or more robots do not know their own position accurately enough. Such a robot cannot communicate its own position to teammates and the teammates will not always assign a role to the robot in question. For instance, if all robots are in relative mode and one of them controls the ball each of them will assign the AttackWithBall role to itself. Each robot thinks it is the only robot in the team and the most important role in team strategy *attack* is AttackWithBall.

InterceptBall and PassiveDefend are the more popular roles which seems correct. Team strategy *intercept* is dominant during most matches (see table 6.2) and InterceptBall is its primary role. In each team strategy there is a slot for PassiveDefend which explains the relatively large amount of time spent in this role.

For the other matches these results can be found in the role distribution sub-tables of tables A.13 through A.16.

6.6 Distribution of action types over roles

To see the influence of a robot's role on its action selection we have included table 6.4. It shows what type of actions have been chosen while the robot was in a certain role during the quarter final. Move and Chase are the overall most popular action types. Even in PassiveDefend Chase is used perhaps more than one would expect. The nature of the AttackWithoutBall role is clearly

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
PD	●	●	·	·	●	●	•	886.83 s
AD	•	●	·	·	•	⬤	·	339.02 s
IB	•	●	·	·	●	⬤	·	1097.96 s
AWB	•	•	●	·	•	●	·	440.66 s
AWoB	●	⬤			·	•		79.49 s
SL	•	⬤						295.25 s

Table 6.4: The portion of time spent executing an action of a certain action type while in a certain role. The surface area of the circles is proportional to the amount of time spent in executing the action type relative to the total time spent in the role. Right column shows total time of each role. The data presented is a combination of three robots gathered from the RoboCup 2001 game against CS Freiburg.

represented by the fact most of its actions involve Moving. In AttackWithBall Dribble and Chase are occasionally alternated with Shooting.

For the other matches these results can be found in tables A.5 trough A.8, which show a similar picture.

6.7 Distribution of roles over action types

The same data used for table 6.4 can also be presented the other way around by looking at in which role which types of have been taken as done in table 6.5. As was to be expected by far the most Dribbling and Shooting has been done in AttackWithBall, although PassiveDefending robots also sometimes kick the ball away. The other action types are rather equally distributed if one takes into account the high amount of time which has been spent in PassiveDefend or InterceptBall. For the other matches similar pictures can be found in tables A.9 trough A.12.

6.8 Self localization performance

An important aspect of the system for the Team skills module is the performance of the Vision self localization. To illustrate the performance of the self localization mechanism during RoboCup 2001 we have plotted in figure 6.2 each position at which the world model received a position update from the self localization. The self localization performance of the other matches has been plotted in figures A.1 through A.4. A more detailed discussion on the performance of the self localization mechanism can be found in [4].

These events were not directly logged but could be deduced from the log of the world model (as logged by the Team skills module). When the uncertainty in x, y or θ decreases the only explanation is that the World model has received a position update. The only times would not be the result of the self localization

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle
PD	⬤	●	•	●	●	●	⬤
AD	•	•	•	•	•	●	•
IB	●	●	●	•	⬤	⬤	•
AWB	●	•	⬤	⬤	•	●	•
AWoB	•	•			·	·	
SL	•	●					
	419.46	1010.93	164.20	10.98	382.45	1133.97	17.22

Table 6.5: The portion of time spent in a certain role while executing an action of a certain action type. The surface area of the circles is proportional to the amount of time spent in the role relative to the total time executing an action of the action type. Bottom row shows total time in seconds spent executing an action of each action type. The data presented is a combination of three robots gathered from the RoboCup 2001 game against CS Freiburg.

found in

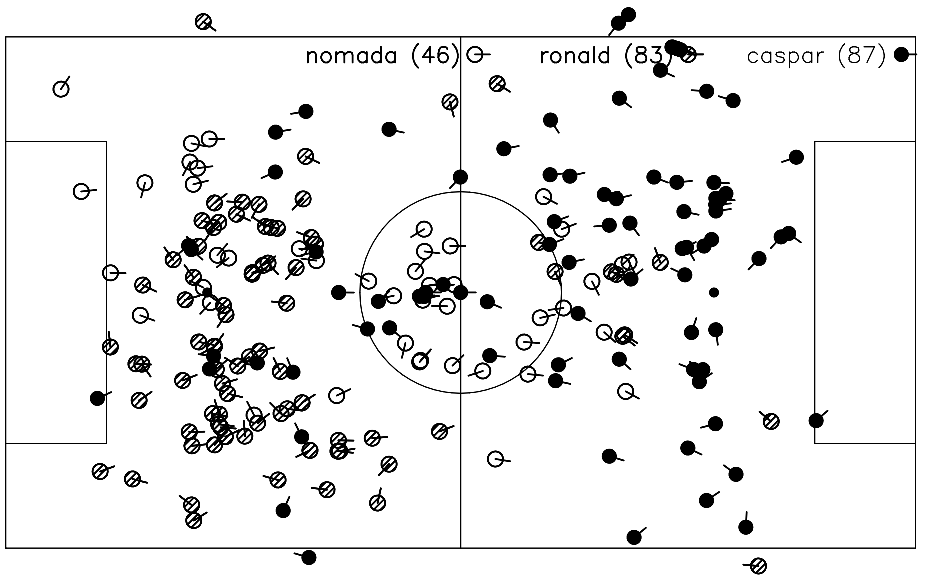


Figure 6.2: The new robot positions with orientation the World model got from the Vision based self localization. Between parentheses the number of updates during this match for each robot. The data presented was gathered during the RoboCup 2001 match against CS Freiburg.

Match	Absolute mode	Relative mode	Total
Trackies (2nd half)	●	●	1180 s
Fun2maS	●	●	2500 s
Artisti Veneti	●	●	2795 s
GMD	●	●	2596 s
CS Freiburg	●	●	3151 s

Table 6.6: Comparison of time spent in absolute or relative mode during the four and a half RoboCup 2001 matches under investigation. The surface area of the circles is proportional to the amount of time spent in the mode type relative to the total time of that match. The data from the three field playing robots is summed.

system would be at the (re)start of the game, when an operator is allowed to set a robot's position. This will not account for much noise as the game is never restarted more than let's say ten times. These manual resets will also only occur on the robot starting positions, clearly identifiable in figure A.1 on page 83: Ronald started in the right defensive position, Nomado in the left defensive spot and Caspar would begin at the kickoff spot (it took a lot of kickoffs during that particular match).

Table 6.6 gives more complete idea of the performance of the self localization mechanism, it shows what portion of time the Team skills module has spent in absolute mode. The amount and frequency of position updates coming from the self localization mechanism directly influence the amount of time spent in absolute mode. During these four and a half matches 49% percent of the time was spent in absolute mode.

6.9 Discussion

Figure 6.1 shows the good performance of our team coordination mechanism under good conditions, but even then inconsistencies sometimes occur. For instance at $t = 131$ Caspar briefly switches to role InterceptBall when it shouldn't. The reason is that Caspar's world model has not received a position update from Ronald's world model for a while, as Ronald has no estimate of its own position that is accurate enough to communicate. From Caspar's point of view it is the only active member of the team and it should thus fulfill the most important role in team strategy *intercept*: InterceptBall. These kind of problems are common and should be properly dealt with.

As table 6.2 shows the total amount of time spent in team strategy *attack* is rather low, only 28% of the time (almost three and a half hours). This would seem to confirm our intuition stated in section 3.5 that extending role-based approaches with a global team strategy makes sense, although we do not have the resources (two complete teams) to confirm this. In RoboCup simulation league however these lines of reasoning are common, for instance see [42].

Most of the time (53%) is spent in *intercept*, resulting from the fact that it is hard to detect whether the other team controls the ball. This would call for a more sophisticated way of determining the applicable team strategy than just looking at ball possession alone. Ball possession is a good first indication but other concepts like ball position on the field or the relative distances of teammates and opponents can be added. This would also reduce the number of oscillations caused by the possibly rapid changes in ball possession.

From table 6.3 we can deduce that all roles except AttackWithoutBall are commonly used during matches. However it is hard to draw conclusions regarding the number and type of roles need to be available for playing a good game of robot soccer without the availability of realistic simulator of the team and environment. From tables 6.4 and 6.5 we can see that a role significantly influences the types of actions chosen. They don't give the complete picture as action type is not always characteristic for the intention of the action. Although Chase actions are almost exclusively offensive (as they normally chase the ball or the opponent goal, not your own goal), Move actions for instance can have an offensive or defensive purpose determined by the action's parameters: target position and heading.

The performance of the self localization mechanism is essential for the success of the team coordination and action planning system. Although figure 6.2 may look promising, manual browsing of the log files has shown that many of these updates are grouped in periods in which they occur at short intervals, while between the "good" periods a large amount of time can pass during which the odometry error rises. Such behavior can be easily explained: once the robot's position has been found the local method is able to keep track of it for a certain amount of time, during which new position updates adding little new information are sent to the world model.

Chapter 7

Conclusions

We have presented an approach for coordinating the actions of a team of real world soccer agents and empirical results obtained during an actual tournament. The results demonstrated the approach performed well under the assumption that the world models of each robot are reasonably consistent. The added notion of a team strategy seems justified although we have not been able to conduct experiments proving this. There is still a lot of work to be done on comparing our approach to other approaches and gaining more insight in how tweaking or learning parameters can improve performance.

Contributions of this thesis include extending the dynamic role distribution approach [2, 52] with a global team strategy. Our utility functions not only take into account the time a robot needs to reach the ball but also its position on the field. For the problem of choosing the next action we have designed a way to evaluate a world state by looking at four soccer heuristics: ball in goal, ball possession, role evaluation and strategic position. The latter two criteria are similar to potential field theory in the sense that attractors and repellers are used to represent good and bad influences on a soccer field. In order to be able to effectively search a continuous domain such as RoboCup middle-size league modeled as a Markov decision process we have chosen to discretize the action space based on the robot's role.

One of main objectives of RoboCup is to provide a unified testbed for a wide range of technologies to be integrated and examined. The problem of integrating several different fields of science in one team is the most present in the middle-size league. What you are testing by playing matches against each other is more than the sum of the scientific ideas incorporated in the team, you're testing also how well all modules are working together. The scientifically most interesting team does not have to win, so the two goals “do interesting research” and “build a winning team” can conflict at some point.

7.1 Future work

It is hard to derive conclusions from actual matches as they are big stochastic processes. The higher the mechanism you want to examine resides in the software architecture the more difficult it becomes to clearly separate the influences of each part of the system. This stresses the need for a realistic simulator in which you can try out the exact same situation as many times as you would like. For initial testing you can even leave out the noise to make it more deterministic.

An extension of the current method to determine the correct team strategy is desirable to improve performance and team coherence by reducing the number of oscillations. Interesting would also be to experiment with the roles associated with each team strategy: to add new ones or change their priorities.

Roles are currently defined as a set of actions A_{role}, a position based evaluation function $\mathcal{H}_{\text{role}}$ and a utility function $\mathcal{U}_{\text{role}}$. Each of these three elements can be further developed to improve a role and focus them more on the role's objective. The most promising opportunity seems to be adding actions to A_{role}: for instance adding Move actions with target positions that directly block an incoming attacker to $A_{\text{PassiveDefend}}$.

The strategic positioning mechanism described in section 5.6.1 could also be tailored for each role. One could imagine customizing the strengths of the attractors and repellers: the influence of the ball attractor for instance should be less strong for defensive roles while offensive roles could be made more aggressive by lowering the strength of the opponent robot repellers.

An interesting avenue of further research could also be including the concept of monitoring the state of the team as Jennings [24] proposed. Our current system does not contain such a mechanism for monitoring teammates, but it does respond to feedback from the Player skills module regarding success, failure or impossibility of own actions. However, if a robot is closest to the ball but is stuck behind an opponent robot (without realizing with) while trying to chase the ball all of the team members will rely on this one robot to obtain control of the ball. Such situations should be avoided by checking whether or not the action a robot executes has the desired effect as predicted (section 5.5.1).

Another challenge will be integrating Utrecht University's Pioneer 2 in the team coordination mechanism. Being a team heterogeneous in both hardware and software will pose new problems. The first one will be the sharing of the world models of both architectures.

Appendix A

RoboCup 2001 extended results

This appendix gives a more detailed report on the performance of Clockwork Orange during the RoboCup 2001 tournament. Not only will data be presented regarding the Team skills module but also results about the performance of Vision based self localization. Note that even though some of the results don't concern the Team skills module they have been deduced from the log-files of the Team skills module. The data comes from the same games as discussed in chapter 6: the four and a half last matches. The raw data used for the results in this appendix and in chapter 6 is presented in separate appendix B, which is available from the author upon request.

A.1 Distribution of team strategies over roles

Tables A.1 through A.4 show the distribution of roles over team strategies for the three and a half other games than the one against CS Freiburg.

A.2 Distribution of action types over roles

Tables A.5 through A.8 show the distribution of action types over roles for the three and a half other games than the one against CS Freiburg.

A.3 Distribution of roles over action types

Tables A.9 through A.12 show the distribution of roles over action types for the three and a half other games than the one against CS Freiburg.

A.4 General impression

Tables A.13 through A.17 summarize the results of our team coordination mechanism and action selection for the games listed above. They show the portion of time the team has spent in each team strategy, the portion of time each role had been assigned, the portion of time each robot has been executing an action of a certain action type, the status of the ball possession and how long each has

	PD	AD	IB	AWB	AWoB	SL
Attack	●	·	•	⬤	⬤	●
Defend	●	⬤	•	·	·	●
Intercept	●	•	⬤	·	·	●

Table A.1: Same type of table as table 6.1, only here data is presented from the RoboCup 2001 game against Trackies (second half).

	PD	AD	IB	AWB	AWoB	SL
Attack	●	·	·	⬤	⬤	●
Defend	●	⬤	•	•	•	●
Intercept	●	•	⬤	•	•	●

Table A.2: Same type of table as table 6.1, only here data is presented from the RoboCup 2001 game against Fun2maS.

been in absolute or relative mode. In the ball possession sub-tables the columns Me and Our team of each robot should sum to about the same size.

A.5 Vision based self localization

Figures A.1 through A.4 show the position updates from the Vision based self localization from the other three and a half games.

	PD	AD	IB	AWB	AWoB	SL
Attack	●	•	•	⬤	⬤	●
Defend	●	⬤	•	•	•	●
Intercept	●	•	⬤	•	·	●

Table A.3: Same type of table as table 6.1, only here data is presented from the RoboCup 2001 game against Artisti Veneti.

	PD	AD	IB	AWB	AWoB	SL
Attack	●	·	•	⬤	⬤	●
Defend	●	⬤	•	•	·	●
Intercept	●	•	⬤	·	·	●

Table A.4: Same type of table as table 6.1, only here data is presented from the RoboCup 2001 game against GMD.

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
PD	•	●	·		●	●	•	293.22 s
AD	•	●	·	·	•	●	·	134.59 s
IB	•	●	·	·	●	●	·	533.54 s
AWB	•	·	●	·	·	●		58.45 s
AWoB	•	⬤				·		30.66 s
SL	·	⬤						132.80 s

Table A.5: Same type of table as table 6.4, only here data is presented from the RoboCup 2001 game against Trackies (second half).

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
PD	●	●	·		●	●	·	928.12 s
AD	•	•	·	·	•	⬤		153.32 s
IB	•	●	·	·	•	●	·	466.70 s
AWB	•	•	●	·	•	●		628.31 s
AWoB	●	●	·		·	●		80.22 s
SL	·	⬤						249.07 s

Table A.6: Same type of table as table 6.4, only here data is presented from the RoboCup 2001 game against Fun2maS.

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
PD	●	●	·		●	●	·	752.68 s
AD	•	•	·	·	•	●	·	372.20 s
IB	•	•	·		●	●		1160.43 s
AWB	●	•	●	•	•	●	·	270.99 s
AWoB	●	●	·		•	•		81.84 s
SL	•	●						169.69 s

Table A.7: Same type of table as table 6.4, only here data is presented from the RoboCup 2001 game against Artisti Veneti.

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
PD	●	●	·		•	•	·	711.11 s
AD	•	●	•	·	·	●	·	330.26 s
IB	•	●	•	·	•	●	·	985.77 s
AWB	•	•	●	•	•	●	·	415.61 s
AWoB	●	●	·			•	·	25.01 s
SL	•	●						144.51 s

Table A.8: Same type of table as table 6.4, only here data is presented from the RoboCup 2001 game GMD.

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle
PD	●	●	•		●	●	●
AD	•	•	·	●	•	●	•
IB	●	●	•	•	●	●	•
AWB	•	·	●	●	·	•	
AWoB	•	•				·	
SL	•	●					
	79.03	468.58	35.29	3.53	309.31	274.95	12.57

Table A.9: Same type of table as table 6.5, only here data is presented from the RoboCup 2001 game against Trackies (second half).

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle
PD	⬤	●	·		⬤	•	⬤
AD	·	·	·	•	·	•	
IB	•	•	·	·	●	●	●
AWB	•	•	⬤	⬤	●	●	
AWoB	•	·	·		·	·	
SL	·	●					
	417.68	775.44	184.36	20.69	342.01	758.09	7.47

Table A.10: Same type of table as table 6.5, only here data is presented from the RoboCup 2001 game against Fun2maS.

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle
PD	●	●	·		●	•	⬤
AD	•	•	•	•	·	●	•
IB	•	●	•	·	⬤	⬤	•
AWB	•	•	⬤	⬤	·	•	•
AWoB	•	•	·		·	·	
SL	·	●					
	315.61	785.43	87.67	11.02	657.65	943.71	6.74

Table A.11: Same type of table as table 6.5, only here data is presented from the RoboCup 2001 game against Artisti Veneti.

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle
PD	●	●	·		●	●	●
AD	•	•	•	•	·	●	•
IB	●	●	•	•	●	●	•
AWB	●	•	●	●	•	●	•
AWoB	·	·	·			·	•
SL	·	●					
	329.06	832.49	135.31	13.59	371.93	919.93	9.95

Table A.12: Same type of table as table 6.5, only here data is presented from the RoboCup 2001 game against GMD.

Team strategy

	Attack	Defend	Intercept	Total
all	•	•	●	1170

Role distribution

	PD	AD	IB	AWB	AWoB	SL	Total
caspar	•	•	●	•	•	•	422.31
nomado	●	•	●	·		•	384.88
ronald	•	•	●	•		•	376.06
all	•	•	●	•	·	•	1183

Action type distribution

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
caspar	•	●	·	·	●	•	·	422.31
nomado	•	●	·	·	●	•	·	384.88
ronald	•	•	•	·	•	●	·	376.06
all	•	●	·	·	●	●	·	1183

Ball possession

	No one/unknown	Me	Our team	Their team	Total
caspar	●	·	•	•	423.82
nomado	●	·	•	•	365.86
ronald	●	•	·	•	375.47
all	●	·	•	•	1165

Absolute mode

	Absolute mode	Relative mode	Total
caspar	●	•	422.39
nomado	●	●	388.39
ronald	●	●	369.69
all	●	•	1180

Table A.13: Visualization of data regarding team strategy, role distribution, action types, ball possession and absolute mode. The data is gathered during the RoboCup 2001 match against against Trackies (second half).

Team strategy

	Attack	Defend	Intercept	Total
all	●	•	●	2539

Role distribution

	PD	AD	IB	AWB	AWoB	SL	Total
caspar	•	•	●	●	·	•	899.31
nomada	●	•	•	•	·	•	804.52
nomado	●	•	•	●	•	•	801.91
all	●	•	•	●	•	•	2506

Action type distribution

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
caspar	•	•	•	·	•	●	·	899.31
nomada	•	●	•	·	•	•	·	804.52
nomado	•	●	•	·	•	●	·	801.91
all	•	●	•	·	•	●	·	2506

Ball possession

	No one/unknown	Me	Our team	Their team	Total
caspar	●	•	●	•	893.29
nomada	●	·	●	•	701.10
nomado	●	•	●	•	787.72
all	●	•	●	•	2382

Absolute mode

	Absolute mode	Relative mode	Total
caspar	●	●	904.59
nomada	●	●	793.95
nomado	●	●	801.91
all	●	●	2500

Table A.14: Same type of table as table A.13, only here data is presented from the RoboCup 2001 match against Fun2maS.

Team strategy

	Attack	Defend	Intercept	Total
all	•	•	●	2811

Role distribution

	PD	AD	IB	AWB	AWoB	SL	Total
caspar	•	•	●	•	·	·	840.27
nomada	●	•	●	•	·	•	1004.53
ronald	●	•	●	•	·	·	963.03
all	●	•	●	•	·	·	2808

Action type distribution

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
caspar	•	●	·	·	●	●	·	840.27
nomada	•	●	·		•	●	·	1004.53
ronald	•	•	·	·	●	●	·	963.03
all	•	●	·	·	●	●	·	2808

Ball possession

	No one/unknown	Me	Our team	Their team	Total
caspar	●	·	•	•	840.25
nomada	●	·	•	•	1007.99
ronald	●	•	·	•	962.97
all	●	•	•	•	2811

Absolute mode

	Absolute mode	Relative mode	Total
caspar	●	●	826.81
nomada	●	●	1002.84
ronald	●	●	964.91
all	●	●	2795

Table A.15: Same type of table as table A.13, only here data is presented from the RoboCup 2001 match against Artisti Veneti.

Team strategy

	Attack	Defend	Intercept	Total
all	•	•	●	2588

Role distribution

	PD	AD	IB	AWB	AWoB	SL	Total
caspar	●	•	●	•	·	·	930.45
nomada	•	•	●	•		•	824.52
ronald	•	•	●	•	·	•	857.30
all	•	•	●	•	·	•	2612

Action type distribution

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
caspar	•	●	•	·	•	●	·	930.45
nomada	•	●	•	·	•	●	·	824.52
ronald	•	•	•	·	•	●	·	857.30
all	•	●	•	·	•	●	·	2612

Ball possession

	No one/unknown	Me	Our team	Their team	Total
caspar	●	•	•	•	916.55
nomada	●	•	•	•	828.33
ronald	●	•	•	•	833.35
all	●	•	•	•	2578

Absolute mode

	Absolute mode	Relative mode	Total
caspar	●	•	911.02
nomada	●	●	827.55
ronald	●	•	857.38
all	●	•	2596

Table A.16: Same type of table as table A.13, only here data is presented from the RoboCup 2001 match against GMD.

Team strategy

	Attack	Defend	Intercept	Total
all	•	•	●	3147

Role distribution

	PD	AD	IB	AWB	AWoB	SL	Total
caspar	•	•	●	•	·	·	1084.18
nomada	●	·	●	•		•	1010.96
ronald	●	•	●	•	·	·	1044.07
all	●	•	●	•	·	•	3139

Action type distribution

	Turn	Move	Dribble	Shoot	Seek	Chase	Idle	Total
caspar	•	●	·	·	•	●	·	1084.18
nomada	•	●	·	·	•	●	·	1010.96
ronald	•	●	·	·	•	●	·	1044.07
all	•	●	·	·	•	●	·	3139

Ball possession

	No one/unknown	Me	Our team	Their team	Total
caspar	●	•	•	•	1084.14
nomada	●	·	●	•	1017.26
ronald	●	•	•	•	1045.22
all	●	•	•	•	3147

Absolute mode

	Absolute mode	Relative mode	Total
caspar	●	●	1078.58
nomada	●	●	1023.89
ronald	●	●	1048.65
all	●	●	3151

Table A.17: Same type of table as table A.13, only here data is presented from the RoboCup 2001 match against CS Freiburg.

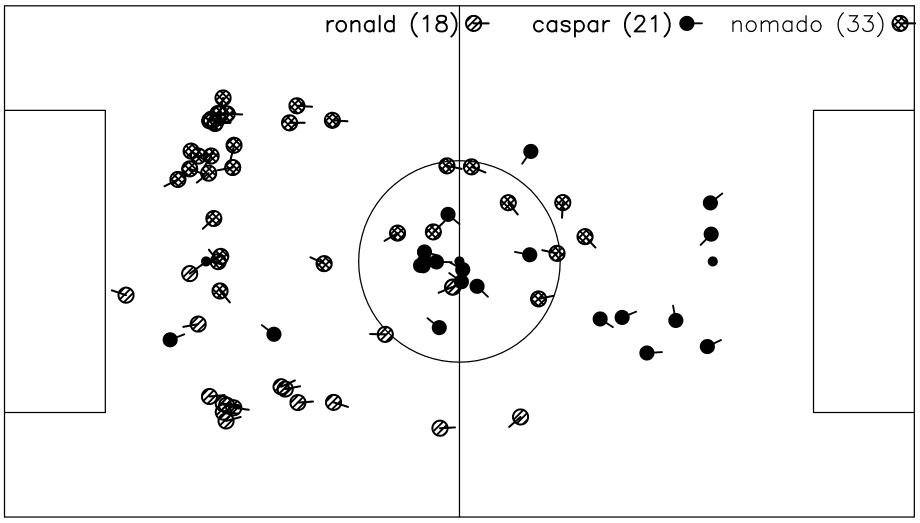


Figure A.1: Same type of figure as figure 6.2, only here data is presented from the RoboCup 2001 match against Trackies (second half).

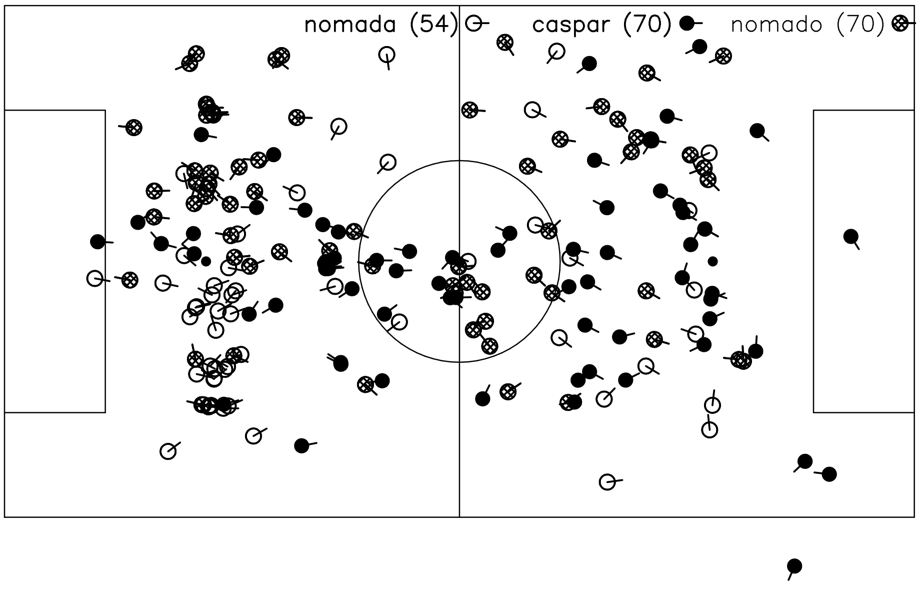


Figure A.2: Same type of figure as figure 6.2, only here data is presented from the RoboCup 2001 match against Fun2maS.

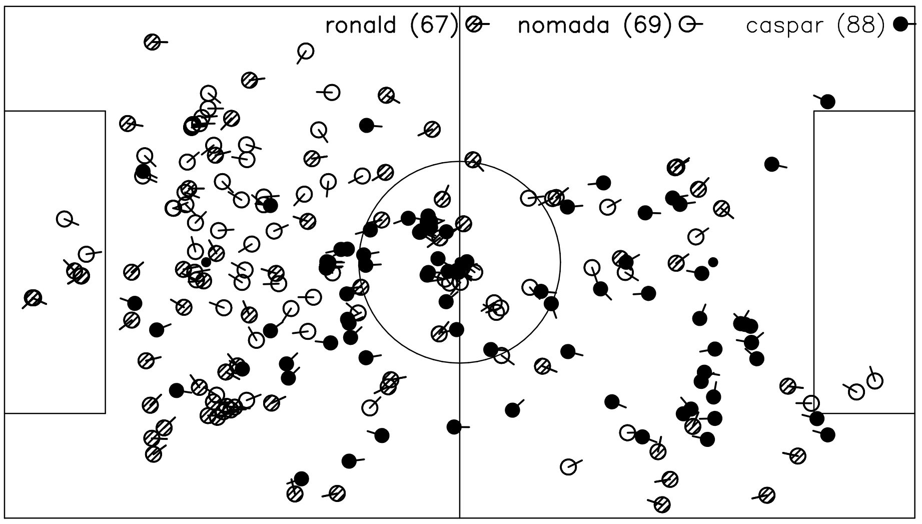


Figure A.3: Same type of figure as figure 6.2, only here data is presented from the RoboCup 2001 match against Artisti Veneti.

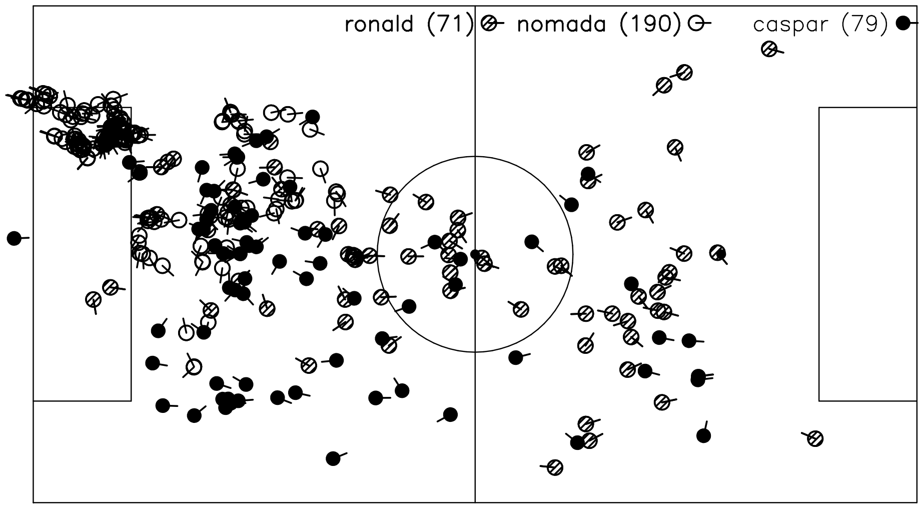


Figure A.4: Same type of figure as figure 6.2, only here data is presented from the RoboCup 2001 match against GMD.

Bibliography

[1] ActivMedia Robotics. `http://www.activrobots.com`.

[2] G. Adorni, A. Bonarini, G. Clemente, D. Nardi, E. Pagello, and M. Piaggio. ART'00 - Azurra Robot Team for the year 2000. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[3] W. Altewischer. Implementation of robot soccer player skills using vision based motion. Master's thesis, Delft University of Technology, 2001.

[4] R. Bartelds. Real time vision based self localization. Master's thesis, Delft University of Technology, 2002. To appear.

[5] E. Boendermaker. Een simulator voor RoboSoccer Middle Size League. Master's thesis, University of Amsterdam, 2002. To appear, in Dutch.

[6] R. de Boer, J. Kok, and F. Groen. UvA Trilearn 2001 team description. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[7] Johann Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 5, October 1996.

[8] A. Bredenfeld, V. Becanovic, Th. Christaller, H. Günther, G. Indiveri, H.-U. Kobialka, P.-G. Plöger, and P. Schöll. GMD-robots. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[9] BreezeCOM. `http://www.alvarion.com`.

[10] R.A. Brooks. Achieving artificial intelligence through building robots. *MIT AI Lab Memo 899*, 1986.

[11] Clockwork Orange website. `http://www.robocup.nl`.

[12] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3):213–261, 1990.

[13] P. R. Cohen and H. J. Levesque. Confirmations and joint action. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 951–957. Morgan Kaufman Publishers, Inc., San Mateo, California, 1991.

[14] P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4), 1991.

[15] J. van Dam, A. Dev, L. Dorst, F.C.A. Groen, L.O. Hertzberger, A. van Inge, B.J.A. Kröse, J. Lagerberg, A. Visser, and M. Wiering. *Organisation and Design of Autonomous Systems*. Lecture notes. University of Amsterdam, 1999.

[16] K. S. Decker and V. R. Lesser. Designing a family of coordination algorithms. Computer Science Technical Report 94-14, University of Massachusetts, 1995. A shorter version appeared in the proceedings of ICMAS-95.

[17] R. Donkervoort. An evaluation of the world model of Clockwork Orange. Master's thesis, University of Amsterdam, 2002. To appear.

[18] J. Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[19] F.C.A. Groen, J. Roodhart, M. Spaan, R. Donkervoort, and N. Vlassis. A distributed world model for robot soccer that supports the development of team skills. In *Proceedings of the 13th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'01)*, Amsterdam, 2001.

[20] B. J. Grosz. Collaborative systems. *AI magazine*, 17(2):67–85, 1996.

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

[22] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.

[23] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3), 1990.

[24] N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75:195–240, 1995.

[25] F. de Jong, J. Caarls, R. Bartelds, and P. P. Jonker. A two-tiered approach to self-localization. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[26] P. P. Jonker. On the architecture of autonomously soccer playing robots. Technical report, Applied Physics Department, Delft University of Technology, 2000.

[27] P. P. Jonker, J. Caarls, and W. Bokhove. Fast and accurate robot vision for vision-based motion. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[28] P. P. Jonker, W. van Geest, and F. C. A. Groen. The Dutch team. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[29] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2), 1998.

[30] Uwe-Philipp Käppeler. CoPS team coordination. Personal communication, November 2001.

[31] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[32] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347. ACM Press, 1997.

[33] G. K. Kraetzschmar, S. Enderle, S. Sablatnög, Th. Boß, M. Ritter, H. Braxmayer, H. Folkerts, G. Mayer, M. Müller, H. Seidl, M. Klinger, M. Dettinger, R. Wörz, and G. Palm. The Ulm Sparrows: Research into sensorimotor integration, agency learning, and multiagent cooperation. In M. Asada, editor, *RoboCup-98*, volume 1604 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1999.

[34] Gerhard Kraetzschmar. RoboCup middle size robot league (F-2000) rules and regulations for RoboCup-2001 in Seattle. `http://smart.informatik.uni-ulm.de/ROBOCUP/f2000/rules01/rules2001.html`, 2001.

[35] R. Lafrenz, M. Becht, T. Buchheim, P. Burger, G. Hetzel, G. Kindermann, M. Schanz, M. Schulé, and P. Levi. CoPS-Team description. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[36] J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[37] V. Lesser. Reflections on the nature of multi-agent coordination and its implications for an agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):89–111, 1998.

[38] V. Lesser, K. Decker, N. Carver, A. Garvey, D. Neiman, M Nagendra Prasad, and T. Wagner. Evolution of the GPGP domain-independent coordination framework. Computer Science Technical Report 1998-05, University of Massachusetts, 1998.

[39] H. J. Levesque, P. R. Cohen, and J. T. H. Nunes. On acting together. In *Proceedings of AAAI-90*, pages 94–99, 1990.

[40] K. Müller. Roboterfußball: Multiagentensystem CS Freiburg. Master's thesis, Universität Freiburg, 2000. In German.

[41] L. E. Parker. *Heterogeneous Multi-Robot Cooperation*. PhD thesis, Massachusetts Institute of Technology, 1994.

[42] L. P. Reis and N. Lau. FC Portugal team description: RoboCup 2000 simulation league champion. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[43] RoboCup official site. `http://www.robocup.org`.

[44] Th. Schmitt, S. Buck, and M. Beetz. AGILO RoboCuppers 2001 utility- and plan-based action selection based on probabilistically estimated game situations. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[45] M. Spaan, M. Wiering, R. Bartelds, R. Donkervoort, P. Jonker, and F. Groen. Clockwork Orange: The Dutch RoboSoccer Team. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[46] Y. Takahashi, S. Ikenoue, S. Inui, K. Hikita, and M. Asada. Osaka University "Trackies 2001". In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[47] Y. Takahashi, T. Tamura, and M. Asada. Strategy learning for a team in adversary environments. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

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

[49] M. Tambe and W. Zhang. Towards flexible teamwork in persistent teams: Extended report. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(2):159–183, 2000.

[50] H. Utz, D. Maschke, A. Neubeck, P. Schaeffer, M. Ritter, Ph. Baer, I. Baetge, J. Fischer, R. Holzer, M. Lauer, J. Lindenmair, A. Reisser, F. Sterk, G. Palm, and G. Kraetzschmar. The Ulm Sparrows 2001. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[51] W. J. M. van Geest. *An implementation for the Message System*. Delft University of Technology, 2000.

[52] Th. Weigel, W. Auerbach, M. Dietl, B. Dümler, J. Gutmann, K. Marko, K. Müller, B. Nebel, B. Szerbakowski, and M. Thiel. CS Freiburg: Doing the right thing in a group. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001.

[53] Th. Weigel, A. Kleiner, and B. Nebel. CS Freiburg 2001. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001*. Springer-Verlag, to appear.

[54] P. H. Winston. *Artificial Intelligence*. Addision-Wesley, third edition, 1992.