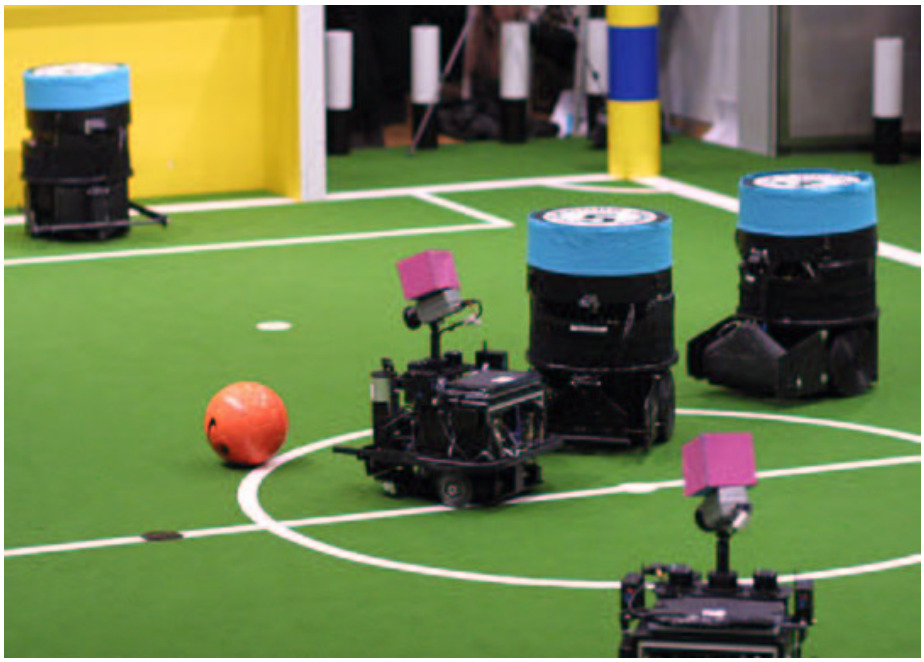
Evaluation of the world model of Clockwork Orange

Master's thesis Artificial Intelligence

University of Amsterdam



Raymond Donkervoort

July 12, 2002

Abstract

The main objective of this thesis is to describe the world model of Clockwork Orange, the Dutch Robot Soccer Team, and to evaluate its performance. After a short introduction in autonomous agents, the team will be introduced, giving an overview of its design and the tasks of the different modules. In describing the world model, special attention is paid to the important problems encountered when building a model of the world. The most important aspects are self-localization, object tracking and the sharing of the world models between the different agents. While dealing with these problems one has to take into account that we are acting in a real world domain, so we have to deal with uncertainty in sensor information.

Self-localization is done by a combination of the odometry sensor and a vision-based self-localization method. A weighted average of the two sources of information is used to estimate the new position. For the object tracking, based on observations made by the vision system of the robot, a linear least squares algorithm is used, which enables us to do noise-filtering and lag-compensation. The communication system enables us to share the local sensor information with the other robots after having converted it (using the self-localization information of the robot) to world-relative coordinates.

The evaluation of the performance of this world model is based upon the results of the team during the RoboCup 2001 in Seattle and some testing done on the field at the University of Amsterdam Robot Lab. The results show us that the world model gives an accurate and complete representation of the real world. The self-localization however turned out to pose some problems. Odometry information is accurate on short distances only. The vision-based self-localization turned out to be too inaccurate and arrives too infrequent for the robot to continuously have a good estimate of its own position.

Acknowledgements

There are a few people I would like to thank for their support during the months of my graduation work. In the first place I would like to thank my supervisor Frans Groen for his guidance during the work on the software as well as during the writing of my thesis. Furthermore I would like to thank all the people of the ClockWork Orange team. To begin with Jeroen Roodhart, for his help during the first stage of my graduation by helping me undertand the complex material of his world module. Next the guys from the Delft University for the good cooperation and pleasant company during the preparation to the RoboCup 2001 tournament and the tournament itself. Most importantly I would like to thank the guys from Amsterdam, especially Matthijs Spaan and Bas Terwijn for making the time I spent with them in the Robot Lab a great time. Finally I would like to thank my family and friends for their support throughout the years.

Preface

Chapters one and two are a result of a collaboration between Matthijs Spaan, Bas Doodeman and myself.

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.

We hope 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 software agents. 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 industrial products, intelligent transport systems, bringing medicine to patients in

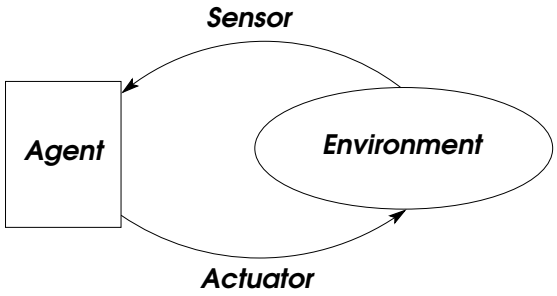


Figure 1.1: An agent interacting with its environment.

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 are examples of that.

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 the 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:

- Since multi-agent systems are more modular than single-agent systems it is easier to change a specific part of the system. Instead of having to change the entire system, just the involved agent has to be adjusted.

- 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 does make it harder to form this model since all the data has to be fused 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 the others 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 agents 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 very 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 by 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 vs. 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 necessary, but in others it is better not to use it since it is liable to interference.

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

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

dynamic vs. 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 vs. 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 vs. 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 or in situations which consist of lot of dependencies 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, like robot soccer.

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 systems approach 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 [11, 12].

1.3 The RoboCup project

In the year 1997 the Robot World Cup Initiative ([18, 22] 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 the game of soccer. A game of soccer contains most of the important aspects that are present in a real world multi-agent application. 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 (the rules of the middle-sized league will be described in the next section). 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 participated in this league [4].

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 using the color coding of the ball and the different robots a world model is constructed. 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 participates. This team will be described in the next chapter. 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 the 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

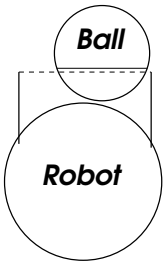


Figure 1.2: The ball handler rule.

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 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 the Clockwork Orange participates is the middle-size league. Unlike the Sony four 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 (having extendible and retractable actuators and thus having multiple possible configurations) is not allowed to have a possible configuration in which its projection on the floor does not fit in a 60×60cm square. A robot should also have a minimal 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. Before the start of each game the team leaders and the referee will decide which team will play with which color.

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

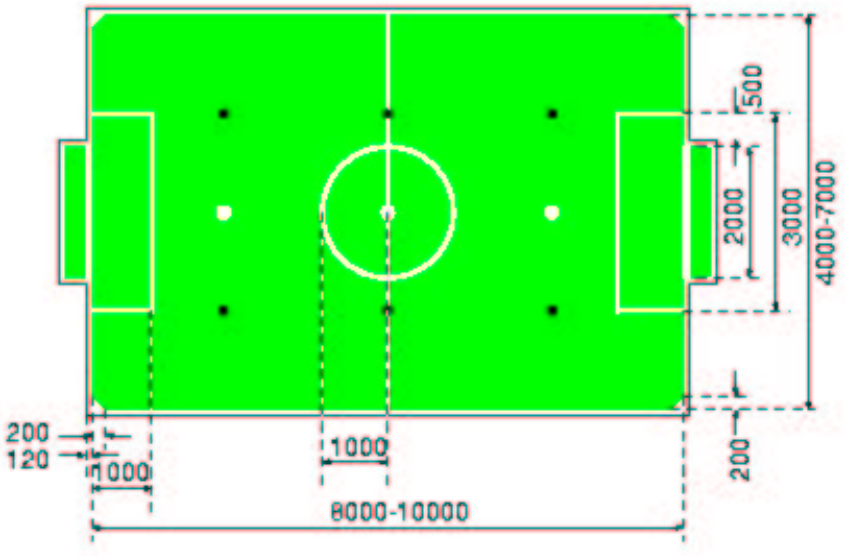


Figure 1.3: The sizes and lines of a robot soccer field

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 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 opponents 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 if necessary change the batteries and fix their robots. The complete FIFA and RoboCup rules can be found in [19].

1.5 This thesis

The main objective of this thesis is to study the performance of the world module (i.e. the module responsible for maintaining the world model) of Clockwork Orange, the Dutch Robot Soccer Team and to come up with possible improvements. The design of the world model and the used algorithms will be described and all the problems encountered when constructing a world model in a flexible real world domain will be discussed. Special attention will go to the key aspects such as self-localization, object tracking, dealing with uncertainty and distribution of the world model.

In chapter 2 a general description of the team will be given. Both the hardware specifications of the robots and the software architecture of the team will be described. World models in general and some related work will be discussed in chapter 3. In chapter 4 a detailed description of the world module of Clockwork Orange will be presented, complete with the chosen software architecture and the algorithms that are used. Furthermore chapter 5 will give the results of the world module as we used it during the last RoboCup tournament in Seattle. In chapter 6 some improvements that could be used will be proposed, results of these methods will be given and a comparison will be made with the results of the current world module. The final chapter will consist of the conclusions and a proposition of some future research topics.

Chapter 2

Clockwork Orange

Clockwork Orange [10, 25] 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 world 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.

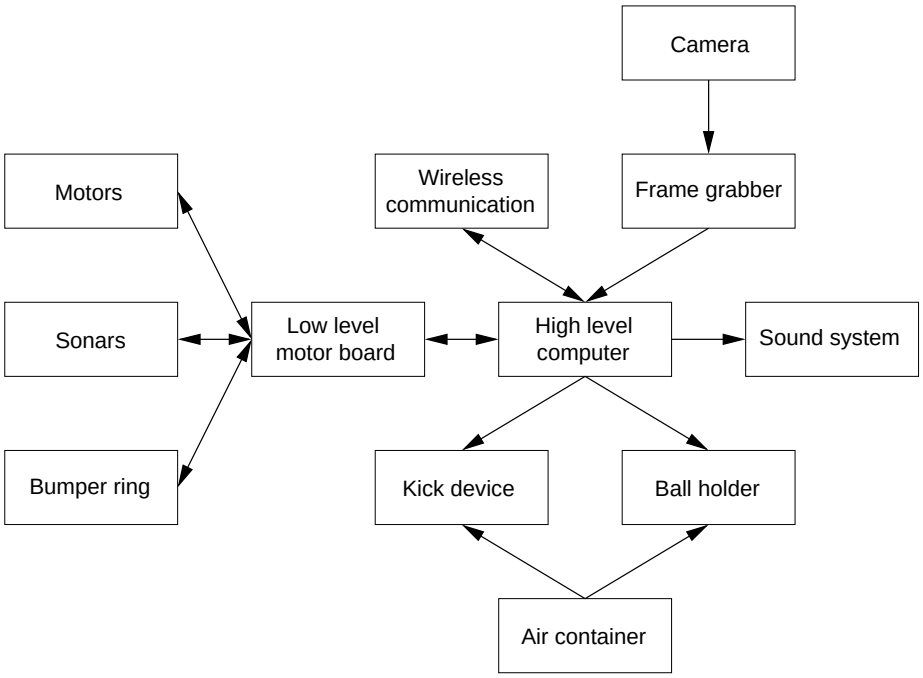


Figure 2.3: Hardware setup of our Nomad Scout.

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 the low motor board

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.

to malfunction.

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 [7] 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 within this band 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 and world models 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 [8] 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 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.

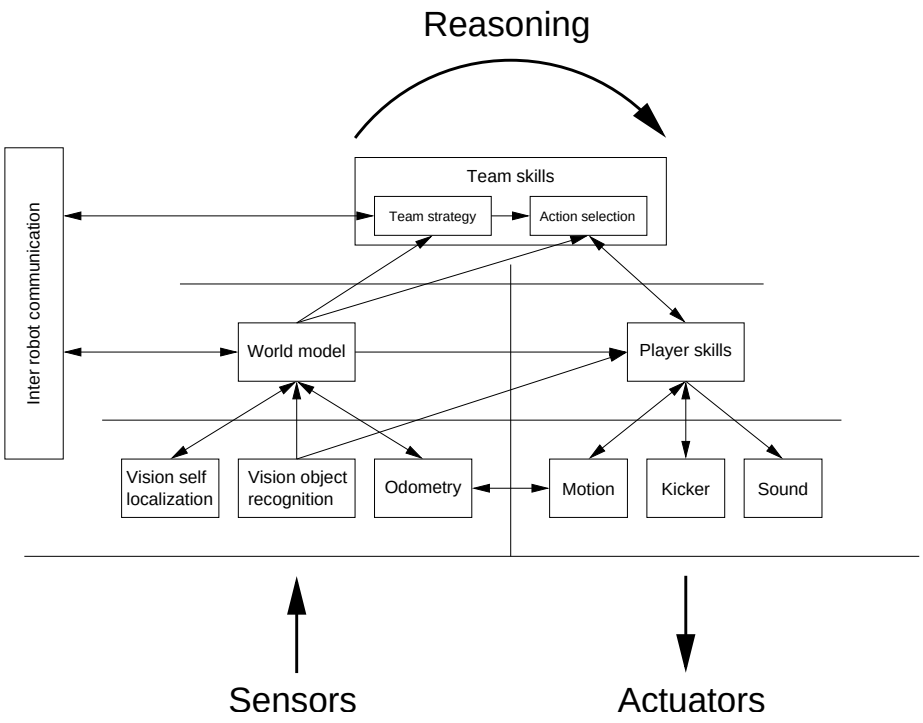


Figure 2.5: Functional architecture decomposition.

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 adjust a direct move command from the Team skills module when an obstacle appears. Another example is executing a dribble with ball action, during which fast movement corrections based directly on vision information have to be executed.

One should keep in mind the architecture has been designed a few years ago ([15], short version appeared in [17]) 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 [5] has been run to estimate the systematic error, which increases with the traveled distance. For a typical Nomad this error turned out to be 15 cm after driving a 3 m square (1.2%).

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 [16] 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 with sufficient intensity. 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 [16] 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** [14] 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 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 and the world module are 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 chapter 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.

[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.

The **Kicker** module controls the pneumatic kick mechanism used for shooting 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 takes care that 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 current orientation of 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 thinks it will lose the ball while turning it will shoot it at that time.

Goto(x,y,ϕ,v), move to 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. 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 of the ball handling mechanism 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** is also distributed. Each of the robots in our team locally maintains a world model which is shared with the other robots in our team. The design of the world model is described in chapter 4. The performance of the world model is discussed in chapter 5.

The task of the **Team skills** module is twofold: it coordinates team strategy and it chooses the next action the Player skills should execute. The advantage of an absolute shared world model is that team coordination is greatly simplified. We use a global team strategy from which each robot derives its individual role.

The team skills module determines the current team strategy using a simple finite state machine. Input of the state machine is ball possession: which team, if any, controls the ball. It outputs the applicable team strategy: *attack*, *defend* or *intercept*. A team strategy is a distribution of roles over the available field playing robots. The roles associated with a team strategy have a certain priority to be able to deal with the possibility of not having three field robots at your disposal.

There are about half a dozen different roles available, all of which have an attacking, defending or intercepting purpose. Utility functions are used to distribute the roles among the team members. The utility functions are based on the time a robot estimates its needs to reach the ball besides a position based role evaluation. The first criterion is applicable for offensive roles while the latter is suited for defensive ones. This dynamic role distribution technique is similar to the ones used by other participants in the middle-size league [9, 29], but we have extended existing approaches by adding a global team strategy.

Assigning roles to robots only makes sense if the robot takes its role (and possibly those of its teammates) into account when selecting the next action it should take. It should execute the next action which benefits the team the most, and its role provides the robot with a description of what the team expects of it. Our action planning approach is similar to the one Tambe [27] described for use in the simulation league.

Action planning is modeled using Markov decision processes and the role of a robot determines its action space and influences its reward function. In order to be able to find a good solution to the Markov decision problem we discretize the action space (instead of the state space which remains continuous), which means a robot considers only a finite set of actions at a time. Actions are defined as having a certain type like move or dribble and certain parameters like target positions, whose number is potentially infinite. As a solution to this problem we only consider a finite number of target positions. The size of the set of actions lies in the order of magnitude of 50. The role of a robot determines the contents of this set: a defensive role will lead to more move actions than shoot or dribble actions while an offensive role will contribute more shoot and dribble actions than plain move actions.

Our reward function is designed as follows: estimate the desirability of the current world state by looking at several soccer heuristics, simulate the action on this world to obtain a new world state, estimate the desirability of this new world state, and the difference between the two estimates is the reward. One of the four soccer heuristics is the position based role evaluation used in the role distribution mechanism. So to what extent an action is considered beneficiary to the team partially depends on the robot's role. The other three heuristics are about whether the ball is in one of the goals, ball possession and strategic positioning. A detailed description can be found in [24].

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 [28] 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, each 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 would, however, 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.

The mechanism we use for the triggered notification communication type in some aspects resembles a subscription mechanism. 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

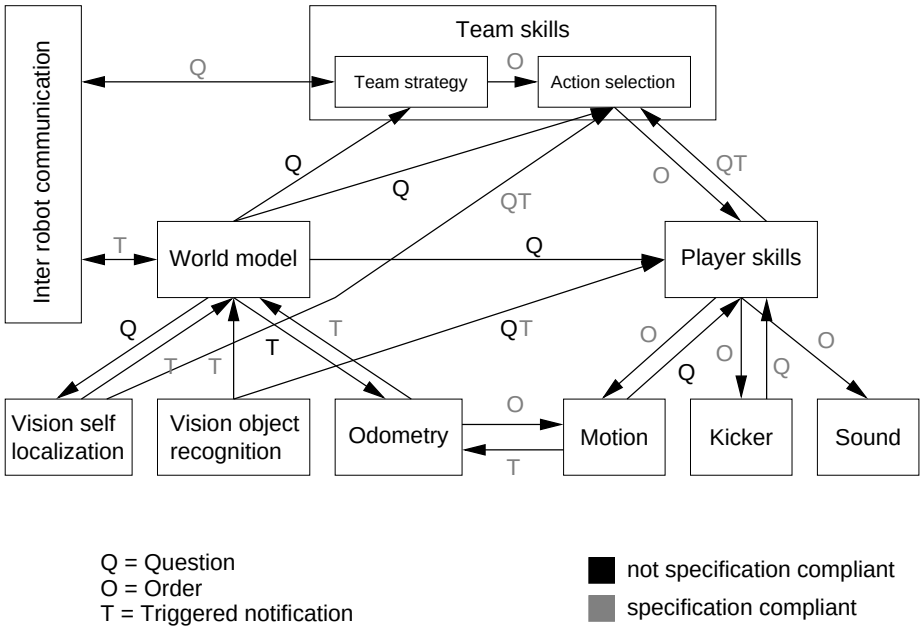


Figure 2.6: Communication details. The directions of the vertices correspond to the directions of the information flow.

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 [28] 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

World Models

3.1 A world model

In order for any agent to make sensible decisions about the actions it should take, it should have some information about its environment. In the domain of robot soccer this means that any player should know where the ball is (or try to find it when this is unknown) and to score a goal, it would also be useful to know where the opponents goal is. To be able to perform some kind of cooperative team behavior it is also necessary to know where the other robots on the same team are, and finally to introduce tactics it is also necessary to know where all the opponents are. As we can see here, if the complexity of the desired behavior increases so does the need for more complete information about the environment. Every robot has to store this dynamic information internally in a so called world model. This model of the world will typically contain the objects in this world with their features, such as the position, orientation, speed and some form of classification. It is desired that the world model is as **complete** as possible, meaning that as much as possible of the relevant objects in the world are represented in the model. Most of the time it isn't possible for any player to completely perceive the entire game because some of the objects will be occluded by other object or are outside the field of vision of the robot's sensors. Another important aspect of a world model is its **accuracy**. We of course want the state estimates of the objects in the model to be as accurate as possible. Because of the inaccuracy of the sensors and noise present in the measurement this will require special filtering techniques. In order to create an up-to-date model of the world we also want the delay of the dynamic information to be as small as possible. This will be achieved by using frequent updates using new sensor data and by using a system that is capable of processing this data fast enough or by designing efficient software.

3.2 Design decisions

When designing the software that is used to create a world model several decisions have to be made.

3.2.1 The sensors

The design is partly influenced by the sensors the robot possesses. The sensors are used to detect and recognize the other objects in the game and to perform self-localization. Some of the sensors often used in the robot soccer domain are:

- *Odometry.* The odometry sensor counts the number of rotations of the wheels of the robot using an absolute encoder. This sensor is used by every robot on every team since it is a very cheap sensor which is very reliable for a short period of time. However it doesn't give the absolute position of the robot.

- *Camera.* A color camera is often used to detect the objects in the game, which all have their specific color code (e.g. the ball is orange). Some teams (including the Clockwork Orange team) also use the camera for self-localization purposes.

- *Omni-directional camera.* Self-localization can be done more easily with an omni-directional camera. A conic mirror hanging above the camera gives it a 360 degree angle of view, allowing it to see the entire field (as far as it isn't occluded by any objects). This way the robot will almost always see distinctive landmarks such as the goals.

- *Sonar.* Sending ultra sound waves in up to 360 degrees this sensor scans the field for objects. This sensor will give a reasonably accurate model of the field.

- *Laser range finder.* Using a laser to scan the field will give a very detailed and accurate model of the field, which makes it an extremely useful sensor for self-localization purposes [1]. However laser range finders tend to be very expensive. When a system uses this sensor to detect the objects in the game it has to distinguish these objects by shape, since this sensor won't perceive the colors. The laser range finger also only gives an intersection of the world at a certain height, making occlusion an even greater problem than it is when using a camera.

- *Infra-red sensor.* This sensor uses infra-red to accurately determine the distance to objects.

Each of these sensors has its pre's and con's as mentioned above. In the robot soccer domain self-localization and object detection are the most important problems. For the self-localization a combination of odometry and a laser range finder would be ideal (provided that there are walls surrounding the field). The odometry could keep track of the position for short periods and at a given interval the laser range finder could give an absolute update of the position. A camera is preferable to do the object detection. A normal camera is more accurate but a omni-directional has a larger angle of view, however both are usable to do the object detection. An omni-directional camera also is very useful for self-localization, since the omni-directional field of view will increase the chance of the robot seeing one of the goals making it easier to perform its

[1] The absence of walls around the field as prescribed by the new 2002 rules will make it much harder to perform self-localization using the laser range finder.

vision-based self-localization. Since some of these sensors (especially the laser range finder) are quite expensive most teams will have to make do with a less than optimal configuration of sensors.

3.2.2 Sensor data tracking

From the data of these various sensors a world model has to be constructed taking into account the uncertainty of the observations, the noise and possible delays. It is up to the software constructing the world model to do noise filtering and lag compensation. Noise can be filtered out by using stored past observations of the object and assumptions about the objects motion. For instance using the linear least squares approach (described in [11] and [21]), a line is fitted through these past measurement and the new measurement, using their error estimates to filter out noise as much is possible. For this approach we assume that the data can be represented by a linear function.

Another, more dynamic way of updating estimated states of objects and filtering out the noise is the well known Kalman filter (described in many books and articles, for instance [11] and [13]). The Kalman filter is a recursive, linear estimator. The filter is supplied with initial information, including the measurement error covariance, and estimates of the initial parameters and associated error, and these are used to calculate a gain matrix. The error between the parameter estimates and the measured data is determined and multiplied by the gain matrix to update the parameter estimate and estimated error. The updated error and parameters are used as input to a model of behavior, to predict the projected error and parameters at the next time instance. Initially, when the model parameters are only rough estimates, the gain matrix ensures that the measurement data is highly influential in estimating the state parameters. Then, as confidence in the accuracy of the parameters grows with each iteration, the gain matrix values decrease, causing the influence of the measurement data in updating the parameters and associated error to lessen. One of the great advantages of the Kalman filter over other filtering techniques is its ability to incorporate the effects of noise from both the modeling and the new measurements.

3.2.3 Shared worldmodels

As mentioned before, a robot will often perceive only a small part of the entire game, due to the limits of its field of view and parts of the field being occluded by objects in the field. The completeness of the world model could be enormously increased by also using the observations of the other robots in the team to construct the world model. A wireless Ethernet system enables the robots to communicate with each other, also allowing them to share their information about the world. This information can either be an entire world model or just observations or state estimates of individual objects in the game. Sharing information enables the robots to also keep track of the objects in the game which aren't perceived by the robot itself but only by its teammates. Furthermore shared information can be used to increase the accuracy of the state estimates. In order to share a world model it has to be world-relative instead of ego-relative. To convert the relative coordinates of the observed objects into absolute coordinates the robot's own position is needed.

3.2.4 Self-localization techniques

There are many different methods for self-localization of course depending upon the used sensors. The easiest method for self-localization is using odometry. The great disadvantage of an odometry sensor however is that its position estimates are based upon the number of rotations of the wheels and not upon the real traveled distance. When the wheels skid or the robot collides and moves without turning its wheels, this will result in the odometry position estimate losing its correct position. Furthermore odometry sensors typically come with a systematic error. This means that from time to time the information about the robot's position has to be updated by an absolute self-localization method. There are various different possible self-localization methods, largely dependent upon the type of sensor that is used. Since our team only uses the vision system to perform this task, only the vision based self-localization techniques will be described. These can roughly be divided into 3 categories:

Feature based. This approach consists of extracting certain features from the grabbed images and using them to estimate the robot's own position. For instance when the image shows us two goal-posts and we can accurately determine the positions of these goal-posts and a corner-flag with respect to the robot, we can use the triangulation technique to determine an approximation of the robot's position.

Edge based. Instead of using the objects in the game other features of the field could also be used. This could be the lines on the field, maybe even in combination with the walls around the field and the goals. By iteratively trying to match the perceived lines to an estimated model of the world, the robot's position could be calculated. Clockwork Orange, the Dutch robot soccer team also uses an edge based approach for self-localization (described in [14] and summarized in [25]).

Appearance based. Opposite to the other approaches, using the appearance based approach no features are extracted from the image. The entire image or (to increase efficiency) a lower dimensional representation of the entire image is used. This image is compared with the images the robot has in his database to do an estimation of the position. An omnidirectional camera is highly desirable for this approach.

On deciding which method to use in the robot soccer domain, one has to take into account the amount of information that will typically be found in an image captured by the vision system while playing a game of soccer. Since the number of easily detectable static objects required for the feature based approach is very limited (only goal-post and corner-posts can be used in current field configurations), this approach is not very suited for robot soccer. The edge based approach is very suitable for robot soccer since the robot will almost always perceive some lines of the field which can be matched to a relatively simple model of the field. However symmetry of the field will, when depending solely on the lines, result in at least two potential position candidates being found. To evaluate these position candidates one has to use other information (e.g. in which direction a goal is seen) or use some kind of tracking method. The appearance based approach is also usable in robot soccer. However the highly

dynamic nature of this domain will make it much harder to use because of the movable objects on the field disturbing the images.

3.3 Approaches of other Middle-size league teams

The teams differ largely in the way they construct a model of the world. In this section some examples of used methods by the different teams are given. The information presented has been taken from the 2001 team description papers (except for ART, which has been split into several teams), in which references to information about their specific approaches can be found. Table 3.1 lists the institute, robot base on which their robots are built and the sensors of each team. The questions whether or not the teams are homogeneous and if they communicate are also answered.

CS Freiburg [30] Each robot builds and maintains a local world model on the basis of its own sensor readings (a.o. a laser range finder). This model consists of the position, speed and heading information of all the objects in the game. The local world model is extended by the results of a global fusion component that runs on an off-field computer and combines all estimates from the players. Observations from different robots with regard to the same object are fused using Kalman filtering. In addition for the global ball position, a probabilistic approach known as 'Markov localization' is employed to exclude observations from sensor fusion which are most probably entirely wrong.

ART [2] A Hough transform based self-localization technique is used, detecting landmarks on the field (the positions of which are know) and based upon that estimating the pose of the robot. A Kalman filter is then used to combine this state estimate with the odometry information. Local world models are communicated with the other robots on the team.

AGILO RoboCuppers [23] A probabilistic vision-based self-localization method is used to estimate the state of the robot. This method also enables the robots to track the positions of the other moving objects. Local world models are shared with teammates to increase accuracy.

GMD [6] This team uses revolving cameras to perceive the world. Although having only a limited field of view the cameras can, being revolving, be used to see in all directions without the robot having to change its orientation. This camera information is used to form a model of the world and estimate the absolute position of the robot, using the landmarks on the field. The local world models are shared with the other robots on the team.

CoPS [20] The architecture of CoPS is structured in a reflexive, a tactical and a strategical layer. The reflexive layer is responsible for sensing and interacting with the hardware. The the level of the tactical layer, the Scene Detector agent stores and updates the state of the world. This model is based upon both local sensor information and communicated information of the teammates. Finally the strategic layer is used to control the robots in special occasions such as the

(re)start of the game. Self localization is based upon scans of the field with a laser range finder and sonar.

Trackies [26] Every robot has a reactive behavior, which is based upon a relative world model. The robots don't communicate which each other, so the local world models aren't shared. There also is no form of team behavior, since the robots don't know what the other robots on the team are planning to do. Self-localization is done using omni-directional cameras.

Most of the teams use an approach in which the self-localization has a crucial role. Knowing the robot's position, an absolute model of the world can be constructed, which could potentially be shared with the other robots. This also facilitates the cooperative team behavior. The main drawback of this approach is that it relies to heavily on the robot accurately knowing its own position. The approach the Trackies use relies much less on the self-localization, the robot's behavior is largely based on a relative model of the world. However the lack of communication between the robots will result in a much less accurate and less complete world model and make team coordination almost impossible.

Team	Institute	Robot base	Sensors	Homog.?	Comm.?
CS Freiburg	Albert-Ludwigs-Universität Freiburg	Pioneer 1	laser range finder, camera	yes	yes
ART	Univ. di {Parma, Padova, Genova, Roma "La Sapienza"}, Politecnico di Milano	Custom, Pioneer 1	camera, sonars, infrared sensors, omnidirectional camera	no	yes
AGILO	Technische Universität München	Pioneer 1	camera	yes	yes
GMD	GMD	Custom	360° panning camera, gyroscope, infrared sensors	yes	yes
CoPS	Universität Stuttgart	Nomad	laser range finder, camera, sonars	yes	yes
Trackies	Osaka University	Custom	camera, omnidirectional camera	yes[2]	no
Clockwork Orange	Univ. of Amsterdam, Delft Univ. of Technology, Utrecht Univ.	Nomad, Pioneer 2	camera	yes	yes

Table 3.1: Comparison between several middle-size teams

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

3.4 Discussion

Now that an overview has been given of the options one has when designing a world model for a multi-agent system in general and a robot soccer team in particular, let's take a look at the choices the Clockwork Orange team made. An absolute (world-relative) world model was chosen to facilitate cooperative team behavior and to allow the robots to share the information they have about the world in order to increase the accuracy and completeness of all of their world models. In order to construct a world-relative model it is necessary to know the robot's own position. A vision-based approach was chosen to perform self-localization. Given the fact that the field is surrounded by walls, a laser range finder would have been easier, but the fact that these walls were to be abolished at some time in the future and the fact that a laser range finder is very expensive made that a vision-based approach was chosen. A feature based approach turned out to be unsuccessful, because most of the time too little features are visible. Currently the team uses an edge based approach, which works quite good, but still is far from perfect. The appearance based approach will probably be a good alternative. The performance of the self-localization would certainly get much better if the team started to use omni-directional cameras. Most other teams use laser range scanners or omni-directional cameras and seemingly have less trouble in performing self-localization. A normal camera is used to detect the other objects in the game. It gives us accurate estimates of the positions of the objects. An omni-directional camera would considerably enlarge our field of view (we could detect the ball lying behind us) but would also be much less accurate. All the sensor information is processed using a linear least squares filter which makes us less sensitive to noise. Our communications system enables us to share the local world models with the other robots in the team, which also increases the accuracy and completeness considerably.

Chapter 4

The World Module

In this chapter the world module of Clockwork Orange, the Dutch robot soccer team will be discussed. First the way we represent the real world will be presented. In the following sections the algorithms used to create and update a world model and some of the other features of this module will be described. Special attention will be given to the self-localization because of the importance of this aspect in building an absolute world model.

4.1 A model of the world

The first thing one has to decide when building a model is how to represent the real world in the model. We chose an object-oriented approach in our software architecture (all software is written in the C++ programming language). This object-oriented approach can also be seen in the way we represent the real world in our world model. To make the fundamentals of the world model usable for other applications as well as robot soccer, we use a *generic world* object. As could be expected the world consists of a list of objects, the so called *world objects* which can be divided into *movable objects* (which can move around in the world freely) and *fixed objects* (which have a fixed position). All the objects are given their own unique ID-number (ranging from 1 to the number of objects in the world) which is used to identify them in the model representation, this number however is only used internally and is of no interest to the other modules. For the specific robot soccer domain we use a sub class of the generic world, the so called *soccer world*. The main difference between these two classes is that the soccer world also consists of lists defining which objects belong to which team and special lists for soccer related objects such as the goals and the lines of the field. Furthermore the soccer world consists of a list of the objects that are tracked and a list of the observations which aren't matched to an object. The way these lists are constructed and which purpose they serve will be described later on in this chapter.

All the objects in the world, both movable and fixed, have certain features that describe the state of the object.

- *position* Every object has a certain position (the center of the object) consisting of the x and y coordinates, the heading of the object and uncertainty estimates for these values. The coordinate system we use in

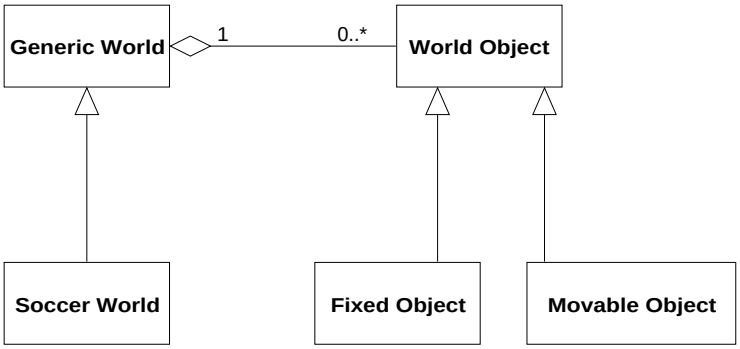


Figure 4.1: UML class diagram of the world model.

our robot soccer team is shown in figure 4.2. The x and y coordinates are measured in millimeters and the heading is measured in units of 0.1 degrees.

- *speed* The movable objects also have a speed, consisting of a velocity in the x direction and in the y direction. Furthermore every movable object has an angular speed. Speeds are measured in millimeters per second and 0.1 degrees per second.
- *color* Each object has a certain color, for now this feature isn't used in this module, but it could in the future be used to distinguish teammates from opponents by the color of their markers.
- *shape* The shape by which each object is detected. This could be shapes like ORANGE_BALL, BLACK_ROBOT etc.
- *classification* Every object in the game has an unique classification. Robots are classified by their player number and the team they are on, resulting in names like OT_PLAYER_1 (meaning player number 1 of our team).
- *size* The length, height and depth of the object, measured in millimeters. This can be used for collision avoidance. When we know how large an object is we know how far we have to stay away from the center of the object in order to avoid collision.
- *validation mask* Since we're dealing with a lot of unknown or uncertain information it is necessary to keep track the information that is valid and the information that is invalid. The validation mask tells us for all the features of an object if they are valid. For instance using only one vision observation it is possible to estimate the position of the object but it's impossible to determine the objects speed. The validation mask of this object will therefore indicate that the position information of the object is valid but the speed estimate isn't.
- *activity flag* For movable objects in a game of soccer this flag tells us whether an object is participating in the game. This feature is necessary since any player can be substituted or banned from the game, as a result of a blue card, and can thus be out of the game for some time.

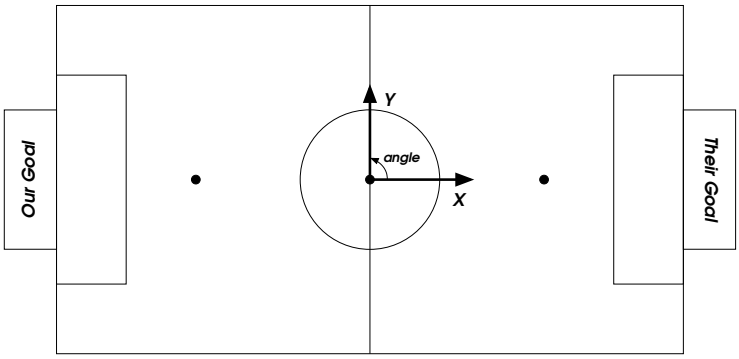


Figure 4.2: The coordinate system

- *latest update time* Every time we receive an observation of an object we update the position information of this object, the moment this happens is known as the latest update time. We use this time for position predictions and tracking purposes.

The world model is based upon observations of the objects in the world. This information comes from the virtual sensor modules in the form of a so called measurement data-structure. This data-structure consists of position, shape, color and optional classification information. Because of the possible delay in our communication system it is necessary that every observation comes with a certain time-stamp telling us precisely when the object was observed.

Because the world module has to handle a lot of incoming information as well as use this information to update the model and fulfill the information requests of other modules, a multi-threaded approach has been chosen. The main thread is the receiver thread which handles all incoming observations. After some simple operations on this information a different thread uses this information to update the world model. Other threads are used to handle data subscriptions and deal with the more complex information requests. This approach is necessary since the world module should to be able to operate in the very dynamic domain of robot soccer, where it is essential to have an as up-to-date model of the world as possible. Whilst increasing speed a major drawback of multi-threading is that it makes the design of the module more complex because of the risk of death-locks.

4.2 Locating and tracking objects

The information the world module receives about the world comes from different origins. First we have the odometry sensor which gives us the robot's own position in absolute coordinates. The world module subscribes itself to observations of this module. This means that the odometry module will send the latest information it has about the position to the world module at a given interval. The next local source of information is the vision module. The world module also subscribes itself to information from this module, so that at a certain interval the vision module will send all the shapes detected in the latest grabbed frame. Another possible source of incoming data are the other robots in the team. The

way robots share their world models will be described in section 4.3. All the information originating from these sensors is handled in the same way, which is described in section 4.2.3 and further. The final source of information for the world module is the vision-based self-localization, the way this information is processed is described in section 4.2.2.

4.2.1 Dealing with uncertainty

Before going any further into the way the new observations are used to form a model of world, first lets take a look at the aspect of uncertainty estimation. Because real world sensor data typically comes with some noise and isn't always very accurate it is necessary to keep a measure of the accuracy of the estimated states. In the equations that will be encountered throughout this chapter the uncertainty estimates will be denoted as Cx, Cy and $C\theta$ indicating respectively the uncertainty in the x-direction, the y-direction and the orientation. It is up to the virtual sensor modules to determine the value of these uncertainties. The values are based upon an error model for the corresponding sensor. The parameters of this model are estimated based upon experiments. Both in vision and odometry one can let the sensor do a number of position estimations and from these measurements calculate the typical standard deviation to the real position. For instance using the UMBenchmark (as described in chapter 5), in which the robot drives squares and the error is determined, in order to estimate the error of the odometry. The odometry sensors estimate the uncertainty to be about 5% of the traveled distance. The vision system calculates the uncertainty in the observation of an object using the distance to the object and the angle in which it is observed. The final source of local sensor information, the vision-based self-localization doesn't provide a correct uncertainty estimate. It always assumes its position estimate to be correct within an uncertainty region of about 10 cm. Chapter 5 will show us that much of the updates done by the vision self-localization will be less accurate than this 10cm error region, so a better uncertainty estimate would be very desirable. Also when using the new observations to update the world model, one has to take into account the uncertainty estimates, and also update these values.

4.2.2 Self-localization

The robot measures the objects in the world with respect to itself. To share information about the world with other robots it is necessary have an absolute model of the world instead of a relative one. When constructing an absolute world model from relative sensor information it is necessary to know the robot's own position. The robot's own position information is based upon two different kinds of sensors. The first one is the odometry sensor, which will give us an absolute position by determining the traveled distance from a known point. However the position estimate of the odometry is based upon the number of rotations of the wheels and not the real traveled distance, meaning that it would completely lose its value when the robot collides, skids or is picked up and placed somewhere else. Furthermore the systematic error which typically comes with any odometer causes the accuracy of the position estimates to decrease as the traveled distance increases making it useful for short distances but much less useful for use over longer periods of time(more detailed information about this

can be found in chapter 5). The other source of position information is the vision-based self-localization. As described in [14] (summarized in [25]) this method uses the lines on the field to estimate the position of the robot. The great advantage of this method is that it will give us an accurate absolute position estimate which isn't dependent on the traveled distance. The disadvantages are that it is a very expensive (both in time and resource) method meaning that can't be used to update the position at a high frequency and the estimated position will arrive with quite a large delay. Also this method won't always be able to tell us the position since it is very dependent on the number of lines that are observed by the robot.

In practice the robot's position will be mainly based upon the odometry information which the world modules receives very frequently and which is handled using the linear least squares algorithm. From time to time, when the vision-based self-localization is sure enough about an estimate, this will be send to the world module, which will use this to reset the robot's position to an absolute point (which, in the current implementation, is determined by combining the old estimated position and the new position estimate from the vision-based self-localization) after which the odometry will start tracking the robot again. The position estimates from the vision-based self-localization typically arrive with quite a large delay.

Each of the objects in the world (and thus also the object that represents the robot itself) has an history of the positions where it was estimated to have been so far. Every time the objects position is updated this new position is stored in the objects position history. Let these state estimates over time-span T be denoted by $\{p^{t-T}, ..., p^{t-1}, p^t\}$ where every p is a vector (x, y, θ) with an corresponding uncertainty vector (C_x, C_y, C_θ) and t is the time of the latest position estimate. When a repositioning from the vision-based self-localization $p_{ext.}$, made at time $t-\tau$, arrives, the history is used to determine where the robot thought it was at time $t - \tau$. This position $p^{t-\tau}$ is determined by interpolating between the two positions in the memory that are closest in time to this vision-based estimate. This interpolated position and the new external estimate then have to be combined, the method we use is that of the weighted average (where the uncertainty estimates are used as the weights). The following equations show how this new position estimate $\hat{p}^{t-\tau}$ is calculated, with w being the weights.

$$\hat{p}^{t-\tau} = p^{t-\tau} w_{p^{t-\tau}} + p_{ext.} w_{p_{ext.}} \tag{4.1}$$

$$w_{p^{t-\tau}} = 1 - \frac{C_{p^{t-\tau}}}{C_{p^{t-\tau}} + C_{p_{ext.}}} \tag{4.2}$$

$$w_{p_{ext.}} = 1 - \frac{C_{p_{ext.}}}{C_{p^{t-\tau}} + C_{p_{ext.}}} = 1 - w_{p^{t-\tau}} \tag{4.3}$$

The uncertainty estimates of the new combined state are determined in the same way. A weighted average of the both uncertainty estimates is used in which the uncertainty estimates themselves are the weights. Using the combined position and and the old interpolated position estimate we can calculate a position shift telling us the difference between the place the robot though he was and where he really was at that time.

$$p_{shift} = p^{t-\tau} - \hat{p}^{t-\tau} \tag{4.4}$$

Going through the position history of the object we delete all the position

prior to the external estimation time, since they have no value any more. All the newer positions are now translated and rotated according to the position shift. Now the latest position estimate will give us the current position tracked by the odometry and adjusted by the vision-based self-localization. The technique we use here is minus some minor differences comparable to the well-known Kalman filtering technique.

When we 'reset' our position this way, we should also take in mind that the observations of other objects were done relative to the robot and have been converted to world relative coordinates using our former position, which possibly was incorrect. We could shift the positions of all the observed objects corresponding to our own position-shift, but to keep thing simple, we just delete the old observations of the objects. So from then on, the position estimates of the objects will completely be based upon observations done after the resetting of the robot's position. This problem could be overcome by maintaining an ego-relative world model as well (instead of only a world-relative world model). In this ego-relative world model all the objects' positions relative to the robot are being tracked. The accuracy of these position estimates isn't influenced by the robot's self-localization.

Sadly at the time of the RoboCup 2001 in Seattle the local tracking mechanism of the vision-based self-localization couldn't cope with significant position-shifts so we used the external position estimate as the combined measure (in the given equations, this would mean that the weight of the external estimate is 1 and the weight of the original estimate is 0).

4.2.3 Conversion to absolute coordinates

Now that the robot knows its own position it can use this to convert the incoming observations, originating from a relative sensor, to an absolute coordinate system. When a new observation arrives at the world module it will first be checked on its time-stamp. When the observation is too old, which probably means that the message system had some problems sending the observation, it is useless for creating a world model and is therefore disregarded. When accepted the relative observation can now be converted to absolute world coordinates using the following equations, where P^C is the position in the camera coordinate system (i.e. the position relative to the robot), P^M is the position in the world model (i.e. the absolute position), θ is the orientation of the robot and t is the absolute position of the robot.

$$P_x^C = P_x^M \cos\theta - P_y^M \sin\theta + t_x \tag{4.5}$$

$$P_y^C = P_x^M \sin\theta - P_y^M \cos\theta + t_y \tag{4.6}$$

The uncertainty estimates of the observation also have to be updated when converting the position to world-relative coordinates. The uncertainty about the robot's position has to be added to the uncertainty about the observation in order to come up with the uncertainty about the position of the object in the absolute world model. We now have an observation in absolute coordinates which we can test on being within the boundaries of the field. Since all objects in the game should also be positioned on the field we can assume that any observation which is way out of bounds (more than 2 meters) will be due to

noise an can therefore be disregarded. Another possibility would be that the observation is out of bounds because the estimated position of the robot itself is wrong. When encountering a lot of these outliers it would be reasonable to decrease the certainty about the robot's position. However at this time this has not yet been implemented. Having converted and filtered the incoming observations, we can use them to update the corresponding objects.

4.2.4 Matching observations to objects

Unlike the observations done by the odometry, vision-based self-localization or other robots, the observations originating from vision will probably lack a valid classification. The vision module can only detect the so called shapes of the object in a grabbed frame (more information about the vision system can be found in [16]). Actually the vision system only detects the colors, however due to the chosen terminology in the design of the team, these distinctive features will be referred to as shapes. These shapes could be the things typically found in a game of robot soccer like, BLACK_ROBOT, ORANGE_BALL or fixed objects like BLUE_GOAL. It is up to the world module to match these shapes to any of the objects in the game. The ORANGE_BALL shape is easy to match since there only is one ball object. However there is a possibility that the observation is wrong due to noise within the vision system or because an object with a color closely resembling the color of the ball is on the field (e.g. brown shoes). Therefore we have to determine whether the observation could possibly be the ball. To do this we have to check if the uncertainty estimate of the new observation has an intersection with the uncertainty region of the ball object in our model, taking into account the predicted motion of ball in the time between the old position estimate and the observation. The uncertainty regions are rectangular areas. Because the vision system isn't able to determine the orientation of the detected object we don't use this information in the matching process. This results in the following equations where Δt is the time difference between the position estimate of the object and the new observation, v_x and v_y the velocity of the object in the respectively x and y direction and Cx and Cy the uncertainty estimates:

$$x_{obj} + v_x\Delta t - Cx_{obj} - Cx_{obs} \leq \quad x_{obs} \quad \leq x_{obj} + v_x\Delta t + Cx_{obj} + Cx_{obs} \tag{4.7}$$

$$y_{obj} + v_y\Delta t - Cy_{obj} - Cy_{obs} \leq \quad y_{obs} \quad \leq y_{obj} + v_y\Delta t + Cy_{obj} + Cy_{obs} \tag{4.8}$$

The other movable object, the BLACK_ROBOT shape, is handled the same way. However it is possible that multiple objects match the observation. Therefore we have to make a list of all the tracked robots that could possibly match the new observation (i.e. they fit the rules given above). From the robots in the list we choose the one that is most likely to match with the observation. We use a greedy approach to do so. To determine how likely it is that observation matches an object we use the distance between the estimated position of the object and the new measurement. If the distance increases the likeliness that we have a positive match decreases. The measure we use to indicate how good an observation corresponds to an object is:

$$P = 1 - \frac{|x_{obj} - x_{obs}| + |y_{obj} - y_{obs}|}{Cx_{obj} + Cx_{obs} + Cy_{obj} + Cy_{obs}}$$

Another possibility is that there are no tracked objects that match the observation. It is possible that the observation corresponds to one of the objects one would expect in the game, but which isn't being tracked in the current world model. In that case we can start tracking the object using the new observation. For instance, when we are keeping track of three opponents, and we receive a new observation of a robot, which doesn't match to any of our teammates nor to any of the opponents, we can assume that the object corresponding to the observation is the fourth opponent and start tracking it. This situation typically occurs at the beginning of a game when the opponents are being observed for the first time. When the observation is matched to a certain object, it is added to the object's measurement memory (which can contain up to 6 measurements). This memory is ordered chronologically so that the first measurement is the newest and the last measurement is the oldest. When the memory is full and we receive a new measurement, we will delete the oldest one.

When all the objects one would expect in a game are being tracked and a new observation doesn't match to any, we can't use it to update any. However this doesn't mean the observation is false. It could very well be that the position estimate of one of the tracked objects is false and therefore doesn't match the observation or that there is an unexpected object on the field which doesn't normally take part in the game (for instance when the referee enters the field). It would be a waste of valuable information to throw away the observation. Observations not matching to any of the tracked object are therefore stored in an *unmatched observations list*. The observations in this list can be used for instance in collision avoidance by the player skills module. If we see another robot right in front of us but this observation doesn't fit in our world model it would still be wiser to drive around it.

4.2.5 Updating positions

Once we matched an observation to an object, we can use this observation to update the object's position. To estimate the position, we use the memory of observations we have about the object. If there are only one or two of these measurements in memory the only thing we can do is estimate the object to be at the position of the last observation. If there are more then 2 measurements in memory the linear least squares algorithm is used to estimate the objects position at the current time. Using a linear least squares algorithm means that we assume the motion to be linear (which is a good approximation over short distances).

The linear least squares algorithm tries to fit a line through the measurements in the memory. Since we are dealing with three dimensions (i.e. x-direction y-direction and time) with errors in two of these coordinates (i.e. the x-direction and y-direction) it isn't straightforward just to fit a line. We will try to separately fit two lines in respectively the (x,t)-plane (as shown in figure 4.3) and the (y,t)-plane. Since these straight-line models are first order functions this results in the following equations, where x and y are the position of the robot in respectively the x-direction and y-direction and t is the time of the observation.

$$x(t) = a + bt \tag{4.9}$$

$$y(t) = c + dt \tag{4.10}$$

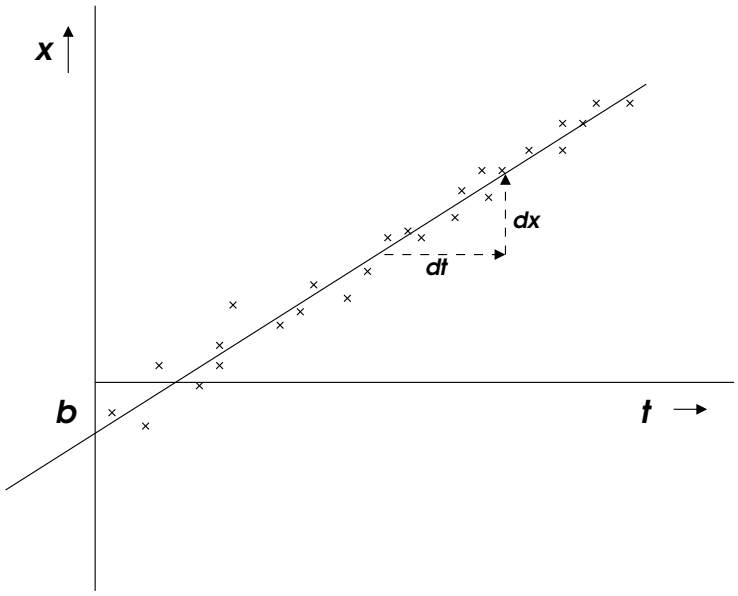


Figure 4.3: Fitting a line through data points in the (x,t)plane. Here $a = \frac{dt}{dx}$.

We will try to fit a set of N data points (x_i, t_i) and (y_i, t_i) to the corresponding line models. From this point only the (x,t) case will be described as the (y,t) case is handled in exactly the same way. In this approach it is assumed that every uncertainty estimate σ_i associated with the measurement x_i is known and that the value t_i is known exactly. As a measure to indicate how good the model agrees with the actual data the so-called chi-square merit function is used. The lower the value of this merit function the better it agrees with the data.

$$\chi^2(a,b) = \sum_{i=1}^{N} (\frac{x_i - a - bt_i}{\sigma_i})^2 \tag{4.11}$$

We want the model to be as close as possible to the actual data so the merit function has to be minimalized by adjusting the a and b parameters. To get the minimum of the merit function we use the partial derivative of $\chi^2(a,b)$. This gives us the following equations:

$$0 = \frac{\partial_{\chi^2}}{\delta_a} = -2 \sum_{i=1}^{N} \frac{x_i - a - bt_i}{\sigma_i^2} \tag{4.12}$$

$$0 = \frac{\partial_{\chi^2}}{\delta_b} = -2 \sum_{i=1}^{N} \frac{t_i(x_i - a - bt_i)}{\sigma_i^2} \tag{4.13}$$

We will now define the following sums.

$$S \equiv \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \quad S_t \equiv \sum_{i=1}^{N} \frac{t_i}{\sigma_i^2} \quad S_x \equiv \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} \tag{4.14}$$

$$S_{tt} \equiv \sum_{i=1}^{N} \frac{t_i^2}{\sigma_i^2} \quad S_{tx} \equiv \sum_{i=1}^{N} \frac{t_i x_i}{\sigma_i^2} \tag{4.15}$$

Using these definitions the equations can be rewritten to:

$$aS + bS_t = S_x \tag{4.16}$$
$$aS_t + bS_{tt} = S_{tx} \tag{4.17}$$

and given the following definition of Δ:

$$\Delta \equiv SS_{tt} - (S_t)^2 \tag{4.18}$$

The solution is given by:

$$a = \frac{S_{tt}S_x - S_tS_{tx}}{\Delta} \tag{4.19}$$
$$b = \frac{SS_{tx} - S_tS_x}{\Delta} \tag{4.20}$$

Since the measurements used to form this model of the line have some uncertainty the model should also represent some kind of uncertainty. The uncertainty estimate of the a and b parameters are given by:

$$\frac{\partial a}{\partial x_i} = \frac{S_{tt} - S_t t_i}{\sigma_i^2 \Delta} \tag{4.21}$$
$$\frac{\partial b}{\partial x_i} = \frac{St_i - S_t}{\sigma_i^2 \Delta} \tag{4.22}$$

This results in the following uncertainty estimates for a and b:

$$\sigma_a^2 = \frac{S_{tt}}{\Delta} \tag{4.23}$$
$$\sigma_b^2 = \frac{S}{\Delta} \tag{4.24}$$

Where the actual uncertainty estimate of the position is given by:

$$C = \sigma_a + \sigma_b t \tag{4.25}$$

Now that we know the model of the line we can, predict the position and the uncertainty estimate both in the x and y-direction of the object at any moment in time. The motion of any of the objects in a game of robot soccer of course seldom is linear, however over very short distances the motion can be approximated by a linear function.

4.2.6 Speed and heading estimation

Using the measurements in memory we can also try to determine the objects speed and the direction in which the object moves. Since the model of the line we use is based upon either the x-coordinate $x(t)$ or the y-coordinate $y(t)$ and the time t, the equations of the line can be formulated as follows, where v_x and v_y are the speed of the robot in respectively the x and y-direction:

$$x(t) = v_x t + x_0 \tag{4.26}$$
$$y(t) = v_y t + y_0 \tag{4.27}$$

This means that the a parameter as used in the previous section gives us the speed of the object in either one of the directions. Knowing the speed of the object in both directions also enables us to determine the heading of the robot as follows:

$$\tan\theta = \frac{v_x}{v_y} \tag{4.28}$$

The only thing left to do now is to determine the uncertainty in the heading estimate of the object. To do so we take use:

$$C_\theta = \tan\frac{v_y \pm C_y}{v_x \pm C_x} - \theta \tag{4.29}$$

Where the pluses and minuses depend upon the quadrant of the heading.

4.2.7 Tracking objects

By matching the observations to the known objects in the game and updating their positions through these new observations we can keep track of the objects. However when we lose an object out of sight it should also be known to the world module that we no longer know the object's position but that we do expect the object to still be in the game. The world module has a list of all the movable objects in the game which can be tracked, i.e. the ball, a number of opponents and our teammates who have announced themselves via a broadcast to us. When one of these objects hasn't been observed for more then 4 seconds this object is removed from the list. So the object isn't deleted and the old data isn't thrown away, but since the object isn't in the list, we know that its position information is no longer valid and we shouldn't communicate this information to other modules. When we receive an observation with the shape of this untracked object, we can use the old data we still have to determine whether the observation could possibly correspond to this object (taking into account the position we last saw it and the estimated uncertainty region and speed of the object).

4.3 Sharing the world model

One of the great benefits of our absolute world model is that it enables us to share it with our teammates. Sharing information about the world model will give us more complete and accurate information about the world. Since every robot only sees a small portion of the field it will only be able to form an up-to-date model of a very limited part of the game. By sharing the observations with the members of the same team, the robots can also include the objects in their model which they can't perceive themselves but which are only perceived by their teammates, making the locally maintained world model much more complete. Sharing also makes the model more accurate since all the shared observations can be seen as extra sensor information about the object which will help us reduce the uncertainty we have about the state of the object. When sharing the world-relative information about an object, the uncertainty estimate of the object consists of the uncertainty of the vision observation and the uncertainty of the position of the observing robot.

The sharing of information can be done in two ways (which both were used at the same time during the RoboCup in Seattle). The first and most precise way is to subscribe to each others information. When a robot enters the game it will send a broadcast to all the robots in the same team telling them it is joining the game. The robots will then subscribe to each others information. This way it can be very precisely regulated which information will be send to which robot. It however makes the module more complex (and thus prone to bugs), since all subscriptions have to be handled by different threads and all should be neatly canceled when a robot leaves the game . The second more efficient way is to send new information directly to all the other robots by broadcast, all the robots will receive the information and it's up to them to decide what to do with it. At this time only information about the state of the ball and the robot's own state are broadcasted to the other robots, but this could easily be extended to the sharing of the information of all the objects in the game.

4.4 Predicting future states

Although it is very difficult to predict the future states of the objects in the game, the world module contains a very simple feature allowing a peek into the future. To determine the future state of any of the moving objects in the game, assuming it has a constant velocity, we simple take the current position and the speed of the object allowing us to predict the position of the object at any given time, using the following formula:

$$x_{t+\Delta t} = v_{x_t} \Delta t + x_t \tag{4.30}$$

$$y_{t+\Delta t} = v_{y_t} \Delta t + y_t \tag{4.31}$$

Since the field is surrounded by walls [1] the objects won't leave the field, therefore collisions with the wall have to be taken into account when predicting the movement of the objects . In our simple model we assume the ball to collide with the wall according the normal physical laws (the angle in which the ball leaves the wall equals the angle in which it hits the wall). When a robot collides with a wall it is assumed to stop. To make our model more accurate we could also take into account the deceleration of the ball. However this would require extensive testing to come up with an accurate model of the deceleration of a ball on a RoboCup field including the effect of collisions with different kinds of objects. To predict the positions of the robots an accurate model of the behavior of the robots is needed, which, for the robots of the other team, takes us into the research area of the opponent modeling. These techniques could improve our predictions a lot, but since we hardly use long term prediction anyway it isn't necessary increase the complexity of this feature.

4.5 The ball

One of the most important aspects of playing a game of soccer is knowing who has possession of the ball. Both individual player and strategic team decisions

[1] As mentioned before, in the near future these walls will be removed, making robot soccer more challenging and increasing the resemblance to human soccer.

are based upon who has the ball. Knowing where all the robots and the ball are, the world module should be able to determine which of the robots is in possession of the ball. There are, so far, four possible states of ball possession.

- *I have the ball.* When the ball is within a certain distance in front of the robot (i.e. the ball is between the ball handlers of the robot), this is detected by the player skills module (using vision). This module on its turn sends a trigger to the world module notifying it that the robot is in possession of the ball. When the ball leaves the ball possession region in front of the robot a trigger is also sent to the world module notifying that the robot has lost possession of the ball.

- *We have the ball.* When one of the robots in the team gets in possession of the ball it sends a broadcast to all the other robots in the team telling them that the team is in possession of the ball. Vice-versa, if a robot loses possession of the ball it also sends a broadcast telling its team members that the team has lost ball possession.

- *They have the ball.* Determining whether an opponent has the ball is much harder since it is almost impossible to detect if the ball is in the opponents ball handling mechanism. We try to detect ball possession of the opponent by checking for all the opposing robots that are being tracked in our world model whether they are close to the ball(less than about half a meter). If this is the case we compare the heading and speed of the ball with those of the opponent if they are roughly the same we assume that the opponent has the ball. This seems to be a reasonable assumption because the opponent may be expected to get the ball if it remains near it for some time. In the rare case where both one of our teammates as well as an opponents seems to have the ball, for instance when they are fighting over the ball and thus both are near it, we assume that our player has the ball, and disregard the opponent, which can of course quickly be corrected should the opponent steal the ball.

- *Nobody has the ball.* In this final state, there either is no robot near the ball or we just don't know where the ball is. In both cases the same behavior is needed, namely search the ball and try to intercept it, so we don't have to distinguish these two situations.

4.6 A robust module

Playing a game in a dynamic real world situation means that the robot can encounter many unexpected situations. As many as possible precautions have been taken to make the world module as robust as possible. There are mainly two kinds of precautions, the first deals with problems in the software, such as crashes of modules or faults in communication, the other is meant to make the world module as robust as possible to faulty incoming data. Some examples of the precautions are:

- *heartbeat system.* Because of the large dependencies between the different modules on a robot, the crashing of one the modules will often result in problems with all of the modules. To detect when any of the modules

is in trouble, all the modules will have to send a heartbeat to the coach module. As soon as any of the modules doesn't give any heartbeat, which probably means the module has crashed, all the other modules are neatly closed down and the software on the particular robot is restarted.

- *crashes of team members.* Should one of the team members fall out without sending a broadcast telling the world modules on the other robots that it has left the game, this would cause the other world modules to crash, due to the subscriptions which are used between the modules, which have to be neatly cancelled. To deal with this the Msm module (the module responsible for starting up and monitoring the other modules) on the robot which world module has crashed will detect the crash as the module doesn't send a heartbeat and will send a broadcast to the other robots telling them this robot has left the game. A neater way of dealing with this kind of problems would be to broadcast information instead of using the subscription mechanism.

- *filtering on time.* Due to system problems or other unpredictable situations it is possible that the world module receives more new observations than it can handle, meaning that the communicating queue fills up. When the world module finally is ready to handle the observations in the queue will be aged, and less useful since they will only lead to an out-of-date world model. To solve this problem, the world module will disregard incoming observations that are too old (this of course is very arbitrary, but at this time, too old means older than 2 seconds). This way the world module will make up its arrears and only use up-to-date observations to update the world model.

- *lost/found trigger.* In the situation that the vision-based self-localization isn't able to determine the robot's position (because and doesn't see enough lines or can't determine which candidate is the best), it is possible to determine whether the estimated position of the robot is correct based upon vision information. For instance when the robot's estimated position indicates that it should see the yellow goal at 8 meters distance but in fact it sees the blue goal at 1 meter distance, it would be reasonable to say that the estimated position is wrong. In that case the vision module will send a lost/found trigger to the world module. The world module will dramaticly increase the uncertainty estimate of its own position information. The world module should now no longer send the information it has about the world to its teammates because it is faulty data, and the team skills module will go into relative mode, meaning that it will make its actions based upon relative information and no longer based upon the absolute model of the world.

- *bump trigger.* The odometry sensor is very sensitive to collisions so it would be desirable to be able to detect when the robot collides. Sadly it is not possible to detect all collisions but when the robots gets stuck to the wall or another robot and spins its wheel without actually moving (which will cause the odometry sensor to think the robot has moved while it hasn't) this will be detected by the player skills module which will send a trigger to the world module telling it that the position estimate

based on odometry is no longer valid. The world module will increase the uncertainty estimate of its own position dramatically. This too will mean that the team skills module will go into relative mode and the world module will stop broadcasting information about the world model.

4.7 Communication with higher level modules

In the previous sections the way a world model is maintained, based upon the incoming sensor data, is described. On the other hand the world module also has to supply the information of the world to the higher level modules which determine the actions of the robot both on individual level (Player Skills module) as well as team strategy level (Team Skills Module). Several interface functions have been designed.

- *Read World* This function will send the latest update of the entire known model of the world to the receiver. Because of the message system only enabling us to send a limited amount of data in each message, this function will send a concise model, consisting only of the states and classifications of the 9 objects in the game (i.e. the 8 robots and the ball).

- *Object info* This function will give all the information we have about one particular movable object in the game. The given information is much more extensive than the information in a *Read World* request. It also allows the request of a prediction of a future state of the object.

- *Blobs in Range* For use in collision avoidance the player skills module will use this function to get all the objects (except for the ball) and unmatched observations which are partly or completely within a certain range (i.e. they are between two relative angels and within the maximum distance).

- *Ball possession information* In making team strategy decisions it is extremely important to know who has possession of the ball. This function will tell the requester whether the robot itself, the team, the opponent or nobody has the ball.

- *Information of Fixed Objects* Finally the other modules can also request information about the fixed objects in the game. Questions like: "Where is the blue goal", "Which goal is ours", "Where is the penalty dot" will be answered.

Chapter 5

Results

In this chapter the performance of the world module is evaluated. The data used in this chapter originates from tests on the UvA robot-lab soccer field and the log-files created at the the RoboCup tournament in Seattle. Three main features of the world module will be discussed here: self-localization, object detection and the sharing of information with the other world modules.

5.1 Clockwork Orange at RoboCup Seattle

As mentioned before, Clockwork Orange attended the RoboCup 2001 event in Seattle. It participated in the middle-size league which consisted of eighteen teams divided over three groups of six teams each. The other teams placed in the group of Clockwork Orange were JayBots(USA), ISocRob (Portugal), Cops Stuttgart (Germany), Trackies (Japan) and Fun2Mas(Italy). The games against the Jaybots, ISocRob and Fun2Mas resulted in a win for Clockwork Orange (respectively 2-0, 3-1 and 5-0). The spectacular game against Cops Stuttgart ended in a draw (2-2) and the game against Trackies ended in a dreadful defeat (0-8). Clockwork Orange finished third in this group meaning that it had to compete with the teams that finished third in the other groups: GMD (Germany) and Artisti Veneti(Italy), for the two remaining quarter final places. A win over Artisti Veneti (3-0) and a loss against GMD (1-4) gave Clockwork Orange their deserved place in the quarter finals. The next opponent, three time RoboCup champion CS Freiburg, turned out to be too strong, the match ended in a 0-4 defeat. CS Freiburg reached the finals in which it defeated Trackies. Being eliminated by the winner of the tournament at the quarter finals stage means that the participation of ClockWork Orange at the 2001 RoboCup can be called successful. Part of the information used in this chapter will come from a representative set of log files of the games against Fun2Mas, Artisti Veneti, GMD and CS Freiburg.

5.2 Self-localization

Knowledge about the robot's own position is based upon the information of two different sensors: i.e. odometry and vision-based self-localization. In this section

the accuracy and the suitability for robot soccer of both sources of information and the current combination method will be discussed.

5.2.1 Odometry

As mentioned before, odometry information is very useful when driving short distances, but it loses its accuracy when driving longer distances. The three possible causes for the increase in the error in the position estimation by the odometer are: collision, slip and bias. When the robot is ran into by another robot this could cause the robot to move without the odometry noticing this movement. The wheels of the robot skidding makes that the odometry senses motion while in fact the robot isn't moving. And finally there's the systematic error which occurs in every odometry sensor. The first two of these causes can't be easily detected, they require special sensors or very complex vision-based methods. The systematic error however can easily be determined using the University of Michigan Benchmark (as described in [5]). This error is caused by imperfections in the design and mechanical implementation of a mobile robot. The two main causes for this error are:

- *Unequal wheel diameter.* The robots use rubber tires to improve traction. It is however almost impossible to manufacture rubber tires that have exactly the same diameter. Furthermore an asymmetric weight distribution of the robot will result in differently compressed tires. This will cause translation faults, meaning that the robot will turn away from a planned route.
- *Uncertainty about the wheelbase.* Because the tires have a contact region instead of a contact point with the ground, it is hard to determine the effective wheelbase. This causes an orientation fault.

In the UMBenchmark Test the robots drive along a 4×4m square path[1]. They do so five times in a clockwise direction and five times in a counterclockwise direction. This driving in both directions will avoid underestimating the error due to the counteracting of the factors mentioned above. Each time the robot has driven along the path once, the real position (relative position to the planned end-point of the path) and the position estimation by the odometry are measured. Subtracting these two will result in the error of the odometry ϵ_x and ϵ_y. The results for the robot Nomado are shown in figures 5.2 and 5.1. Using all five measured errors in one direction we can determine the center of gravity for the corresponding direction.

$$x_{c.g.,CW/CCW} = \frac{1}{n}\sum_{i=1}^{n} \epsilon_{x_{i,CW/CCW}} \tag{5.1}$$

$$y_{c.g.,CW/CCW} = \frac{1}{n}\sum_{i=1}^{n} \epsilon_{y_{i,CW,CCW}} \tag{5.2}$$

The absolute offset of these two centers of gravity from the origin are given by:

$$r_{c.g.,CW/CCW} = \sqrt{(x_{c.g.,CW/CCW})^2 + (y_{c.g.,CW/CCW})^2} \tag{5.3}$$

[1] Due to the restrictions of the size of our field, we performed the test using a 3×3m square

real			odometry			$\epsilon_{X/Y}$		
x	y	ω	x	y	ω	ϵ_x	ϵ_y	ϵ_ω
-120	-170	70	-10	-78	28	-110	-92	42
-95	-180	80	-2	-76	21	-93	-104	59
-108	150	70	-7	-73	29	-101	-77	41
-106	-185	70	-2	-78	23	-104	-107	47
-130	-206	90	-5	-76	27	-125	-130	63

Table 5.1: Clockwise results Nomado(in mm and 0.1°). Shown are the real position relative to the starting/ending-point, the position estimate from the odometry and finally the corresponding error in the odometry sensor.

real			odometry			$\epsilon_{X/Y/\omega}$		
x	y	ω	x	y	ω	ϵ_x	ϵ_y	ϵ_ω
51	-30	-10	-7	65	-29	58	-95	19
67	-30	0	-2	76	-25	69	-106	25
61	-21	-30	-5	73	-36	66	-94	6
69	-20	0	-5	76	-32	74	-96	32
79	-25	0	-7	78	-25	86	-103	25

Table 5.2: Comparable to 5.1 only this time the results come from a Counter-clockwise run.

Finally the maximum of $r_{c.g.,CW}$ and $r_{c.g.,CCW}$ is defined as the measure of dead-reckoning accuracy for systematic errors.

$$E_{max,syst} = max(r_{c.g.,CW}; r_{c.g.,CCW}) \tag{5.4}$$

For our robot Nomado this results in $x_{c.g.,CW} = -106.6$ and $y_{c.g.,CW} = -102$ for the clockwise direction and $x_{c.g.,CCW} = 70.6$ and $y_{c.g.,CCW} = -98.8$. This gives us respectively $r_{c.g.,CW} = 147.5$ and $r_{c.g.,CCW} = 121.4$. Taking the maximum of these two allows us to conclude that the systematic error in the odometry is 147.5 mm. In other words the systematic error is about 1.2% of the traveled distance of 12 m. Originally a value of 5.0% of the traveled distance was taken as the uncertainty estimate of the odometry (this includes a safety margin to cope with small external effects like irregularities of the surface of the field and minor collisions). However these tests indicate that a smaller value of about 3.6% (three times the systematic error: 3σ) could also be sufficient.

Given this data and the results given in 5.1 and 5.2 we can conclude that the error in the x-direction is caused by the fact that the wheels of the robot skid. And the error in the y-direction is caused by one wheel being slightly larger or skidding more than the other wheel.

Estimating the robot's own speed. The estimation of the robot's speed is based upon the odometry sensor as well. Using the number of rotations of the wheels in a certain time interval, the speed can be calculated. We tested the accuracy of this speed estimation by letting the robot drive along a straight line for 7 meters. Figure 5.1 shows the estimated speed of the robot from the odometry. By measuring the time it took the robot to reach the end, the average speed of the robot could be determined. It turned out to be about 0.75 m/s,

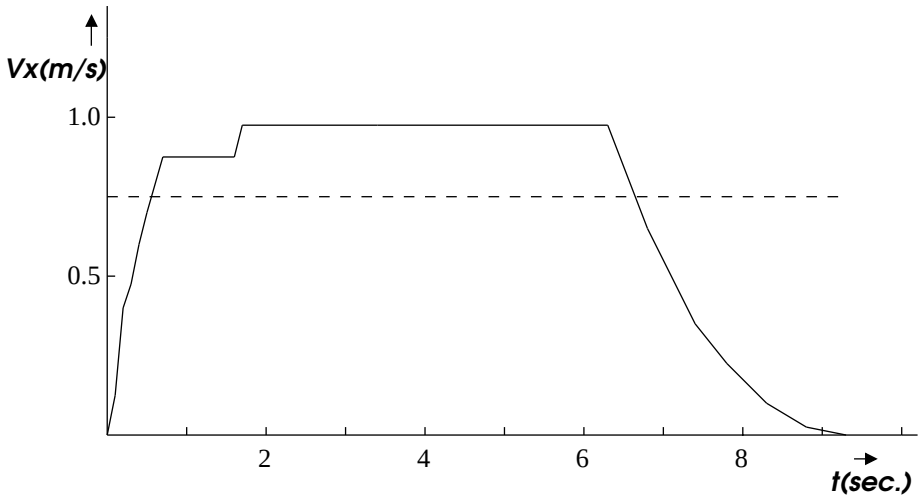


Figure 5.1: The robot's speed estimate when driving over a distance of seven meters. The dotted line indicates the real average speed of the robot.

which corresponds to the average of the speed estimates done by the odometry sensor. Also the robot's top speed was measured on a 2 meter long middle segment of the trajectory. The measured top speed was 0.98 m/s which equals the odometry speed estimate.

5.2.2 Vision-based self-localization

This vision-based self-localization method is used to correct the odometry position from time to time. It is based upon a combination of two different methods. A global method tries to fit the lines on the field perceived to a model it has of the field. Ambiguity caused by the symmetry of the field or the lack of sufficient information will result in multiple position candidates being found (this number could be reduced by using additional sensors such as a compass). A local tracking method tries to find which of these candidates is valid by checking the robot's former positions and using external information such as which goal is seen etc. The local tracking method however does require the robot to move making it almost impossible to do a vision-self-localization when the robot is standing still, because multiple candidates will be found and it is impossible to choose which is the right one. When a candidate is finally found which satisfies all the conditions required for a position to be plausible, it is send to the world module which tries to combines it with the odometry information as described in chapter 4. It would have made testing and generating results much easier if we could test it while the robot was standing still, but since this isn't possible we will let the robot drive a pattern across the field and we will compare the position estimates found by vision with the position estimates of the odometry and the real position of the robot.

The results are presented in figure 5.2 which gives a graphical reproduction of the robot's path and table 5.3 which more precisely allows us to compare the vision self-localization updates with the odometry and the real position.

The black line is a rough approximation (up to 20cm) of the robot's real path. The dotted line shows the position estimates from the odometry. And finally the circles represent the positions which were given by the vision-based self-localization, the line-piece indicating the orientation of the robot. The robot started on position 1 and was ordered to drive from via-point to via-point, the via-points being the corners of the goal-areas. As can be seen, the odometry indicates the robot driving neatly from point to point. However the real position differs somewhat from the position where the robot thought it was. After driving the complete path, the difference between the real position and the odometry estimate was about 40 cm (which is 1.9% of the total traveled distance). As could be expected the difference increased proportionally to the traveled distance, and remains within the uncertainty estimate which predicted an uncertainty region of 106.1 cm(5.0% of the 21.23 m total traveled distance). It also stays within the 76.6 cm error region we would get if we would use 3σ (3.6%) for our uncertainty estimate, which indicates that this could be a good value. Twenty-two position updates were sent by the vision system during the 79.2 seconds of this experiment. This means that on an empty field (i.e. none of the other objects of the game are on the field) on average once every 3.6 seconds a vision-based position update is given. Most position estimates seem to lie within a few centimeters of the robot's real position (better results may be expected on a field with better lighting conditions and a better calibration of the vision system). On average about 5% of the updates can be classified as outliers. A good example of such an outlier is the ninth update in table 5.3 which is almost a meter from its real position. The robot also receives some very inaccurate updates when standing still near position 3 (update VI). Including the outliers the average error of the vision updates is 367.4 mm. When we discard the outliers during this fragment the average error is 212.9 mm. These numbers do not correspond to the uncertainty estimate of about 100 mm provided by the vision module. It also indicates that the method we were forced to use during RoboCup 2001, in which the vision-based self-localization is used to reset the position estimate of the robot, is very incorrect. Using this method we assume the uncertainty of the vision update to be 0 mm. The vision updates completely determines the robot's position estimate while it should, according to its real uncertainty, have a much smaller influence.

The frequency in which the world module receives new positions from the vision module depends upon many factors, such as the position on the field, the number of lines visible etc. During the games played in Seattle the robots received a number of position updates varying between as few as 46 and as much as 190 in a game. On average about 81 position updates were done in a game which on average lasts slightly more than 30 minutes. This means that these position updates were send to the world module about once every 24 seconds (which is about 7 times less frequent than on an empty field). Examples of the spatial distribution of the updates of three of the robots (Caspar, Nomada and Ronald) during two of the RoboCup games are given in figures 5.4 and 5.5. The graphs don't give any information about the accuracy of the position estimates since we don't have any ground truth about the robots' locations. However they do give us some information about the frequency and distribution of the updates.

In figure 5.4 data from the match against Artisti Veneti is shown. The updates are quite neatly distributed over the entire field. Caspar received quite

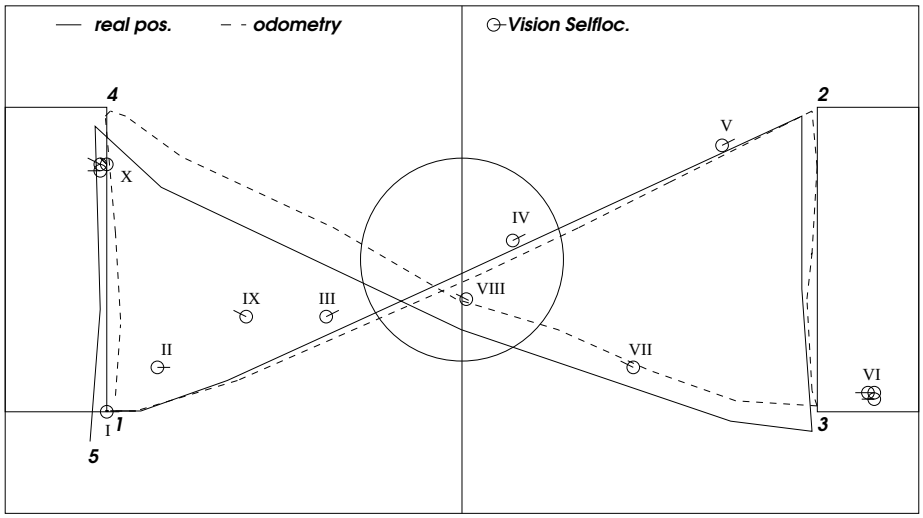


Figure 5.2: Self-localization test. The real position, the odometry position estimate and the the vision-based self-localization updates are shown.

	Vision SL.				odometry				real	
	x	y	ϵ_x	ϵ_y	x	y	ϵ_x	ϵ_y	x	y
I	-3497	-1498	3	2	-3497	-1498	3	2	-3500	-1500
II	-3001	-1184	-1	216	-3000	-1480	0	-30	-3000	-1450
III	-1331	-614	-231	136	-1109	-777	-9	-27	-1100	-750
IV	491	265	-209	165	706	109	-6	9	700	100
V	2468	1223	118	173	2451	970	111	80	2350	1050
VI	4132	-1395	1132	455	3477	-1435	277	415	3200	-1750
VII	1786	-1173	36	327	1846	-1087	96	413	1750	-1500
VIII	-68	-401	-18	299	73	-294	123	406	-50	-700
IX	-2272	-629	-372	879	-1783	584	217	334	-1900	250
X	-3523	854	277	-346	-3439	1452	361	252	-3750	1200

Figure 5.3: The table shows a representative set of world-relative positions of the updates and the corresponding odometry estimates (both with their errors compared to the real position) and rough approximations of the robot's real position.

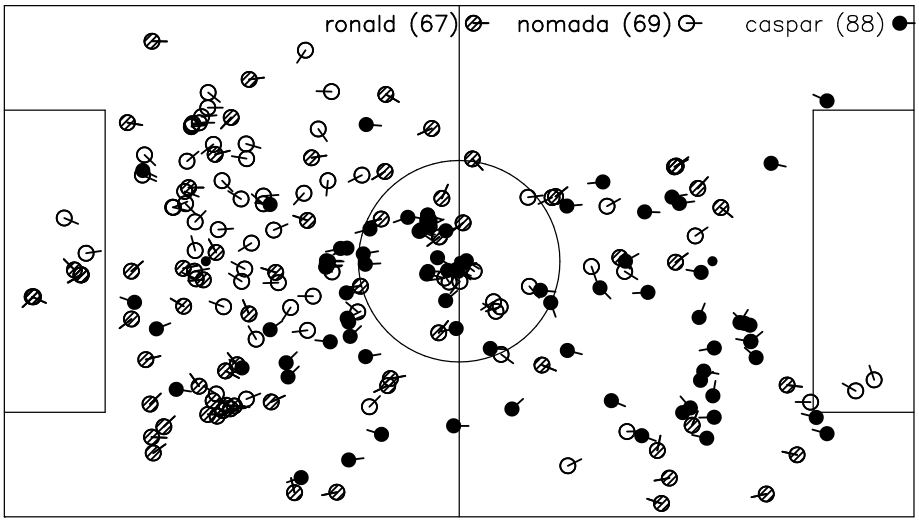


Figure 5.4: Positions (with orientation) of the updates by the vision-based self-localization during the RoboCup 2001 match against Artisti Veneti.

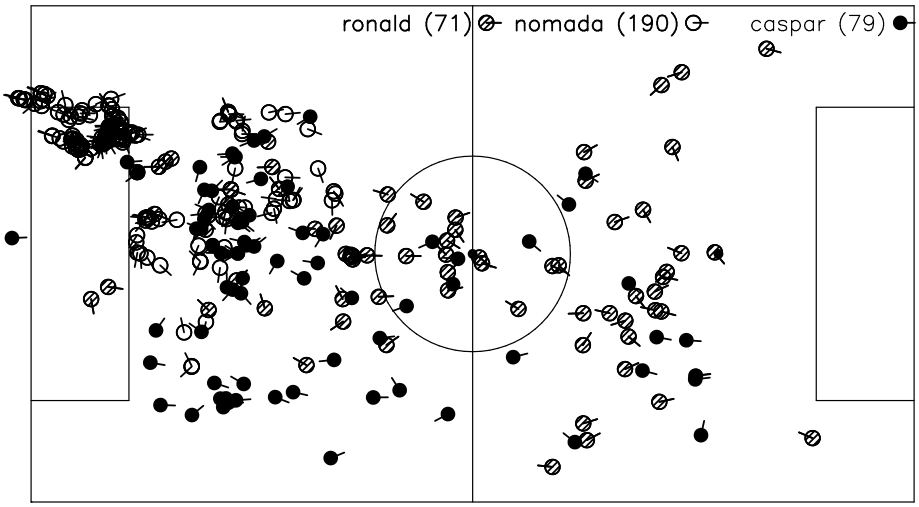


Figure 5.5: Same type of figure as figure 5.4, only here data is presented from the RoboCup 2001 match against GMD.

a large number of updates on the opponents half because it was in attack-mode most of the time. The other two robots received most updates on their own half of the field because they were in defend-mode. Relatively few updates were done near the outer sides of the field. This is caused by two factors: the robots stayed away from the boarding for most of the time and when a robot was near the boarding it could only perceive a limited part of the field and thus too little information was found to do a good self-localization.

Figure 5.5 shows data from the match against GMD. As can be seen Nomada received an extremely large amount of updates. Most of which were near the same location on the field. It seems that the robot spend most of the time on that position, where it could also find the desired information to do a self-localization without this information being occluded by other robots. It shows us that a much higher frequency of updates can be achieved in a more optimal situation. Furthermore it is clear that in this figure most updates were done on the teams own half, which makes sense because the game was played against the strong GMD team and the Clockwork Orange team was forced to defend most of the time.

5.2.3 Combined self-localization

During the matches of RoboCup 2001, the vision-based self-localization information was used to reset the odometry information. This means that whenever the vision system finds a plausible position estimate this will become the new position of the robot. Figures 5.6 and 5.7 give a general idea of how the position and uncertainty estimates change during a real game of soccer. Presented here is data from a 190 second long segment of the RoboCup 2001 match against Artisti Veneti. The black line and the dotted line in figure 5.6 respectively represent the robot's estimated x- and y-coordinates. The vertical lines indicate that a vision-self-localization was done at that moment.The fragment begins just after a game restart. The robot remains near the center of the field for some time before it starts moving around. While standing still, a number of vision updates are done, but they don't affect the position estimate since it already seems to be correct. After that the robot drives around for about 40 seconds before the next vision update is done. As could be expected after driving such a distance, the vision update differed quite a lot from the odometry position, resulting in quite a large position shift.

The uncertainty estimates (x, y and orientation as shown in 5.7) increase proportional to the traveled distance until an update is done. Throughout the RoboCup 2001 games the uncertainty estimate never exceeded 600 mm in any direction. This means that no robot ever drove more than 12 meters in any of the 2 directions (600 mm is 5.0% of 12 m) without getting a new update. On average the traveled distance between two updates was about 2.5 meters.

A special situation occurs when the game is about 540 seconds old. Apparently the vision system detected that the robot's position estimate is false without knowing what the real position should be, for instance because a fixed object (e.g. a goal) is perceived at a position where it shouldn't be perceived given the robot's supposed position. The world module is notified that its data is erroneous and it sets the uncertainty estimate to a very high value. Shortly after this moment a new position update is done making that the position estimate used in the world model doesn't conflict with the objects perceived by

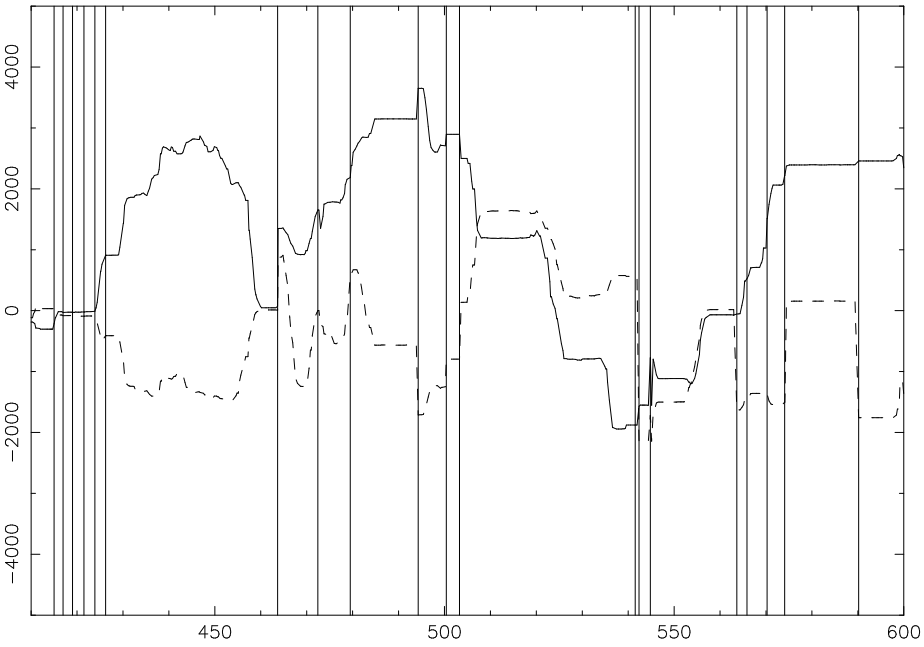


Figure 5.6: Example of the position estimates (world-relative x and y coordinates in mm) of the robot and the vision self-localization updates against the time (horizontal axis in seconds). The continuous line is the x-coordinate and the dotted line is the y-coordinate. The dramatic position shifts caused by the updates can be clearly seen.

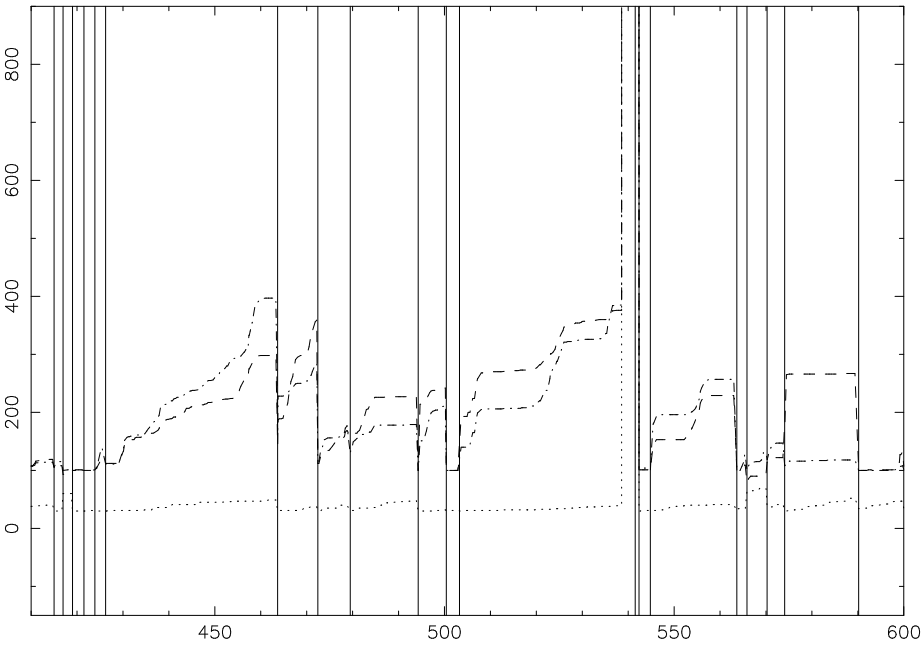


Figure 5.7: The estimated uncertainty in the robot's position in mm(i.e. 5% of the traveled distance) with the self-localization updates during the same game segment as presented in 5.6.

vision anymore. The uncertainty estimates are restored to their normal values.

Although we don't have any ground truth about the robot's real position, the data does provide us some information about the error in the odometry estimates. We know that for the odometry the uncertainty estimate increases proportional to the traveled distance, so we can deduce the traveled distances (between the self-localization updates) from the uncertainty estimates as in 5.7. Comparing the position shifts as shown in 5.6 with the traveled distances gives us an average error of about about 14% (of the traveled distance) in the x-direction and 13% in the y-direction. This is comparable to the expected error we get when we combine the expected 3σ (3.6% of the traveled distance) error in the odometry and the average error of the vision self-localization (212.9mm), which is about 18% of the traveled distance in both directions. This shows us that the assumption of the error of the odometry being 3σ, is correct.

Throughout the games of RoboCup 2001, the robots thought (using the lost/found trigger as described in 4.6) their position to be false for 51% of the total time. This is quite a large amount of time, but occlusions of large parts of the field by the other robots make that it is hard for the vision system to come up with self-localization updates (as the frequency of one update every 24 seconds already indicated). This together with the fact that odometry quickly loses its accuracy, especially in an environment as dynamic as RoboCup, where the robot will collide quite often, explains why the robots thinks their position to be false for such a large amount of time.

5.3 Object detection

Of course knowing your own position isn't enough to be able to play a good game of soccer. Knowing where the ball is and where the other players are is just as important. This section will present the performance of the world module on the aspects of determining the position of a perceived object, estimating the objects speed and telling objects apart when multiple similar objects are perceived at the same time.

5.3.1 Object position estimation

To test the accuracy of the estimation of the position of the other objects by the world module we placed the ball at 12 predefined positions (as shown in figure 5.8) and compared the position estimates with the real positions. The results are shown in table 5.9. The accuracy of the position estimates decreases when the object is placed further away from the center of the robot's field of vision (this could be due to lens abberation as described in [11]). It also decreases when the object is placed further away from the robot. As can be seen in table 5.9, the errors in the position estimates are all neatly within the expected uncertainty region. The uncertainty region increases when the object moves further away from the robot or is placed near the edges of the robot's field of vision. It is also influenced by the motion of the object or the motion of the robot (the position estimate being relative to the robot makes that these to have the same effect). When the object is further away from the robot, a small amount of noise will result in quite a large estimated speed of the object, explaining why the uncertainty estimates in the x-direction increase more on the larger distances

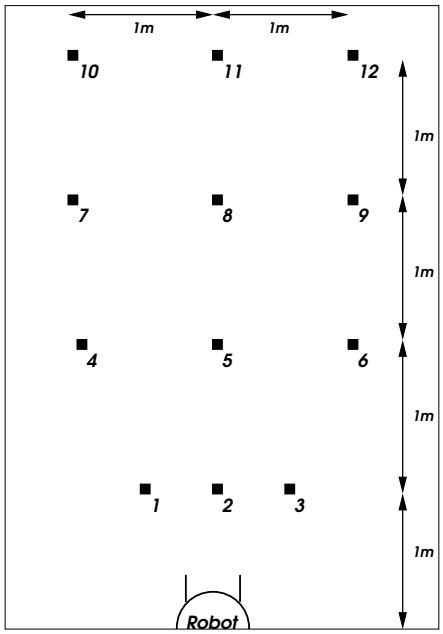


Figure 5.8: The 12 predefined positions of the ball.

pos. #	ϵ_x	ϵ_y	Cx	Cy
1	68	100	206	160
2	118	-31	211	103
3	69	-166	207	171
4	147	-165	839	517
5	140	-63	710	240
6	268	200	871	614
7	96	-89	1100	510
8	215	-104	1156	303
9	436	-331	1199	629
10	280	-199	1440	488
11	315	-175	1297	289
12	649	-384	1498	634

Figure 5.9: The errors in the position estimates of the observed object and the corresponding uncertainty estimates.

then would be expected based upon the actual errors. It might be expected that additional experiments with robots or other black objects of which the diameter is known, will give results which are comparable to this test with the ball.

Since there are multiple robots on the field and they all are black it makes sense to take a look at how well the robot can distinguish them. Three robots were placed on the 12 predefined positions (as in 5.8) and the world module was consulted on where it thought the robots to be. Figure 5.10 shows the results in various situations. The circles indicate which of the robots were detected, the crossed out circles indicate that the robot on that position wasn't detected as being a separate robot. The first situation shows us that the robot is able to distinguish two other robots at a distance of about one meter, when the distance between them is as small as 5 cm. However a robot being partly occluded by another robot will in most cases result in the furthest robot being interpreted as being a part of the closer robot and thus not being detected as a separate object. When two objects are at medium distance another robot at a large distance is perceived between, then the furthest robot will be detected as a separate object as shown in the last situation. This problem will be partly solved by sharing the local world models with the other robots of the team.

5.3.2 Object speed estimation.

In order to evaluate the speed estimation of the other objects in the field, a method had to be found to determine the real speed of the object (in this case the ball). The test setup shown in figure 5.11, where the ball rolls down a slope and passes the points A and B, was used to make sure the ball had the same

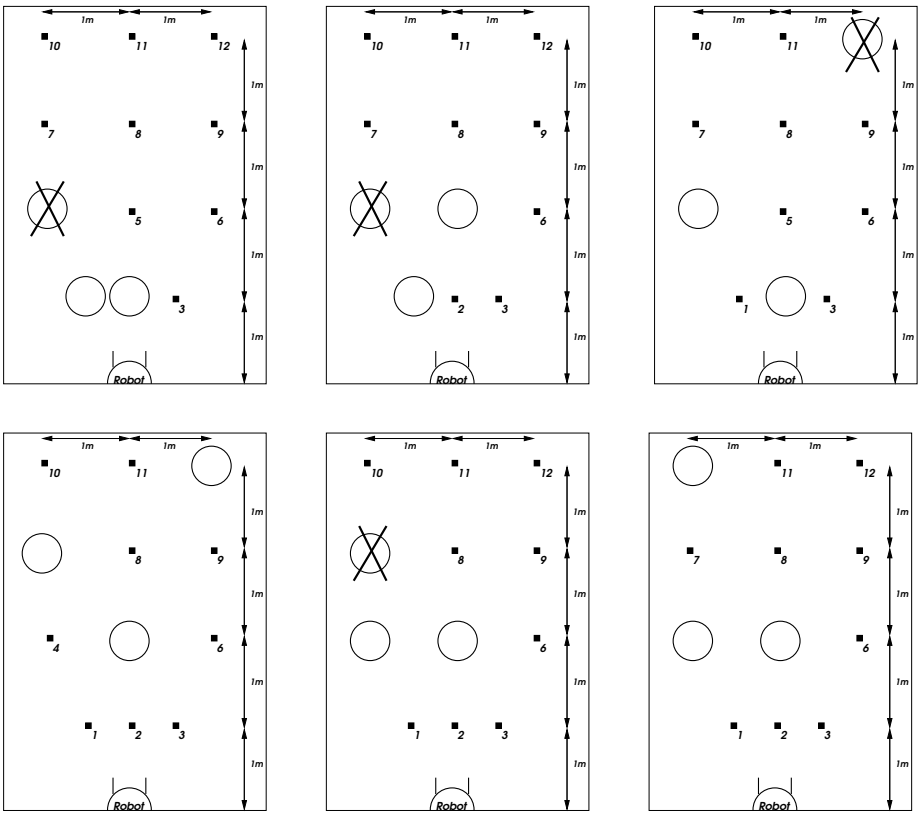


Figure 5.10: Distinguishing other robots in various situations. The circles represent the observed robots being detected as a separate robot. The crossed out circles indicate that this robot wasn't detected as a separate robot.

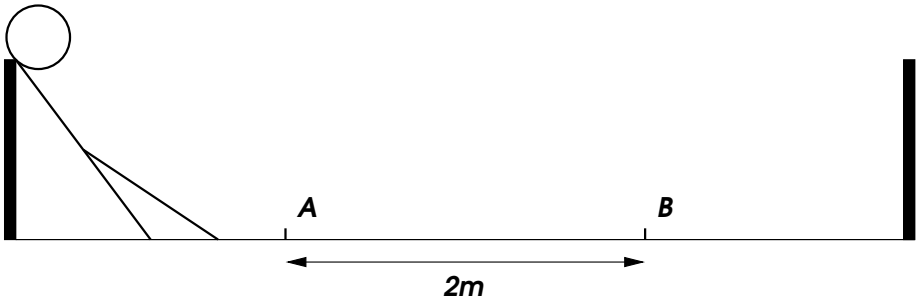


Figure 5.11: Test-setting for testing the speed estimation.

speed at all the tests. The time it took the ball to roll from point A to point B (which are 2 meters apart) was measured in order to calculate the speed. The average real speed of the ball was about 2.3 m/s. Two different type of tests were done with the robot.

In the first test the ball passed from left to right in front of the robot. On average the robot estimated the speed of the ball to be -2.6 m/s in the robot-relative y-direction (meaning that the balled was estimated to roll from left to right relative to the robot) and on average about 0.2 m/s in the robot-relative x-direction, which indicates that the robot is capable of detecting even small speeds, in this case caused by minor deviations from its planned straight path.

The second test was used to determine the accuracy of the speed estimation in the robot-relative x-direction. The robot was placed next to the slope, so the ball would suddenly appear in the robot's view and roll away from the robot. The average speed estimate done by the world module was 2.3 m/s in the robot-relative x-direction (meaning that the ball rolled away from the robot) and about 0.1 m/s in the y-direction. At the end of the trajectory the ball bounced against a wall and rolled back. This could also be detected in the robot's speed estimate. First the speed dropped to zero and after that the ball got a negative speed estimate.

5.4 Sharing world models

Having an absolute model of the world allows us to share it with the other robots in our team to increase the accuracy and completeness of the world models on all the robots. Figure 5.12, shows the world model of the robot Caspar during a 30 second period in the RoboCup 2001 match against GMD. The positions of the shared objects are shown (i.e. the three field players in the team and the ball). As mentioned in the previous chapter, information about the opponents isn't being shared so there's no need to show that data in these figures. We use this figure to give a general idea of the situation on the field during the fragment we want to use for comparing the world models.

The information on which the world model is based comes from two kinds of sources. The first kind is the local source (vision information for the other objects and odometry for the robot's own position) and the second one is information shared by the other robots (each of the robots shares the information it has about the position of the ball and its own position).

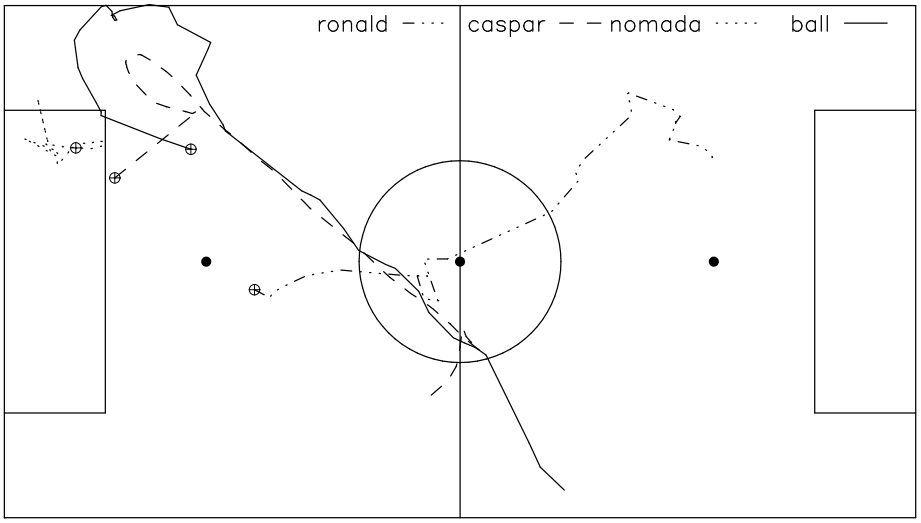


Figure 5.12: A 30 seconds long fragment from the RoboCup 2001 match against GMD, the way Caspar saw it.

The fragment shows Nomada standing in its own goal area, apparently being stuck or just not knowing what to do. This situation is also shown in 5.5 where the great number of updates is caused by Nomada being in this position for quite some time. Furthermore we see Caspar drive to the ball, having it in possession for some time and then losing it. Finally Ronald starts on its own half of the field, drives around a bit and goes to the opponents half.

Now lets compare the world models of the different robots. To begin with the position of the ball according to the different robots as shown in figure 5.13. Since we have no ground truth it is impossible to say how correct the estimates are, but we can conclude that the the world models correspond quite good to each other. The difference in the ball position estimates by the different robots stays within a distance of half a meter. In the beginning of the fragment the position estimates of the robots are based upon both their own observations and the information shared by the other robots, so the estimates will differ slightly. At the end of the fragment only one robot sees the ball, this will result in the estimates of the ball's position done by all the robots being equal, since they all their estimates are based upon the observations from that one robot. The fact that the world models correspond nicely in this situation has everything to do with the robots quite accurately knowing their position. If any of the robots would completely lose its own position, this would result in a estimated ball position that is totally different from the other robots' estimates since this robot would rely on its own perception of the ball and ignore (not being able to match it to its own world model) the information sent by the other robots.

Figure 5.14 (and more clearly in 5.15 and 5.16) shows the position of Ronald according to the different robots. The position estimate of Ronald itself is completely based on its own self-localization. The estimates of Ronald's position done by the other robots is based upon a combination of the information Ronald sends them and their own observations of black objects which they match to Ronald. Figure 5.17 shows the differences between the estimates of the different

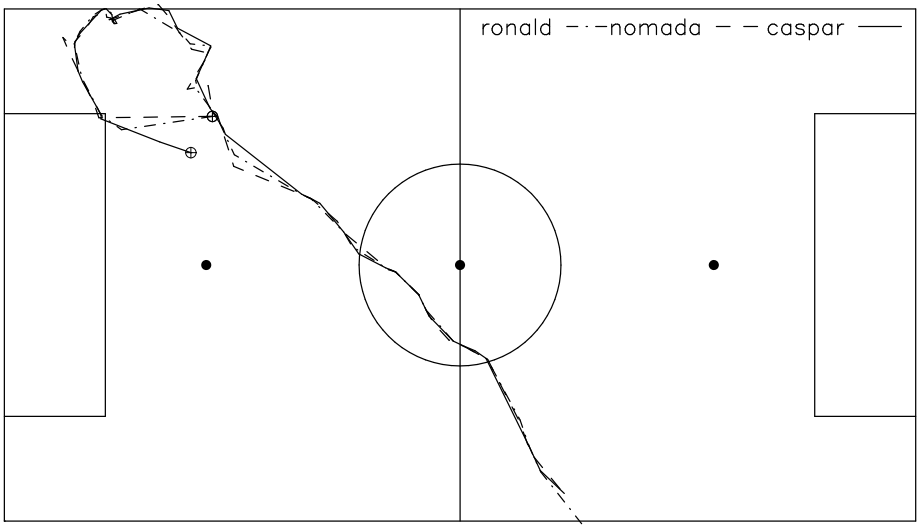


Figure 5.13: The position of the ball in the same fragment as in 5.12 according to Ronald, Caspar and Nomada.

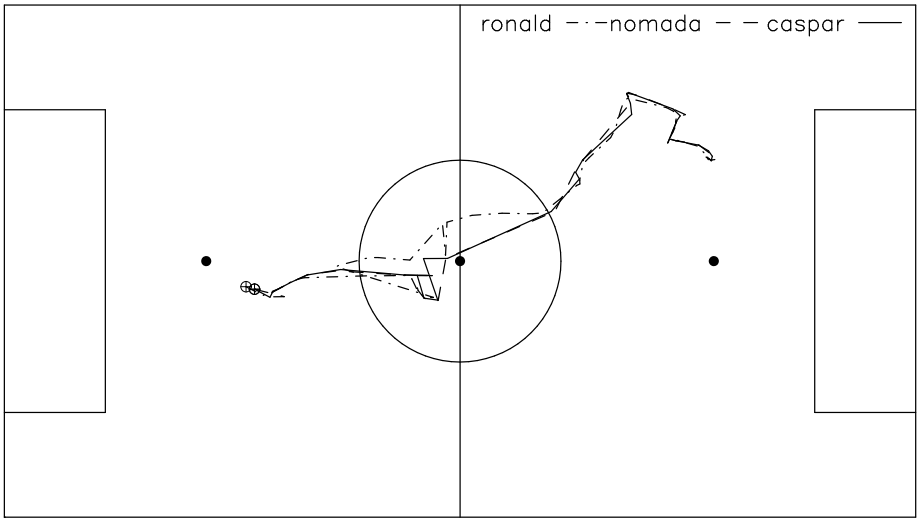


Figure 5.14: The position of Ronald in the same fragment as in 5.12 according to Ronald, Caspar and Nomada.

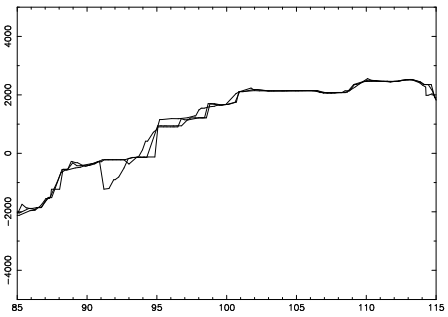


Figure 5.15: The estimates of the x-coordinate of Ronald done by the three robots.

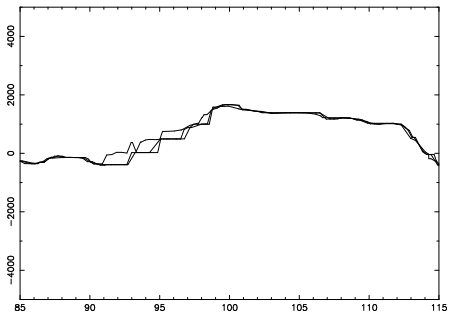


Figure 5.16: The estimates of the y-coordinate of Ronald done by the three robots.

robots and the average of these estimates.

These results show that as long as all robots know their own positions the world models correspond very well (they differ less than one meter for all the shared objects, which corresponds to the uncertainty estimates). When a robot doesn't have a correct estimate of its own position anymore it will stop sending information to its teammates so it won't influence their world models anymore, however it will still receive their information and it will also still maintain its own local world model, which of course will differ from the world models of the other robots. Inconsistency between the robot's own and the other robots' estimates of the position of the ball, could be used as an indication that the estimate either one of the robots has of its own position is incorrect.

5.5 Discussion

Evaluating these results we can say that the world module gives quite an accurate and complete representation of the real world. Sadly the self-localization remains a big problem. The odometry gives us a good estimation for some time but loses its accuracy after longer periods of driving. The originally taken uncertainty measure of 5% of the traveled distance proved to be to loose, a tighter measure of about 3% could be deduced from our experiment. The vision updates do provide some extra information from time to time. However these updates are done too infrequently in a real game to continuously have a good estimate of the robot's position. The average error of 213 mm (not even taking into account the outliers) in an optimal situation (no other objects on the field) indicates that the resetting method as used during RoboCup 2001 doesn't perform good enough. A correct method for combining the odometry and the vision updates will have to be found. An option would be to let the vision module send all its potential position candidates given by its global method to the world module. The world module can then use some form of multiple-hypothesis-tracking (e.g. a particle filter) to estimate the robots actual position. Furthermore it is very necessary that the vision module provides an accurate uncertainty estimate with its updates. It is impossible to correctly combine an update with the old position estimate, if we don't know how accurate the updates is. Finally additional sensors, like a compass, would make the

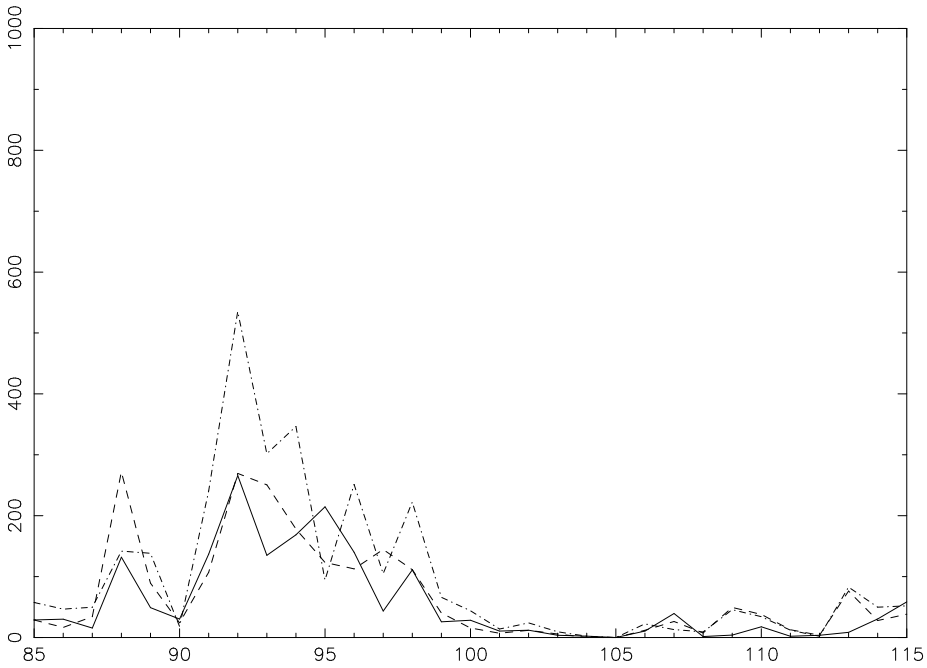


Figure 5.17: The deviations (in mm) from the average of the estimates of the position of Ronald done by the three robots plotted against the time (in seconds).

self-localization task much easier. Knowing the exact orientation of the robot at all time, dramaticly decreases the number of potential position candidates.

The tracking the other objects however went very well. The robots were very capable of distinguishing the other objects and estimating their position and speed. Also the uncertainty estimates correspond to the real error in the estimated positions. Noise in the observations of objects perceived at greater distance will, although filtered out largely by the linear least squares approach, result in the incorrect assumption that the object is moving which will increase the uncertainty estimate too dramaticly. Also given that the robots all know where they are, their world models correspond nicely to each other (i.e. the differences are proportional to the uncertainty estimates).

Chapter 6

Improving Self-localization

The previous chapter shows us that the the self-localization is the major problem in constructing an world-relative model of the world. In this chapter alterations to the currently used algorithms are discussed that could improve the self-localization. No new algorithms or sensing techniques will be presented in this chapter.

6.1 Combination methods

Given the two current sources of information for self-localization, what is the best way to combine them? As mentioned in the previous chapter we planned to use a weighted average of the two. However the vision-based self-localization not being able to cope with this and also not being able to provide us with accurate uncertainty estimates meant that we weren't able to use this technique during the real games. In this section this combination method will be compared to the used resetting method and the odometry-only self-localization. This evaluation of the different methods will be based upon the results of the test as described in section 5.2.2 and shown in figure 5.2, since this is the only test in which an approximation of the ground truth about the robot's position was present. The test data being based on an approximation means that the results presented here are also based upon an approximation.

- **Odometry only.** When the robot's self-localization would be based on the robot's odometry sensor only, one would expect the error in the position estimate to increase proportional to the traveled distance. Figure 6.1 shows the error in the x- and y-direction over the traveled distance (21.255 m).
- **Resetting odometry on every vision update.** In the games during RoboCup 2001 we used the vision updates to reset the odometry, completely ignoring prior odometry information. Using this approach in our test would result in the errors in the position as shown in 6.2.
- **Weighted Average.** Since neither one of the self-localization information sources is perfect, a combination of the two should be used to increase the accuracy. Given that both have a correct uncertainty estimate, this

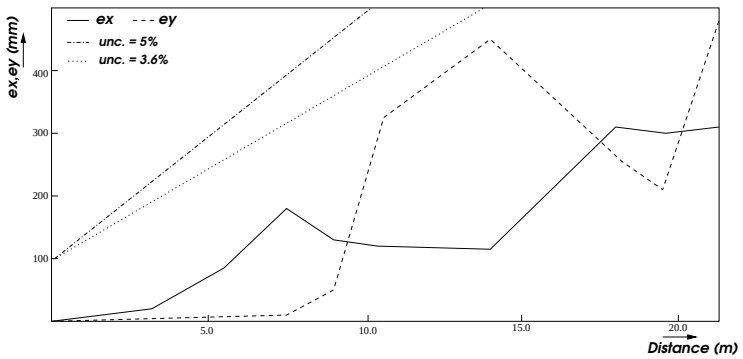


Figure 6.1: The error in the self-localization when based on odometry solely. The original uncertainty estimate of 5% of the traveled distance and the proposed 3.5% uncertainty estimate are also shown.

uncertainty estimate could be used as the weights for a weighted average of the two estimates (as described in 4.2.2). Since the vision updates do not come with a good uncertainty estimate, we will use the outcome of the test as described in 5.2.2, and assume (disregarding the potential outliers) the uncertainty to be about 213 mm, which is the average error of these updates. The errors in the position using this combination method are shown in 6.3.

- **Using a threshold.** Given that no collisions occur we know that the odometry sensor is very accurate for short distances. If we would have reliable information about the accuracy of the odometry data, which we would have using our "lost-found trigger" and a collision detection mechanism, we could rely on our odometry solely and only combine it with the vision updates if the uncertainty estimate exceeds a given threshold value. This will make the system less prone to the outliers in the vision updates. Given a threshold of for instance 1.5 times the average error of the vision updates ($1.5 * 213mm = 320mm$) this would, in the current example, result in the errors in the position as shown in 6.4.

The error of the odometry-only self-localization increases when the traveled distance increases. The resetting method doesn't perform much better. Especially in the first segment of the trajectory it performs significantly worse than the odometry. The outliers of the vision updates can very clearly be seen in this figure. The weighted average method works much better in the first segment because the (at that moment) very accurate odometry information is also taken into account. Also the outliers have a slightly less dramatic effect on the robot's position estimate. Using the threshold method increases the accuracy after a game (re)start but throughout the rest of the game is similar to the weighted average approach. It is clear that, of the methods discussed here, the threshold method has the best performance. It however remains very sensitive to collisions and requires the uncertainty estimates of both odometry and vision-based updates to be very accurate.

It would be very desirable to have a method to detect outliers in the vision self-localization system. This way, when an update is received that could be

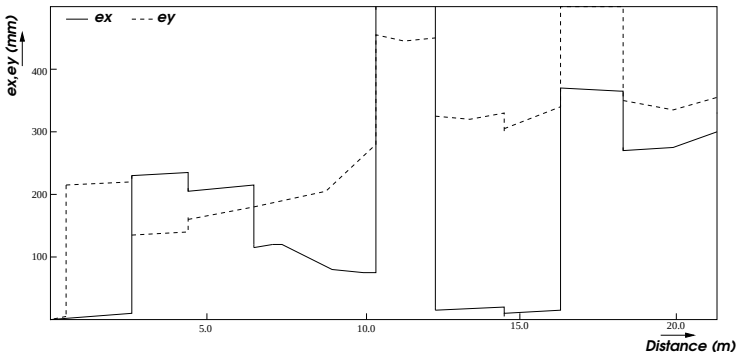


Figure 6.2: The error in the self-localization when resetting the odometry when a vision update is done.

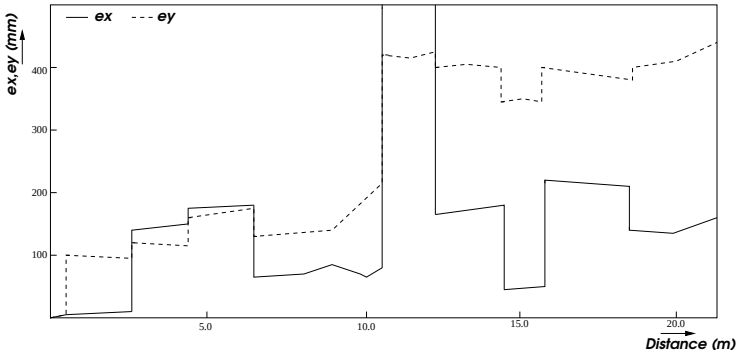


Figure 6.3: The error in the self-localization when using a weighted average of the odometry and the vision updates.

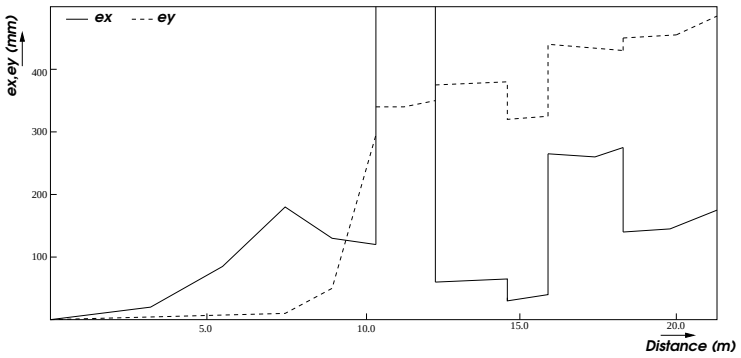


Figure 6.4: The error in the self-localization when only combining the odometry and vision updates if the odometry uncertainty estimates exceeds a threshold of 1.5 times the average error of the vision updates.

classified as an outlier it could be rejected or be given a high uncertainty value, minimalizing its influence on the robot's position estimate. Since we have proven the 3σ uncertainty estimate to be quite accurate, a possible indication that an update is an outlier could be that the update isn't within the uncertainty region of the robot's position. This of course only is an indication that it could possibly be an outlier, since the robot's position and corresponding uncertainty region could also be false because of a collision and we thus have no certainty that the update is incorrect.

6.2 Being observed by other robots

Up until now we did not study one final source of information the robots (in their current configuration) have about their own position: the position the other robots estimate them to be. As the other robots do an observation of our robot they could send the position of this observation to our robot and we can use this to increase the accuracy of our own self-localization. Given that there are four robots in a team on a relatively small field, it will be likely for every robot to be observed by at least one other robot for most of the time. This source of information could therefore prove to be very useful.

Additional self-localization data. The information sent by the other robots could be used as extra information about the robot's position. This way, the other robots could be seen as an additional sensor of the observed robot, providing information about an object (i.e. the observed robot itself).

The data sent by the other robots should come in the form of raw estimates done by their vision sensor converted to world-relative coordinates (as described in 4.6). We only want to use the raw vision data because the position estimates maintained in the world models are also based upon information shared by the observed robot itself (its self-localization information). This would mean that the observed robot would receive an estimate of its own position by the observing robot which is largely based upon its own estimate. Since this is not desirable, the vision data will be shared directly after it has been matched to the robot. The uncertainty of this information will consist of the uncertainty in the vision observation and the uncertainty the observing robot had about its own position. Data originating from an observing robot that knows its own position to be very inaccurate will therefor have a higher uncertainty estimate.

The robot which receives the shared information could process this in different ways. It could be handled the same way as its own odometry information. The data is inserted in the measurement memory at the right place and will be then be used in the linear least squares algorithm to get an as correct as possible estimate of the robot's own position. Another possibility would be to take a weighted average between the new observation and the robot's own position estimate. This would be comparable to the way vision-based self-localization updates are handled.

To answer the question whether this new source of information will really improve the robot's self-localization we'll first need to take a look at the uncertainty of this information. Too much uncertainty about the data will, given the used combination methods, mean that it hardly has any influence on the robot's self-localization. Examining the log-files generated at the RoboCup 2001, it can

be seen that as long as the robots think their own position to be correct (49% of the time) the uncertainty estimates of the observations are reasonably low (on average about 480 mm). Knowing that the average uncertainty a robot has about its own position estimate is about 300 mm, it can be expected that information sent by other robots will have quite a large influence on the position estimate of the observed robot. Using a weighted average, the new estimate of the observed robot's position will for about 38% be determined by the external information and for about 62% by the local self-localization information. So, given that the external information is sent frequently, it will certainly have a noticeable influence on the robots position estimate.

Based on the log-files it is impossible to give any exact results about the increased accuracy of the self-localization, since we don't have any ground truth knowledge about the robots' positions. The differences between the estimates that the different robots have about the position of any one of them (as shown for instance in 5.17) will decrease, and it is plausible that this will mean that the accuracy of the self-localization increases.

Outlier detection. Apart from adjusting the robot's position estimate, observations by teammates could also be used to detect outliers generated by the vision self-localization system. If the vision system sends an estimate which is located outside the uncertainty region of the odometry, information of other robots could be used to detect whether the update is an outlier or whether the odometry data is false. If one or more of the other robots on the team estimate the observed robot to be at the same point at which the observed robot's odometry expects it to be, the update probably is an outlier. On the other hand, if the other robots estimate the observed robot to be near the position estimated by the vision self-localization, this will probably mean that something went wrong with the observed robot's odometry. A nice example to show the benefits of this technique can be seen in the following example.

Presented is the situation we encountered before in section 5.4, a 30 second long fragment from the RoboCup 2001 game against GMD. Figure 6.5 shows the x-coordinate of robot Ronald as seen by the three robots. Figure 6.6 shows the y-coordinate. When the game is about 91 seconds old a sudden shift can be seen in one of the lines representing the x-coordinate. This line is the position according to Ronald itself and the sudden shift is caused by a vision self-localization update. This shift of about one meter is much larger than would be expected from the uncertainty estimate of the odometry which at the time was about 30 cm. The other robots also don't seem to agree with the sudden change in the position of Ronald. They all estimate Ronald to be at the position where Ronald's odometry estimates him to be as well. It therefor is very likely that the update should have been classified as an outlier (and its uncertainty estimate should thus be increased dramaticly). Using the information of the other robots on the team, it would have been classified as an outlier and would not have this big an influence on Ronald's position estimate.

It is hard to tell exactly how well either of the techniques mentioned above would perform in a real game since the presence of many moving robots on the field and the fact that the observer itself will be probably be moving as well will make that correctly matching an observation to the corresponding object

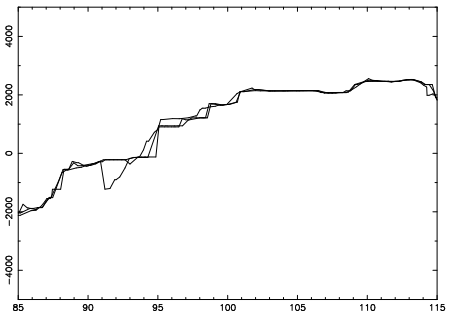


Figure 6.5: The estimates of the x-coordinate of Ronald done by the three robots.

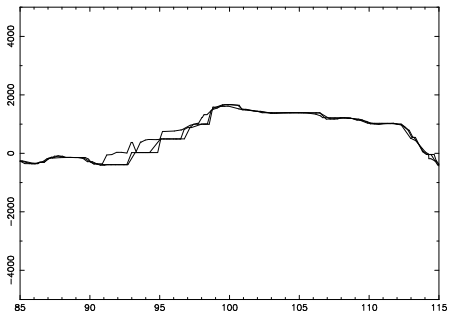


Figure 6.6: The estimates of the y-coordinate of Ronald done by the three robots.

could be difficult. Uncertainty about the observers position and long distance observations will cause a very high uncertainty, which on its turn will mean that this shared information will hardly have any influence on the observed robot's self-localization.

As goes for all techniques where data from various sensors is combined, it is crucial to have correct uncertainty estimates. It doesn't matter when the actual position estimate is very inaccurate as long as this inaccuracy is known, so that its influence on the combined estimate isn't larger than it should be. Given that this condition is fulfilled and that the matching of observations to the corresponding objects works correctly, even in a situation in which there is a large number of moving robots on the field, the presented methods could certainly contribute to a better self-localization. It would be a good idea to implement this method in the near future to see the effect it has in a real game of robot soccer.

Chapter 7

Conclusion

7.1 Discussion

In this thesis Clockwork Orange, the Dutch Robot Soccer Team, has been presented, with special attention to its world model. The methods that are used to construct a model of the real world have been discussed and the performance of the world model has been evaluated. This thesis made it clear that it is far from simple to construct an accurate model of a highly dynamic domain such as robot soccer with limited sensor capabilities.

As one could expect, the performance of the world model completely depends upon the ability of the low-level sensor modules to provide information about the world and supply correct information about their accuracy. When given this data the world module is very capable of constructing an as complete and accurate as possible model of the world. The tracking of objects works quite good, however a less greedy approach to the matching of observation to the object in the model could enable the world module to get better at dealing with ambiguity. The sharing of the local information with the other robots works very good even though at this time it is restricted to the ball and the robot's own position only, but this could very easily be extended with information about the other objects as well. The self-localization remains the major problem for our team. Given our current configuration of sensors it is very hard to come up with a correct estimate of the robot's own position.

The odometry sensor is only accurate on short distances. The results presented in chapter 5 show that 3.6% of the traveled distance is a more accurate uncertainty estimate for the odometry sensor than the originally used 5%. With an average error of about 20 cm (disregarding the outliers), the vision-based self-localization turned out not be very accurate. This would not be a very big problem if only a good uncertainty estimate was provided with the updates. Since this is not the case (and since we have to use the vision updates to reset the odometry information), combining the vision updates and the odometry data after driving short distances will often decrease the accuracy of the robot's position estimate instead of increasing it. However after driving longer distances the vision updates most of the time do increase the accuracy of the position estimate, because in this case the odometry would have become too inaccurate.

There are a number of possible approaches that could improve the self-

localization considerably. The alternative combination methods, as described in chapter 7, already slightly improve the performance. Especially incorporating information sent by the other robots in the team could possibly benefit the self-localization. Other completely new algorithms could also be used to improve the performance. If the vision system would send the position candidates generated by the global method to the world module, a multiple-hypothesis tracking algorithm (e.g. a particle filter) could be used to track them and eventually determine which is the best estimate. Additional sensors could also help enormously. A compass for instance could make sure that the robot at all times accurately knows its orientation. This would reduce the number of potential position candidates.

Of course many possible more complex models of the position, uncertainty and motion could be used to increase the accuracy even further. One has to make a tradeoff between making things as accurate as possible and keeping things simple since we want a efficient system that works and that is relatively transparent for its users. The world model with its current design has all the features to deal with the real world problems without using very complex mathematical models, which would complicate things considerably without any real gains in our performance as a team of soccer playing robots.

One should also take in mind that our RoboCup project is a real world project, in which we have to work with real robots which can break down and present the users with all kinds of other difficulties. Working on a team that wishes to participate at certain events also means that we had to cope with deadlines. Often the weeks before a tournament or an important demonstration were spent on getting the system to work, often done by extensive bug-fixing in all kinds of modules and sometimes by fixing them with quick hacks. Finally the fact that we worked with quite a large team, divided over two universities meant that a lot of time was spent in meetings and waiting for the other people on the team to perform their tasks required to be able to proceed yourself.

Looking at the way the team and in particular the world model performed during the tournaments at which the team participated we can say that it did quite good. A consistent and relatively accurate world model was present on all of the robots for most of the time. Even given the problems we have with the self-localization (due to having to cope with relatively cheap and primitive sensors) we were still capable of playing a reasonable game of soccer.

7.2 Future research

Although the world model of Clockwork Orange did perform quite good, there is always room for improvements. In this section I will make some suggestions for future work. Some of these suggestions will have a greater impact than others, but all will presumable have a positive influence on the performance of the world module.

Maintaining a relative world model. Since our self-localization methods can't always give us a good estimation of our position, it is sometimes impossible to construct a correct absolute world model. The vision module compares the estimated position of the robot with the features extracted from the image from the camera. If the image indicates that the position is wrong, the vision

module will send a trigger to the team skills module to go into relative mode. When the team skills module is in relative mode it makes its decisions based upon information of vision which is relative to the robot and no longer based upon the absolute world model maintained in the world module. This "vision-only-based" relative model of the world this way lacks many of the advantages of the world module such as noise filtering, object tracking, speed estimation and prediction of future states. It would therefor be preferable that the world module not only keeps track of an absolute world module, but also of a relative one which can be used when the robot isn't able to determine its own position.

Furthermore this will facilitate keeping track of the other objects in the game whenever the position estimate of the robots is updated by the vision system. Instead of having to translate and rotate all the objects according to the position shift, we only have to convert the relative position to absolute ones (using the new position estimate of the robot), making it much easier to adjust the entire model.

Sharing the complete world model. Instead of just sharing the ball and our own position with the other robots in our team, we should also share the estimated positions of the other objects in the game. As described in the previous chapter, sending information about the position of our teammates will also give them extra information for self-localization. Also information about the opponents can be used to increase the consistency of the world models of the different robots in our team and thus making it easier to make tacticle decisions. The sharing of the world models should be done by broadcast.

Observance the original architecture of the team. Some tasks are performed by the wrong modules. For instance, at this time the global method of the vision-based self-localization comes up with multiple position candidates which are all tracked by the local tracking mechanism of the vision software. This local tracking of the multiple candidates should (also according to the original design of the team) actually by done by the world module, since this is the module which has the most information about the position and is therefore the most suitable module for this task. The idea is to let the vision module send all candidates to the world module which on its turn uses for instance a particle filter approach to track all possible positions.

Multiple Hypothesis Tracking for matching observations. Matching incoming observations can sometimes give us more then one potential candidate. For now we just match the observation with the object that has the highest probability. All the other candidates are just being thrown away. This is however a waste of valuable information since it could very well be that the observation should have been matched to another object and that we will get evidence for this later on. We should keep track of all the possible candidates for an observation, resulting in a tree of potential world models, each of which has its own probability. Since we have limited system resources and trees of this kind tend to grow very fast, we need good pruning methods to make sure the trees will not get too big.

Opponent Modeling. In order to make good predictions of future states of the world it isn't good enough to assume linear motion for all the objects in the game, like we do now. It may work quite good for the ball, but robots tend to move around in a much more complex way. If we could find a way to model the behavior of the opponents, then we could use this model to predict the moves an opponent will make based upon the state of the world and this would mean that we could make much better predictions much further into the future.

Alternative sensors. The task of selecting which of the position candidates, generated by the global vision self-localization method, is the best, could be facilitated by using a compass. When we accurately know the orientation of the robot, the number of remaining possible positions will decrease dramaticly. Self-localization could also be improved by using an omni-directional camera instead of a normal one. A omni-directional camera will considerably increase the chance of perceiving a distinct landmark such as one or both goals. Also more lines will be seen, providing extra information for our vision-based self-localization.

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. 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.

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

[6] 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.

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

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

[9] C. Castelpietra, L. Iocchi, M. Piaggio, A. Scalzo, and A. Sgorbissa. Communication and coordination among heterogeneous mid-size players: ART99. 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.

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

[11] 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.

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

[13] P. Goel, S. Roumeliotis, and G. Sukhatme. Robust localization using relative and absolute position estimates, 1999.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge (UK) and New York, 2nd edition, 1992.

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

[23] 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.

[24] M. Spaan. Team play among soccer robots. Master's thesis, University of Amsterdam, 2002.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.