Decision Control in RoboCup Teams

Åsa Åhman
Department of Computer and Systems Sciences
The Royal Institute of Technology, Stockholm

This thesis corresponds to the effort of 20 full-time working weeks.

1.0 Introduction

This chapter will give the reader the background and purpose of this thesis. It will also describe its methodology and give an overview of the structure of the thesis.

1.1 Background

Agent-oriented programming is a relatively new area. The work that has been done is mostly in the area of personal assistants (agents) or smart search-engines on the web. There is relatively few results on teamwork in multi-agent systems (MAS). There have been attempts at building electronic market places where buying and selling agents could meet and exchange products, communicate, negotiate, and working together. But no such systems are used on a large scale, if they are used at all. Most agent systems are thus far built for research purposes only.

A domain perfect for testing many of the ideas in MAS is the RoboCup domain. The first RoboCup was held in the summer of 1997 in Nagoya, Japan [Kitano et. al.-97]. One of the major purposes of RoboCup is to promote research in agent technology. To do this, all documentation and all competing teams must be made public so that in the following years competitors can make full use of previous research, so-called ubiquitous computing.

In the summer of 1998, the next RoboCup will be held in Paris, at the same time as the real World Cup. Most teams from RoboCup'97, both robotic and simulated teams, will enter the RoboCup'98 in refined versions.

1.2 Purpose

This thesis is focused upon the area of decision control in MAS. In a group of agents working together as a team, who should make the decisions? There are systems where centralized control becomes a bottleneck for the performance, whereas there are other systems where rapid changes in strategies are vital, making centralized control preferable.

As a part of testing decision control, decision making will also be investigated. More specifically, the use of an oracle for making decisions will be discussed. The DELTA decision tool will be used as an example of an oracle in order to evaluate the this idea.

The idea of testing every aspect of agent modelling appealed to me, therefore I chose this subject. Normally, research in agent-oriented programming means selecting a small, but interesting area such as communication or social behaviour, and put all effort into that particular topic. In the RoboCup domain all agent-problems have to be dealt with. When constructing a RoboCup team not all aspects can be focused upon and fully investigated, but all of them have to be enlightened enough to make the agents work together and play (good) robotic soccer.

A more vague (or should I say naive) aim of this thesis is to implement a full RoboCup team that would enter the RoboCup'98, or at least implement the basis of such a team, that later can be fully developed by the RoboCup group at DSV, Kista.

1.3 Methodology

In this thesis I will describe the RoboCup domain and how a RoboCup team can be constructed. The original plan was to implement a full team and on that team implement two different approaches of control, but due to lack of time the control issues will only be discussed through scenarios. The team has been partially implemented. At the present time only a few roles are fully implemented. The basic and advanced skills constituting the base for all roles in the RoboCup team described in this thesis have been developed by myself and Helena Åberg [Åberg-98]. Throughout this thesis I will be referring to [Åberg-98] since she has been writing about the same team but from a different view.

Until there is a complete RoboCup team, my ideas presented in this thesis, can not be evaluated. But, if a team is implemented in accordance with my suggestions, I will get feedback and can thereafter fully evaluate the results of my theories.

1.4 Structure of the Thesis

Chapter 2 gives brief information about RoboCup and how the RoboCup players can be made to interact with the Soccer Server. It also discusses some of the restrictions imposed by the domain. In Chapter 3, it is described how a RoboCup player can be modelled, given object oriented-techniques. In the following chapter a RoboCup team is described together with the functionality of the players. Chapter 5 discusses the use of oracles, a decision making entity that guides the players. In Chapter 6, different kinds of control are inspected. Conclusions are presented in Chapter 7, and in Chapter 8 I explain how the work can be developed further.

1.5 Acknowledgement

I would like to thank my supervisor, Magnus Boman, for his constructive criticism which has helped me enormously while working on this thesis.

2.0 RoboCup

2.1 Introduction

RoboCup (The World Cup Robot Soccer) is an attempt to promote AI and robotics research by providing a common task, soccer, for evaluation of various theories, algorithms, and agent architectures [Kitano et. al.-95; Kitano et. al.-97]. RoboCup offers a software platform that forms the basis of the software of synthetic agent league. The goal is to enable a wider range of research in synthetic environments, that are today proving to be critical in training, entertainment, and education [Tambe et. al.-97]. The software agent league also promotes research on network-based multi-agent interactions, computer graphics, and physically realistic animation–a set of technologies which potentially promotes advanced use of Internet.

RoboCup provides a new opportunity for machine learning, strategic planning, and multi-agent cooperation. It is a concrete domain for evaluating these techniques and challenges researchers to develop them to face key constraints fundamental to this domain: real-time demands, uncertainty, and teamwork. For an agent (a real robot or a synthetic agent) to play soccer reasonably well, a wide range of technologies need to be integrated and a number of technical breakthroughs must be made.

2.2 Functionality

The RoboCup system is a client-server system, where every player is a client. The client connects itself to the Soccer Server, a virtual soccer field that handles all the graphical output and sends visual- and audio information back to the client every third time-cycle (300 ms). When the client wants to execute a command it has to send the information to the server that performs one request per client and time-cycle (100 ms). All the communication between the Soccer Server and a client consists of text strings. The strings has to be parsed by the client and the server accordingly to certain protocols.

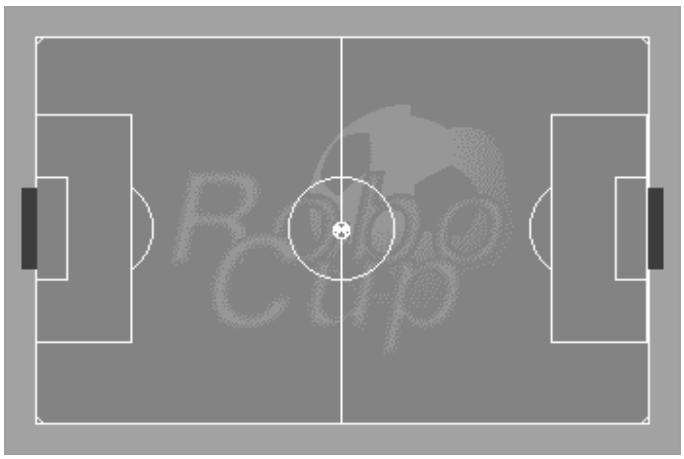

Figure 1. The RoboCup Soccer Server field.

2.2.1 Commands

The available commands for a client includes:

- *move(x, y)*

Moves the player to the coordinate (x,y). This command is only available before kick-off at the start of the game, when the players have to position them selves on the field.

- *kick(power, direction)*

  To kick the ball in a certain angle with a certain power

- *say(message)*

  To say a message

- *dash(power)*

  To run forward with a certain speed in the facing direction

- *turn(angle)*

  To turn a certain number of degrees. Positive angle means turning clockwise, negative degrees means turning counter-clockwise.

- *change view(width, quality)*

  Sets the angle and quality of the view-cone, i.e. the cone in which the player gets information about its environment.

It is with these few basic commands that a complex behaviour of a soccer player can emerge, if combined correctly.

2.2.2 Sensor Input

The player gets information in two formats:

- *Visual information*. The visual information sent to the player from the Soccer Server contains the following information: a time stamp and object information. The object information contains an object name, a distance, a direction, a distance change, and a direction change. The object name has different content depending on the kind of object. If it is a player it contains a teamname and a uniform number. A flag has flag types, which declare whether it is a goal flag, or a penalty flag, etc.

  The visual information the player gets is only from objects in its view cone, which is delimited both in angle and in length. Visual information about objects that are situated far away from the player loses some of the information concerning that object, such as the changes in distance and direction.

- *Audio information*. The player gets two kinds of audio information (hear-message): when the referee (Soccer Server), or when another player says something. The hear-messages contain a time stamp, a direction from where the message was sent, and the message.

  Even the audio information has its constraints. Messages can only be heard within a certain area, with a radius of 30 meters. There is also a limit of the number on messages that can be heard. A player can hear only one message from each team during two time-cycles, all other messages will not be heard. Since there is only one communication channel, all players hear all messages.

2.3 Problems

To construct a soccer team, or even a single player, some major obstacles have to be overcome. Most of them derive from the information format in the Soccer Server, named above.

- *Visual problems*. Since the only unmoving objects are flags, lines, and goals, information about all other objects must be handled with care. During three time-

cycles many of the objects may have moved, turned, or changed speed. This leads to that a player never can trust ‘old’ visual information fully. Letting a player act upon old information is almost like letting the player act blindfolded.

- *Only one command per each cycle*. The responsibility of only trying to execute one command each time-cycle lies on the player. The player can try to execute more than one command but that only leads to that one command is executed and the rest ignored, leaving the player confused without knowing which command was executed.
- *Threads*. To make a player that can listen for information from the server at the same time as it performs some action, multiple threads must be handled. At least two threads are needed. The players described in this thesis uses two threads, one that polls visual- and audio-information from the server, and one thread that executes the commands of the actions of the players.
- *Communication*. The RoboCup Soccer Server only supplies a single low-bandwidth, unreliable communication channel. All players, 11 per team, will have to rely on that single channel for communication. This leads to the conclusion that no team can have actions depending on communication.

3.0 A RoboCup Player

Intelligent agent technology originates from DAI (Distributed Artificial Intelligence) and has attracted lots of attention in recent years. Since the area is relatively new there is not one recognized definition of the term agent, but many. There are definitions that are too general to be of use, such as ‘*Agent is what agent does*’ [Bradshaw-97], as well as there are definitions that are too specific [Wooldridge,Jennings-95].

In this chapter I will give a brief explanation of the term agent and show why a RoboCup client is an agent. Thereafter I will explain the object-oriented hierarchy and the skills that all players described in this thesis master.

3.1 What is an Agent?

A definition that is broad enough to suit this thesis is the one proclaimed by S. J. Russell and P. Norvig [Russell, Norvig-95]. It says that *anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors is an agent*.

Sensors let the agent perceive information about its environment while effectors provide the means for the agent to affect its environment. I will not comment this definition any further. Just to define the term agent is an appropriate subject for a thesis. Such a thesis has been written by Kummeneje and Engström [Kummeneje, Engström-97]. But the definition made by Russell and Norvig suits my purposes perfectly, as explained later in this chapter.

Together with the term agent comes the term *rational agent*, an agent that does the right thing. It is obvious that an agent with irrational behaviour is not desirable, but it is neither easy defining what rational behaviour is, nor is it easy to define how an agent should behave in order to be rational [Boman-95]. One could say that rational behaviour means choosing the action that will cause the agent to be most successful. In traditional decision analysis rational agents are said to maximise something. Most common is to maximise the expected utility. Given a set of alternatives with

different consequences, the agent chooses the alternative with the highest expected utility. An agent that always follows the principle of maximising utility is called a *perfectly rational* agent.

Why this definition is perfect for the RoboCup domain, is quite intuitive. In the RoboCup domain the agents perceive their environment by getting visual- and audio-information from the server, the agents act upon their environment with a few, basic commands, sent to the server that executes them. Since there is no other way for the RoboCup client to get information concerning its surroundings, and it is only the client itself who can tell the server what commands to execute in order to change its surroundings, the client can thus be considered to be an agent.

The agents described in this thesis are in a way rational, but do not qualify to be perfectly rational, since they at times are just reactive agents, i.e. they instantaneously respond to theirs surroundings. The behaviour of the agents in this thesis will be discussed further in later chapters.

3.2 Why Object-Orientation?

Agent and object theory are closely connected. Some even say that agents are nothing but objects. Research has been done in order to evaluate the question whether agents are objects or not. A comparison between agents implemented using agent theoretic-techniques and object-oriented programming (OOP) was made by Tzikas [Tzikas-97]. His conclusions show that programming agents have a lot to gain from using OOP, since it provides a better way to model the agents and their environment, their dependencies, and their relations.

Most people agree that the theory about object-orientation can play a significant part in understanding agent theory. This is why I chose to implement the RoboCup clients in Java, which is an OOP-language. Java also handles threads, which is needed in order to let the RoboCup clients poll information from the server while executing commands (cf. 2.3).

The RoboCup clients, or players as I will call them, are implemented according to OOP-theory. All the players are based upon the *AdvancedPlayer*, a class that holds very basic soccer functions, but functions that all players, goalie as well as forward require. The AdvancedPlayer inherits its behaviour from the *BasicPlayer* class, which in turn inherits its behaviour from the *Player* class.

The Player class implements sending and receiving information to and from the server. The BasicPlayer class implements all the basic functions used in combination in order to get the complex behaviour of all the players.

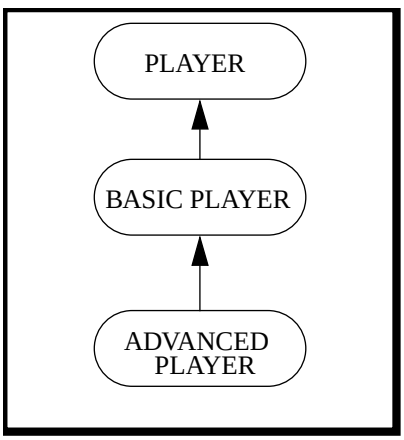

Figure 2. The object model.

3.3 Basic Skills

For an agent to act remotely like a soccer player it needs a few basic skills. At this level the player's only concern is the ball. No other players are involved. The basic skills constitute the base from which all players evolve.

3.3.1 Observing an Object

To observe an object the player does the following:

1. Calculates the position of the object. Every time a player observes an object, its position is calculated. This is because we want to be able to use old information about objects. Instead of always turning in a clockwise direction until the object is seen, we first check with the position in which the object was last seen.
2. Checks the time stamp of the visual information in the memory. If the memory has not been updated recently, then it waits for it to be updated. The player cannot rely on too old information.
3. Looks in the memory to see if the object is visible at the moment and that the last time reference in the memory of the objects is not to 'old'. Tests have shown that a time limit of 20 time-cycles is suitable.
4. If the object is not visible or if that the time reference is too old, then the player turns clockwise *viewAngleWidth* degrees which can be set to NARROW (45 degrees), NORMAL (90 degrees) or WIDE (180 degrees). The player continues turning until the object is visible, i.e., in the player's view cone.
5. When the object is visible and we have old information about the object's whereabouts, the player turns in the direction of the object. This turn does not have to be exact, since turn-commands cost and objects are visible in the player's view cone, so the direction can be within a limit of +/- 5 degrees.

These few steps result in the player following the object with the 'eyes' if the object is in motion, but it can also be used to make the player find an object.

3.3.2 Position Estimation of the Player

To estimate the position of a player, parts of a C-library written by Noda Itsuki (libsclient-3.0) was used and translated into Java. The Soccer Server gives no information to the player about its position, it only sends information about visible objects. All the calculations are made relative to the visible, unmoving objects in the players neighbourhood, i.e. lines, flags, or goals. These objects are given fixed coordinates with a virtual centre flag as origo. The length of the field is the x-axis and it runs through the middle of the goals where the right part of the field is the positive side. The y-axis runs straight through the centre flag and is positive at the lower half of the field.

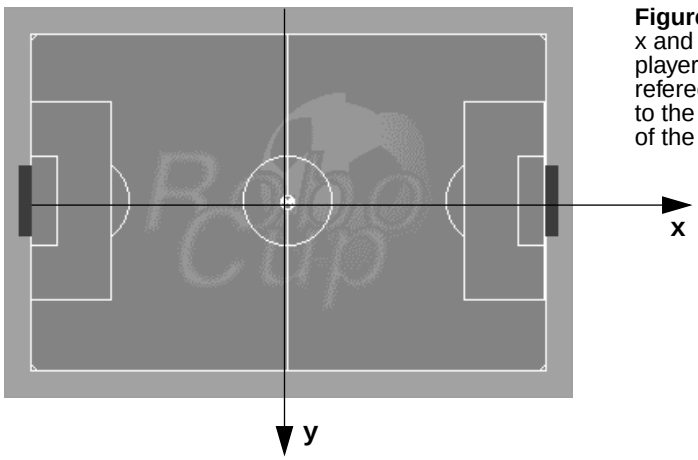

Figure 3. x and y axes for players (and the referee) belonging to the left side of the field.

To be able to estimate its own position, the player must see a line and know its direction, and a goal or a flag must be visible as well. If the player sees those things then its position can be estimated rather precisely. The flag or goal information

gives a relative position, but the line information puts it into place in the coordinate system of the field.

The whole area close to the top (and bottom) line is difficult for position estimation, since only one line is seen. Functions depending on the players position often malfunction when the player is situated in this area. But, in the most recent version of the Soccer Server, there will be more flags situated outside the field. This will hopefully clear the problems with position estimation in this area, since the new flags will be in view.

3.3.3 Position Estimation of an Object

To estimate the position of an object (another player, the ball, or some other target) the player's position is used for reference. Since the player has its own position in x and y coordinates, and the object under investigation has distance- and direction-information, this part of the calculations is trivial.

To make it even easier all the objects are marked with a flag: isMobile. The non-moving objects have this flag set to false, so whenever a player wants to estimate the position of an object that cannot move, no calculations have to be made.

There is only one problem: the coordinates of the non-moving objects are related to the coordinate system of the referee, i.e. the Soccer Server (see figure 3 above). The referee has the same coordinate system as players belonging on the left side of the field. This requires the immobile objects to be scaled for players belonging on the right side of the field, since they have a different coordinate system (see figure 4 below).

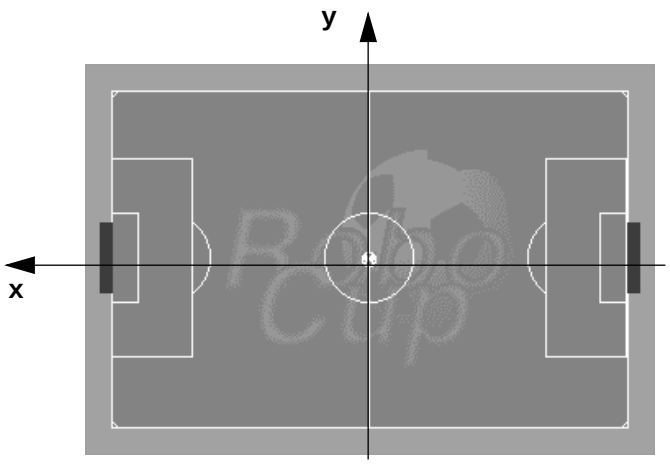

Figure 4. x and y axes for players belonging to the right side of the field.

3.3.4 Run to a Position

In many functions it is desired to make the player run to a specific point, a coordinate, instead of to an object. If no new visual-information has reached the player since the last call to the function, the players assumes that it is facing the point and dashes forward. When the player has new visual-information it checks its angle and distance to the point of destination and corrects the angle if necessary. As the player gets closer to the destination point, it slows down to avoid passing the point when trusting old visual-information.

3.3.5 Ball Collection

When a player wants to ‘get in the way’ of a moving ball in order to aim for a specific target, some calculations are needed. First of all, the player’s position must be known, as well as the position of the target and the ball. Since the ball is moving, its present position cannot be used. Instead, a future position on the path of the ball is calculated, depending on the motion of the ball and how quickly the ball is moving away from the player. In order to get the line of motion, the ball is observed twice within a short interval to calculate two different positions of it and thereby get the trajectory of the ball, so called interpolation.

The *dribble line*, i.e. a straight line between the target and the ball’s future position is calculated. Onto this line, the position of the player is projected to see where the player is located in regard to this line. The player can be on its backward extension, on the line, or on its forward extension. When the player is on its forward extension, it has crossed the *cutting line*; this is the first step of the desired placement. But before the player is behind the *cutting line* it should dash in parallel to the *dribble line*. Depending on which side of the *dribble line* the player is located, it turns away from the *dribble line* to face the *cutting line*, then dashes towards it (see figure 5 below).

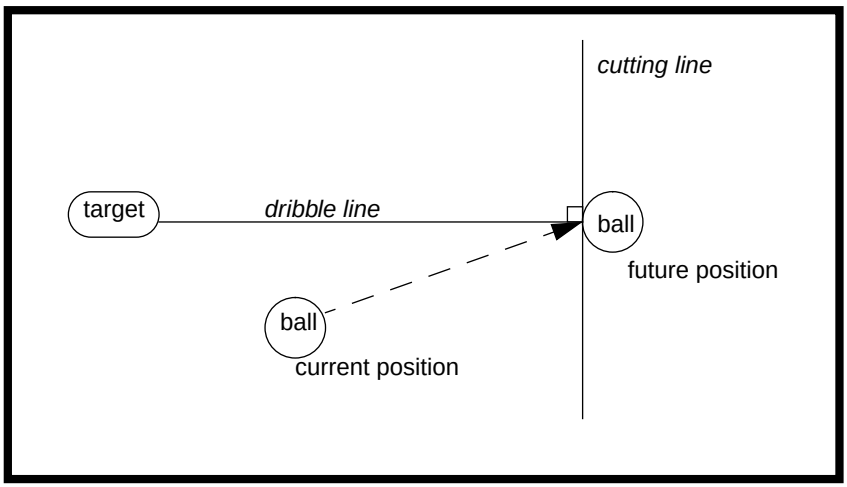

Figure 5. Ball collection.

When the player is located behind the *cutting line*, it runs to the ball. This can result in an ‘edgy’ way of running, but it saves time. This way of solving the problem with ball collection is discussed in [Stone, Veloso-98].

3.3.6 Turn Towards a Target With the Ball

Since no player can ‘hold onto’ the ball, turning with the ball is a problem. But with a combination of the turn- and kick-commands you can simulate the player to turn and keep the ball close. When a player wants to turn towards a target while keeping the ball close, it kicks and turns every other time-cycle until it sees the target. With kicks at a low power and in a wide direction, it will make it seem as if the ball is kept close to the player while turning. Depending on where the target is situated, the player kicks and turns in a positive or negative angle. A more efficient and advanced way of doing this has been done by an Italian RoboCup team, PaSo, their players

performed this action so well it seemed as if the player and the ball were glued together while turning!

3.3.7 Intercept the Ball

To intercept the ball can be seen as a sub-problem of ball collection. When the player wants to intercept the ball without having a target to aim at afterwards, it calculates a point on the path of the moving ball, and runs to it. To find the interception point, the player's position is projected onto the trajectory of the ball. The trajectory is interpolated from position coordinates observed twice within a short interval (cf. 3.4.5).

If the player reaches the interception point ahead of the ball, instead of waiting for the ball to get to the interception point, the player runs to the ball. This can result in the same kind of edgy line of running as with the ball collection, but for the same reason it is justified.

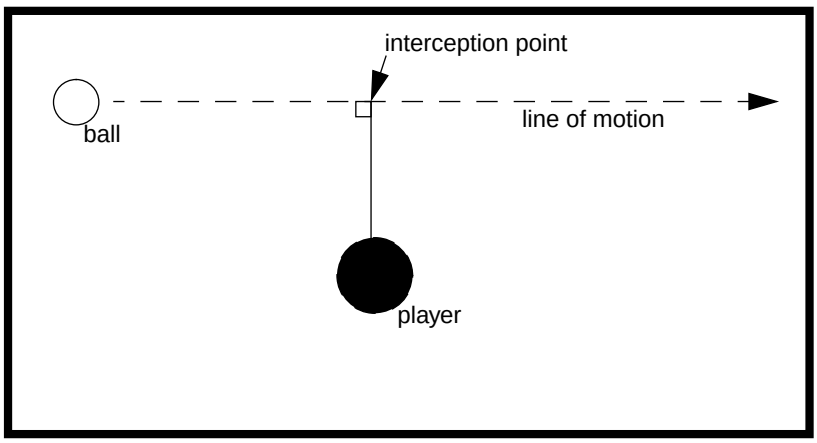

Figure 8. Intercepting the ball.

3.4 Advanced Skills

There would be no soccer if there was only one player. To make a player deal with other players, team-mates and opponents, and to make it adjust to changes of the surroundings, some advanced skills are required.

3.4.1 Communication

Players will have to communicate to achieve good performance. Hearing is free and instant. Saying something, on the other hand costs and you can never be sure that the things that are said gets through to all players. Because of this, communication cannot be relied on and has to be used with utter care.

To avoid confusion, when players say something they always address a certain player. When a player hears something, it always checks if it is from the right team, and if it is, the player checks if it is addressed to it, i.e. if the message contains the uniform number belonging to the player. If the message is to another player, it ignores the message. If the player is doing some important task while it hears a message directed to it, it also ignores it.

3.4.2 Passing

The initiative to a pass can be taken from:

1. *A player other than the player with the ball*. All the other players who at the time do not have the ball, evaluate their position to see if they have a better position than the player with the ball. If a player finds this is the case and that it has a clear view towards the player with the ball (so that a pass is possible) and a clear view towards the target (opposite goal, another player, etc.), then they call out ‘pass to me’.
2. *The player with the ball*. To save the player with the ball from making costly calculations and evaluations, this is not the normal case. But, if the player with the ball finds its surroundings too threatening, it chooses an arbitrary team-mate in its neighbourhood, kicks the ball in that players direction and says ‘I am passing you’. Because of the threatening environment the player can not take the time to find the ‘best’ team-mate to pass, instead it just picks one close by.

When the player with the ball gets a pass-request it evaluates its situation. If it finds its surroundings safe, i.e.there are not enough players of the opposite team close to the player, it ignores the requests. But, if it comes to the conclusion that a pass would be the safest thing to do, it kicks the ball in the direction of the requesting player while saying ‘I am passing to you’. In the message the requesting player’s uniform number is attached, to avoid confusion among the other players.

The other half of passing, the intercepting part, was described in the Basic Skills section above.

3.4.3 Obstacle Avoidance

In every time-cycle the player looks for obstacles.

- *Running into things*. Collisions are unavoidable. Sooner or later two players will run into each other, and they have to be able to get lose without help. One particular problem is when players get stuck without seeing what is holding them back. This can occur when the side of a player gets caught in the side of another player. With a view angle of 90 degrees, they cannot see each other. Since the players have no (easy) way of knowing that the command they send to the server gets executed or whether they are moving or not, the players can be standing in the same place, trying to dash forward without noticing that they are standing still. To remedy this problem is not easy with the current version of the Soccer Server. But in future versions there are plans of a *sense-body*-command that gives the player information about execution of commands as well as current speed.

  When the player does notice that it is caught in something, there are ways to repair the situation. The player dashes backwards a couple of times and tries to run around the obstacle holding it back. It the obstacle still is to close, it dashes backwards until it has reach a *safe* distance.
- *Running without the ball*. When a player is running towards a target without the ball it is that player’s responsibility to avoid running into obstacles. When an obstacle is found, a team-mate or a player from the opposite team, the player runs around it.
- *Running with the ball*. When a player is running towards some target with the ball, the situation is somewhat different. If the ball-keeping player finds that a team-mate is standing in its way, it is that player’s responsibility to move. In order to make the player realise this, the ball-keeping player tells that player to move away. Since messages come with a direction, the player in question can calculate the best way to run in order to get out of the way with minimum effort.

If the ball-keeping player finds an obstacle to be a player from the opposite team, running around it with the intention of keeping the ball is quite naive, instead it passes the ball to a team-mate.

4.0 A RoboCup Team

In this chapter I will describe the structure of a hypothetical RoboCup team and what roles are associated with the players in such a team. I will also describe the behaviour of the players and how they decide what to do in critical moments.

4.1 Team Structure

A team will consist of different types of players with well-defined *roles*. The players are all based on the advanced player with the basic and advanced skills. The different roles described here are based up on a construction by Helena Åberg [Åberg-98], with whom I have been working closely. What I have added is the decision control plus functions and behaviour to make the players act according to centralized and decentralized control.

4.1.1 Players With Different Behaviour

In most of the RoboCup teams that competed in Nagoya, Japan, the players had different roles. Some teams [Ch'ang-97] let the players choose a role according to the current situation. All the players had the ability to choose any role at any time. The players were supposed to analyse the situation and take into consideration all the other players and their positions. Unfortunately this team did not even make it through the first stage and never made it to the actual competition. But for the RoboCup'98, Stone and Veloso has implemented a team with flexible roles that seems to be rather efficient [Stone,Veloso-97b].

The team described in this thesis lets a player have only one specific role, which it will have throughout the game. Letting the players have flexible roles is a robust solution, since the team will function without some of its players. But the main purpose of this team is not robustness, instead the focus is upon making full use of OOP-techniques. The different roles are end-nodes in a hierachic tree structure, built with the *Basic* and *Advanced Skills* in the top nodes. For a more detailed description of the roles, see [Åberg-98]. Some of these roles in the team are:

- *Goalie.* Its main responsibility is to watch the goal. In the latest version of the Soccer Server, the goalie has a special command, *catch*, which lets the goalie instantly stop the ball if it is within reach. Normally this is a big problem for the players since kicking a ball is preferably done when the ball is immobile. Unfortunately, all the benefits of the new Soccer Server could not be taken advantage of at the time of writing this thesis and it is unclear to which extent the new server will be used in RoboCup'98.
- *Defender.* Shows a behaviour similar to the goalie. Both goalie and defenders inherit their behaviour from an abstract defender class. Its main functionality is to prevent players from the opposite team to enter the goal area, and if a defender gets in control of the ball, it tries to pass it to a team-mate further up on the field.
- *Forward.* A very offensive player. It basically tries to push the ball as close to the opposite goal as possible, preferably into the goal itself. It constantly tries to put itself in a good position to score or to be passed to, if it is not in control of the ball.

- *Midfield.* An offensive player, but not as offensive as a forward. The two roles have a similar behaviour, analogously to the goalie and the defender, the forward and the midfield inherit their behaviour from an abstract midfield class. A midfield player acts as a layer between the defenders and the forwards. It helps the defenders with defending the goal area, and do its best to get the ball to a forward.
- *Coach.* This is a very specific player, used only when the control is centralized. It is the player making all the decisions and tells the rest of the team to change strategy if needed.

4.1.2 Teamwork

Teamwork is built into the actions of the players. When a player analyses its current position it takes into consideration the presence of team-mates. For example, if a player find itself in a better position to score than the player who currently controls the ball, it tells the ball-keeping player to pass. Or if a player with the ball feels threatened by opponents, it passes the ball to a team-mate.

Since communication is poor, 'normal' teamwork which would include discussing what to do at certain times, is ruled out. The players have to rely on built-in knowledge about its team, i.e. its team-mates and their roles. When players get good visual information the players can draw conclusion of its team-mate's number to make an accurate guess what role it has, but since players situated far away are difficult to see properly, knowledge about roles can seldom be used.

A player can for example not make a pass thinking the receiving player will force its way to the opposite goal. If the receiving player is a defender, it will most probably just make a pass to another team-mate, and thereby risk losing control of the ball.

4.2 States

At every moment the player is in a *state*, which is based on its observations of the game. All the player actions are controlled by states. In a specific state a player has a number of actions to perform. In order for the player to leave a state and enter another one, certain criteria have to be fulfilled. The criteria are listed as sub-states or tasks. For example, a player that is in the state PASSING, should not exit that state until it has kicked the ball to a team-mate, or lost the ball to a player from the opposite team.

All players have an initial state, in which they have to analyse the situation and decide on what to do next. Depending on the current situation the players can choose almost any state. Whether the player has the ball or not, filters out some of the states as does the number of threats from the other team. A player can, e.g., not enter the state PASSING, unless it is in control of the ball.

An example of a state a player could enter is BALL_COLLECTION. This state is appropriate whenever the player wants to catch a moving ball. The state is split up into sub-states or tasks, in order to let the player continue with the proper action when it enters the state the following time-cycle. First the player has to face the cutting_line, i.e. the straight line between the ball and a future target. When the player is facing that line, it enters the next sub-state. Other examples of possible state a player could enter is: RUN_TO_POSITION, PASSING, INTERCEPTING, LOCATE_TEAMMATE_TO_PASS, etc.

4.2.1 The State Object

In order to understand the idea behind the states one has to understand one of the problems previously listed in this thesis (cf. 2.3). Since it is only allowed to execute one command each time-cycle, the player will have to exit its main loop after every command. In the following time-cycle the player has to 'remember' what it was doing the previous time-cycle in order to continue with its work. This is solved by letting the players have a state-object that holds a pointer to the previous state and a list of all the sub-states or tasks related to the state. After executing a command, the player checks if it has completed a task or even a complete state and updates its state-object accordingly. The next time-cycle, the player checks with its state-object to see what to do next.

4.2.2 Changing States

There are two ways for a player to change state:

- **Automatic changes**. Some states have an automatic transfer of states. By automatic means that the player does not have to do anything specific to change its state, the change is built into some other state. For example, when a player requests a pass, it automatically goes into the state LISTENING. If no answer is received within a certain time period, the player returns, automatically, to its previous state. But, if the player gets a message saying 'I am passing to you', the player automatically enters a state, INTERCEPTING, to intercept the ball. When the player has caught the ball or if a player from the opposite team takes the ball, the player has to change state to its previous state.

  If a player receives a message saying 'I am passing to you' without having made a request, it instantly exits whatever state it is in for the moment and enters the state INTERCEPTING.
- **Oracle suggestions.** Eventually, the player returns to the initial state. The initial state has no sub-states or tasks. Instead of remaining idle, the players asks an oracle for guidance.

5.0 The Use of an Oracle

Having a detached resource of information and decision making can be called having an oracle. The ancient greeks had their oracle in Delphi but the oracles discussed in this chapter are mathematical functions with some statistic analyses and risk control. The use of an oracle would be the rational part of the player, since its main task would be to maximise the expected utility of alternatives, given the consequences of alternative actions.

Having an oracle in the RoboCup domain could enhance the performance of the players, by making them act more rationally. Instead of letting all the players have individual functions for decision making and statistical analyses that would have to be implemented from scratch, these properties could be taken from a decision tool which has already been built and tested. Since there are no other ways of communicating in the RoboCup domain than with the say and hear messages, it is not possible to have a totally detached oracle, without cheating. Such an oracle could either be put in one of the players that would be the dedicated 'oracle' or the oracle would be a function available to all the players: the oracle could be considered a backpack that the players would be carrying around.

Whenever the players have to make a decision or analyse a situation, instead of doing the calculations, e.g., maximising the expected utility, by themselves, they

would consult the oracle. A good candidate for such an oracle is the DELTA Decision Tool. Due to lack of time, no implementations regarding DELTA being used as a oracle for a RoboCup team was made, but in the following chapters a description will be put forward to show how it could work.

5.1 DELTA Decision Tool

In the research group DECIDE at DSV in Kista, Sweden, years have been spent developing and implementing methods and algorithms for decision analysis, that is grouped under the name DELTA [Danielson-96; Danielson-97].

5.1.1 Maximise Utility

The team described in this thesis would only use a small fraction of DELTA's functionality. Given a number of actions and their consequences, DELTA can, in some sense, calculate the best action. To do this, DELTA requires probabilities and values associated with the different actions.

For example, consider two possible actions, A1 and A2. A1 and A2 have two consequences each, C1.1, C1.2, C2.1, and C2.2 respectively.

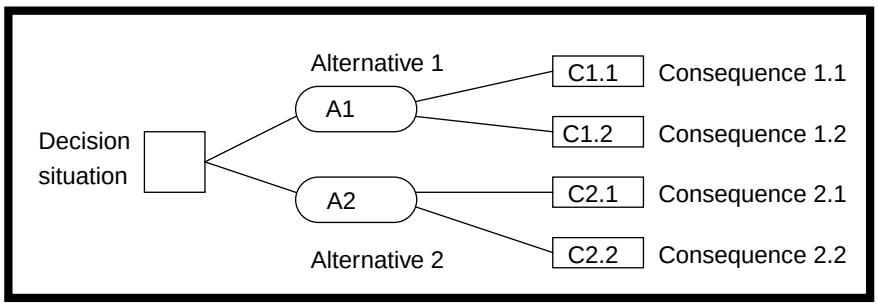

Figure 6. Decision tree.

- C1.1 has the probability span [p1 - p2] and the value is in a span [v1 - v2].
- C1.2 has the probability span [p3 - p4] and the value is in a span [v3 - v4].
- C2.1 has the probability p5 and the value is in a span [v5 - v6].
- C2.2 has the probability span [p6 - p7] and the value is v7.

DELTA then calculates the best alternative of A1 and A2 by maximising the expected utility. This is not the whole truth, but it is how DELTA would be used by a RoboCup team. A more accurate description of what actually happens is that DELTA does not give just the best alternative but rather the background to make a choice between the alternatives with consideration to risks. When DELTA is forced into returning only the 'best' alternative, this is done without regard to the risks involved.

5.1.2 What DELTA Will Choose From

An easy and straightforward way of using DELTA for a RoboCup team would be to set up different queries to choose from. The queries could be viewed upon as a database of rules as in an expert system, there structure would be the same as of the decision tree in figure 6. One possible future use of DELTA is to let the queries be

set up at real-time, i.e. whenever a player is in need of advice, it builds the correct queries based upon its current state.

Whenever a player needs advice from DELTA, it primarily has to match its current state to one of the DELTA queries listed below. Depending on what role a player has, a sub-set of these queries would be used. For example, the goalie would never have an alternative that recommends trying to score just as a forward would not have to deal with alternatives about protecting the goal area.

Player with ball	Player without the ball
In goal area	Sees the ball
Threats	Ball in hands of opponents
No threats	In own goal area
Not in goal area	Not in own goal area
Threats	Ball in hands of team-mates
No threats	In opposite goal area
	Not in opposite goal area
	Does not see the ball

In order to explain the idea behind the queries and their alternatives, only one decision tree was listed here. Another simplification is the development of the alternatives within the queries. Only the queries where the player does not possess the ball are listed below. Most of the queries where the player has the ball would be handled by other processes. For example, when a player is dribbling and experiences threat, it passes to a team-mate.

The different alternatives in the five queries are listed below.

1. **Player without the ball, who sees it in opponents hands in own goal area.**
   - **1.1** Run to the ball
   - **1.2** Stand between the ball and the goal (optimize position)
   - **1.3** Stay still
   - **1.4** Cover an opponent
2. **Player without the ball, who sees it in opponents hands, not in own goal area.**
   - **2.1** Run to the ball
   - **2.2** Stay still
   - **2.3** Cover an opponent
3. **Player without the ball, who sees it in team-mates hands in opposite goal area.**
   - **3.1** Cover an opponent
   - **3.2** Run to a good position to score (optimize position)
   - **3.3** Stay still
4. **Player without the ball, who sees it in team-mates hands, not in opposite goal area.**
   - **4.1** Run to a position better than position that the ball-player currently possesses (optimize position)
   - **4.2** Stay still

4.3 Cover an opponent

5. **Player without the ball, who does not see the ball.**

5.1 Stand still

5.2 Run to a random position for a better look

The queries and their alternatives above would build up decision trees, just as in figure 6. To simplify matters only one of the queries will be fully developed:

If the decision situation (**A**) were: **Player without the ball, who sees it in opponents hands in own goal area**, it would have the following alternatives:

- **A1** Run to the ball
- **A2** Stand between the ball and the goal (optimize position)
- **A3** Stay still
- **A4** Cover an opponent

A1 has the consequences:

- **C1.1** The player succeeds in running to the ball. This consequence has the probability of 0.1 - 0.3 and a value higher then 0.7.
- **C1.2** The player does not get to the ball in time. This consequence has the probability higher than 0.7 and a value lower than 0.3.

A2 has the consequences:

- **C2.1** The player is able to place itself between the ball and the goal (before the opponent kicks the ball). This consequence has the probability of 0.3 - 0.5 and a value between 0.8 and 0.9.
- **C2.2** The player is not able to place itself between the ball and the goal (before the opponent kicks the ball). This consequence has the probability of 0.6 - 0.8 and a value lower than 0.2.

A3 has only one consequence, since it can not falter in standing still.

- **C3.1** The players is standing still. This probability is 1 and the value is close to 0.

A4 has the consequences:

- **C4.1** The player is able to cover an opponent. This consequence has the probability of 0.3 - 0.7 and a value of 0.5 - 0.8.
- **C4.2** The player is not able to cover an opponent. This consequence has the probability of 0.2 - 0.5 and a value of 0.2 - 0.4.

5.1.3 DELTA in Real-time

To be able to use an oracle, such as DELTA, it has to be able to give an answer in real-time. Unfortunately DELTA was not built for real-time use, but since only a small fraction of DELTA's capabilities are supposed to be used, this should not be a problem.

6.0 Control

Due to lack of time, none of the theories given below were implemented. The plan was, given a team structure with a decision making tool or an oracle, to make tests upon performance of the team with centralized and decentralized control. At the moment there is no complete team structure, and there is no implemented interface to interact with an oracle. However, the risk that implementations would not be completed within the work span of this thesis was high from the outset. The present chapter thus deals with the plans for implementations, so that they may be completed at a later date.

6.1 A Test Scenario

The two different types of control were supposed to be tested on a team consisting of defenders, forwards, midfield players, and a goalie. The line up could be done in various ways, one possible line up is presented in the figure below. But a more common line up is 4-3-3 that means the team would consist of 4 defenders, 3 midfield players and 3 forwards (and a goalie of course).

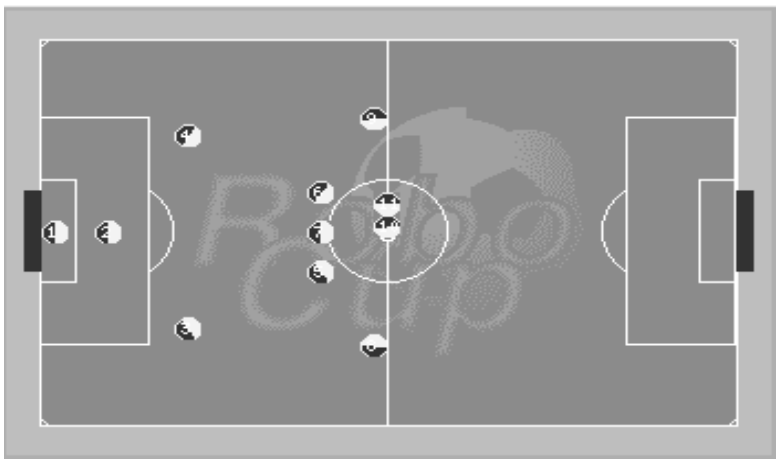

Figure 7. A possible line up for the team.

In order to evaluate the different types of control, various parameters could be logged, e.g., the number of requests to DELTA. The response time, i.e. the time from which the player issues a say-command with its request until the player gets a reply from the coach with the results could also be noted. A very interesting thing to observe is the number of requests that get lost. A player could be made to clock the response time. If the player would be kept waiting for a certain number of time-cycles, a new request should be made and the old request would be considered lost.

Since this test scenario was not implemented, I will just discuss the possible outcome of such a test.

6.2 The RoboCup-team with Centralized Control

6.2.1 What Makes Control Centralized?

When one player, committed to be the coach or leader, has knowledge that makes it capable of making decisions for every player at any time when the other players have to rely on the coach for guidance, the control is said to be centralized.

Those teams from RoboCup'97 who have released their documentation have not used centralized control. In electronic markets, where agents meet and exchange products [Wellman-96], there is a broker that could be said to be 'in control'. But the control is not in having the ability to change strategies or telling the other agents what to do, the broker is more of a connector, that has the information required by the other agents in order to sell or buy products or services.

6.2.2 How Does it Work?

To manage the control, one player has to be assigned the role of coach. When a player has to decide what state to enter next, it sends a message to the coach containing information about the present situation the player is in, i.e. the latest see-information. When the coach hears the message, it makes a match between the player's surroundings and the available oracle queries, to see what query to use. When it has found the right one, it makes a request to the oracle. The result from the oracle is processed, and recommendations on what state to enter are sent back to the requesting player.

6.3 Expected Results with Centralized Control

6.3.1 Advantages of Centralized Control

To make strategy changes, the coach is the only player that has to be changed. If the oracle queries would be expanded, the changes would be made in the coach only. Or if the oracle queries (in some future version of the team) would be made up at real-time, based on the given situation, only the coach would be affected.

6.3.2 Drawbacks of Centralized Control

Since communication is costly and insecure, this method has its weaknesses. A player's request for guidance can be ignored or lost because of bad communication. If many players ask for advice at the same time, only one request will be processed, the others will be lost. Since there is no way to buffer the hear-messages, the coach will not even know it has been getting requests and can therefore not answer them later. Since only one message can be heard every time-cycle, everything the opponents say will distract the coach, as well as the requests from the team-mates.

A player has to be in a certain range of the coach to be able to speak to it and to hear its reply. This means that when a player wants to get some help from the coach it first has to move itself close enough to the coach. This is extremely bad since players could need help in situations where moving would mean losing the ball or perhaps letting a player from the opposite team score.

The coach can do little else than stand in the middle of the field (as close to the centre as possible), listening to requests and answering them. This is because the command *say* 'costs': when the coach has said something, it has to wait for the next time-cycle to issue another command.

Since saying something ‘costs’ one time-cycle, this results in a total cost of at least two time-cycles when a player sends a request to the coach. During those time-cycles the requesting players situation may have changed drastically, and the reply from the coach may not be valid at the time of the arrival.

6.4 The RoboCup-team with Decentralized Control

6.4.1 What Makes Control Decentralized?

When all the players have the ability to make their own decisions at any time, their control is said to be decentralized. The players do not depend on any other player to perform their roles.

Most documented teams, such as the team CMUnited [Veloso, Stone, Han-97], use a decentralized control. No player is a dedicated coach, instead the CMUnited-team uses previously set strategies and communication for updating strategies and plans. Other teams, such as AT Humboldt [Burkhard, Hanebauer, Wendler-97], do not use communication. Instead they use previously set strategies and real-time simulations, in which the players for example try to evaluate if they should run to the ball or let a team-mate do it instead.

6.4.2 How Does it Work?

Whenever a player enters the initial state, i.e. a state that does not automatically lead to another state, it consults the oracle. The player then individually has the responsibility corresponding to a coach. The player analyses its current situation on the basis of the latest visual-information. It matches the information about its surroundings to an oracle query and consults the oracle. When the oracle replies, the player enters the state suggested by the oracle.

6.5 Expected Results with Decentralized Control

6.5.1 Advantages of Decentralized Control

With decentralized control there is no bottleneck in the form of a coach. The players never have to wait their turn to make a request. Decentralized control is more robust since it is not dependent on communication. No request can get lost. The waiting time for an answer is also shorter compared to the waiting time with centralized control. Since no messages have to be sent, and the say-command costs a time-cycle, this time (at least two time-cycles) will be saved. The fact that the reply from the oracle is ‘immediate’ (with respect to the real-time issue) means that there is a bigger chance that the reply is valid. The changes in the player’s neighbourhood will not be so great as to make the reply from the oracle invalid.

6.5.2 Drawbacks of Decentralized Control

A key player can be motionless for some time while it is making a decision. If a critical situation appears, many players may have to question the oracle which may lead to the key players being idle while the oracle processes the request, which in turn would lead to negative results for the team.

If there is a need of changing strategies that has to be either coded into every player from the beginning, or the players must depend on communication. Both alternatives might work, but having every behaviour previously coded is not flexible while communication problems might lead to parts of the team not getting the information.

7.0 Conclusions

7.1 Control

Even though the ideas presented in this thesis were not implemented, it is obvious that decentralized control is preferable in a domain such as RoboCup. Letting the players make their own decisions based upon their environment at the time is the best solution since, e.g., communication is poor and because of the limited visual capabilities. If it were possible to have a player with full view of the entire field at all time, the situation would be different, especially if communication was more reliable. Then I would say centralized control might be a better choice, but at present, the constraints of the RoboCup domain are such that this can not happen. Nor do I expect the RoboCup domain to change in the future, since RoboCup tries to reflect a true soccer game. Communication in soccer is not so very different from the single communication channel available in the RoboCup domain today.

The main point of failure for centralized control is the communication restraints. In other multi-agent systems, such as the military application, an air combat simulation system, made by M. Tambe et. al. [Tambe et. al.-97], reliable communication is vital. The agents in the air combat simulation constantly update their strategies based upon what the other agents experience and believe. This could not be done without good communication. But, in the report by Stone and Veloso [Stone, Veloso-97a], a RoboCup-team with a functional message-system is described. Even though they made tests that indicated that not much was to gain from communication, the team CMUnited got in fourth place in RoboCup'97 with the message-system described in their report. This shows that some communication can be useful, even though it is unreliable.

7.2 DELTA

DELTA may not be the best alternative for a RoboCup oracle. It was constructed for a different purpose than the real-time domain of RoboCup. Since a RoboCup team would use only a small fraction of all its possibilities it feels like an awful waste to use such a powerful tool for such elementary calculations as are required by a RoboCup team. But the idea of having an oracle is still interesting. It seems to be a good idea to let the RoboCup players use an oracle, since a good decision making tool may add plenty to performance. In other domains where the agent has to be portable, an oracle such as DELTA is not preferable. It takes a lot of space and the agent has to be compiled together with a library containing the DELTA-functions. In order to do this, DELTA has to be reachable from the machine on which the agent will be used.

8.0 Future Work

The work that can be put into this RoboCup team seems endless. One requirement is full use of the new Soccer Server. The new server has a shorter sensing cycle that will allow the player to get new sensor information more rapidly, and it also has a new command, catch, that will improve the behaviour of the goalie. There are also many new flags on the field and even some flags outside the field. This will give a more accurate position estimation. Hopefully with the new flags the player will never fail to estimate its position. Another change is that a player can make some command parallel in time, or more accurately, some commands can be executed as often as wanted, they do not 'cost' one time-cycle. For example could a player kick

the ball while saying something at the same time. There is a command *change view* that sets the quality of the see-information. When the quality is low, the see-information comes more often. This, together with the fact that the *change view* command can be done how often as wanted, could result in better performance.

The use of an oracle could be extended. An decision making tool such as DELTA most likely contains lots of other functions, e.g., concerning reliability of probabilities that could be used to make the decision making more exact. Other possible changes include extended flexibility in many aspects. The probability used spans could be updated during the game with some machine learning technique. Updates of the value spans could be made together with the probability span in order to change the players' strategies. When the team has to be more defensive, all the probabilities and the values for actions or consequences concerning defending the goal or not taking risks would be favoured. The given queries could be more flexible and be created at the time of an request, instead of using static queries as is the case now.

The roles of the players could be more complex than what they are today, and most important they could be implemented and evaluated! The team structure described in this thesis was a theoretical structure put together by Åberg, it was neither fully implemented, nor tested.

A fully developed interface to an oracle would also have to be implemented in order to evaluate the oracle-idea.

Good performance with goal-kicks, corner-kicks etc. is necessary to make the team compatible with other teams. The players must make clever formations rapidly, since whenever the opposite team are making the kick, there is no way of telling how much time the players have to place themselves before the kick is done.

More states are needed for the players to choose from in order to get a more complex behaviour. This is particularly important if the players are to get more explicit roles. For example, the forward players could be split into a left, a right, and a centre forward, all with small differences in their behaviour (and thus their states) which would improve the play of the team.

Since code and papers about other competing teams have been made available, lots of information and techniques could be gathered from other researchers' experiences. There are teams that have excellent performance in different areas: some teams are extremely good at turning with the ball, others are experts of passing. Much could be learned from others. One idea is to take a team from a previous year and base all further development on that team. It would require some time to fully understand code written by someone else, but it could be worth it, since it takes a lot of time building a team from scratch.

The communication paradigm described in [Stone, Veloso-97a] should be implemented, in order to have the best communication possible. They have showed that communication can be secure, if used with precaution.

References

[Boman-95] Magnus Boman, “What is Rational Agency?”, Internal Working Note 95-048, DSV, 1995.

[Bradshaw-97] Jeffrey M. Bradshaw, “Software agents”, AAAI/MIT 1997.

[Burkhard, Hanebauer, Wendler-97] Hans-Dieter Burkhard, Markus Hanebauer, Jan Wendler, “AT Humboldt - Development, Practice and Theory”. To appear in: Proceedings of The First International Workshop on RoboCup. LNAI, Springer-Verlag, 1997.

[Ch’ng,Padgham-97] Simon Ch’ng and Lin Padgham, “Team description: Building teams using roles, responsibilities and stategies”. Unpublished material, 1997.

[Danielson-96] Mats Danielson, “DDT - The DELTA Decision Tool”, presented at “Advances in Methodology and Software for Decision Support Systems”, IIASA, Laxenburg, Austria, September 1996.

[Danielson-97] Mats Danielson, “Computational Decision Analysis”, Ph.D. Thesis, DSV, Report No. 97-011, May 1997.

[Kitano et. al.-95] Hirokai Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda and Eiichi Osawa, “RoboCup: The Robot World Cup Initiative”. In Proc. of IJCAI-95 Workshop on Entertainment and AI/Alife, Montreal, 1995.

[Kitano et. al.-97] Hirokai Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Istuki Noda and Minoru Asada, “The RoboCup Synthetic Agent Challange 97”. Unpublished material, 1997.

[Kummeneje, Engström-97] Johan Kummeneje, Henrik Engström, “DR ABBility: Agent Technology and Process Control”, M.Sc. Thesis, DSV, 1997.

[Russell, Norvig-95] Stuart J. Russell, Peter Norvig, “Artificial Intelligence - A Modern Approach”, Prentice Hall, 1995.

[Stone, Veloso-97a] Peter Stone, Manuela Veloso, “Communication in Domains with Unreliable, Single-Channel, Low-Bandwidth Communication”. Unpublished material, 1997.

[Stone, Veloso-97b] Peter Stone, Manuela Veloso, “Task Decomposition and Dynamic Role Assignment for Real-Time strategic Teamwork”. Unpublished material, 1997.

[Stone, Veloso-98] Peter Stone and Manuela Veloso, “A Layered Approach to Learning Client Behaviors in the RoboCup Soccer Server”. Unpublished material, 1998.

[Tambe et. al.-97] Randall W. Hill, Jr., Johnny Chen, Jonathan Gratch, Paul Rosenbloom, Milind Tambe, “Intelligent Agents for the Synthetic Battlefield: A Company of Rotary Wing Aircraft”, AAAI/AAI 1997: 1006-1012.

[Tzikas-97] Ioannis Tzikas, “Agents and Objects: The Difference”, M.Sc. Thesis, DSV, May 1997.

[Veloso, Stone, Han-97] Manuela Veloso, Peter Stone and Kwun Han, “The CMUnited-97 Robotic Soccer Team: Perception and Multiagent Control”. Unpublished material, 1997.

[Wellman-96] Michael P. Wellman, “Chapter 4: Market-Oriented Programming: Some Early Lessons”. In Clearwater, S. (ed.) Market-Based Control, 74--95, World Scientific, 1996.

[Wooldridge,Jennings-95] Michael Wooldridge and Nicholas R. Jennings, “Intelligent Agents: Theory and Practice”. Knowledge Engineering Review, Volume 10 No 2, June 1995.

[Åberg-98] Helena Åberg, “Agent Roles in RoboCup Teams”, M.Sc. Thesis, DSV, February 1998.