University of Groningen, The Netherlands

Artificial Intelligence

RuG
Rijksuniversiteit Groningen

A goalkeeper for the middle-size RoboCup league

Report on term of probation

January-June 2001

Computer Science Department University of Rome *'La Sapienza'*

Floris Maathuis [floris1@ai.rug.nl]
Supervisors in Rome, Italy: Prof. D. Nardi and Ing. L. Iocchi
Supervisor in Groningen: dr. N. Taatgen

31st October 2001

Abstract

This report presents Kerberos, the goalkeeper of the '*S.P.Q.R.*' middle-size RoboCup team. Its task is to defend the goal in a soccer match between fully autonomous robots. An overview of all the system is given, together with the choices made and the theory used while designing the various system components. Furthermore some test results are given. The most important topics discussed are hardware choice, image processing, stereo vision, omnidirectional vision, state information management and a decision module.

K1 960

Contents

1 Introduction

Grown-ups that toy around with soccer playing robots. For most people not exactly the image that fits serious and innovative science. Nevertheless this subject holds an enormous potential of possibilities. The game that every child can play without too much thinking is a huge challenge for computers. There is a huge amount of problems that need to be solved before a robot can play a decent game of soccer. Recognizing the ball, self localization, moving in a fast and reliable way, kicking the ball, cooperate with teammates and many other problems are came across that are far from easy to solve with currently known methods.

1.1 The '*RoboCup Federation*'

In 1997 the *'RoboCup Federation'*[1] was founded. This organization took as a long-term goal to "**develop a team of fully autonomous humanoid robots that can win against the human world soccer champions by the year 2050**" [Kitano et al. 1998]. At this moment we are still far away from this goal. Every single sub-task of the game can, even with state of the art methods only be solved in a way that would embarrass even the worst soccer player.

The RoboCup Federation tries to stimulate all kinds of artificial intelligence research. The soccer playing robot is both a standard problem as a landmark project here. As a standard problem it offers the possibility to evaluate various theories, algorithms and architectures. Moreover it can be used in robotics and AI education. As a landmark project it offers a well defined long term goal. The achievement of this goal itself will not generate significant economic and social impact. However it will be definitely seen as a great achievement of the field. This method has proved to be an effective way to promote engineering research.

To tackle the problem, at this moment the development of algorithms to handle the many sub-tasks is the main research topic. Therefore they are separated in groups. Each group deals with its own subset of problems. Furthermore each group has its own competition form to measure the performances of the different teams. The subgroups are:

1 Simulation League In this league the emphasis lies on tactics. The robots are in fact software agents that play on a virtual field.

2 Small-size League This group acts in the real world. All robots can use the information of a camera mounted above the field. This means that they always have complete information of the game (they know where all the objects in the field are).

[1] On www.robocup.org all information about the RoboCup Federation can be found.

3 Middle-size League The next step to a real soccer player is removing the camera mounted above the field. Now every player has to act with the information of local sensors.

4 Legged League The small and middle-size leagues are both leagues with wheeled robots. In the legged league the robots are robots in the more popular sense of the word: they have legs. Here the many problems introduced by the use of legs are tackled.

1.2 The middle-size league

The research described in this report is done for a robot acting in the middle size league, therefore a short overview of the regulations and rules of this league is given here[2].

1.2.1 The field, the ball and the robots

The matches are played on a field that looks somewhat like a soccer field, but much smaller. Figure 1 shows a map and a picture of a middle-size field. Around the field there is a border, so the ball can never go outside the field.

All objects in the field have a predefined color. These colors are called the RoboCup main colors here.

white border and lines

green the field itself

blue the goal of one team

yellow the goal of the other team

orange the ball

black the robots

cyan markers on the robots of one team

magenta markers on the robots of the other team

The ball is a normal soccer ball.

The robots have wheels in order to move. They may occupy an area of no more than 2025 cm^2. If the robot is shaped as a square, this means it can have sides of about 45 cm. They are allowed to have mechanisms to handle the ball, but are not allowed to grab it.

In a match two teams of four robots play against each other.

[2]The complete and actual overview of all the rules and regulations of the middle-sized robot league can be found on http://smart.informatik.uni-ulm.de/ROBOCUP/f2000

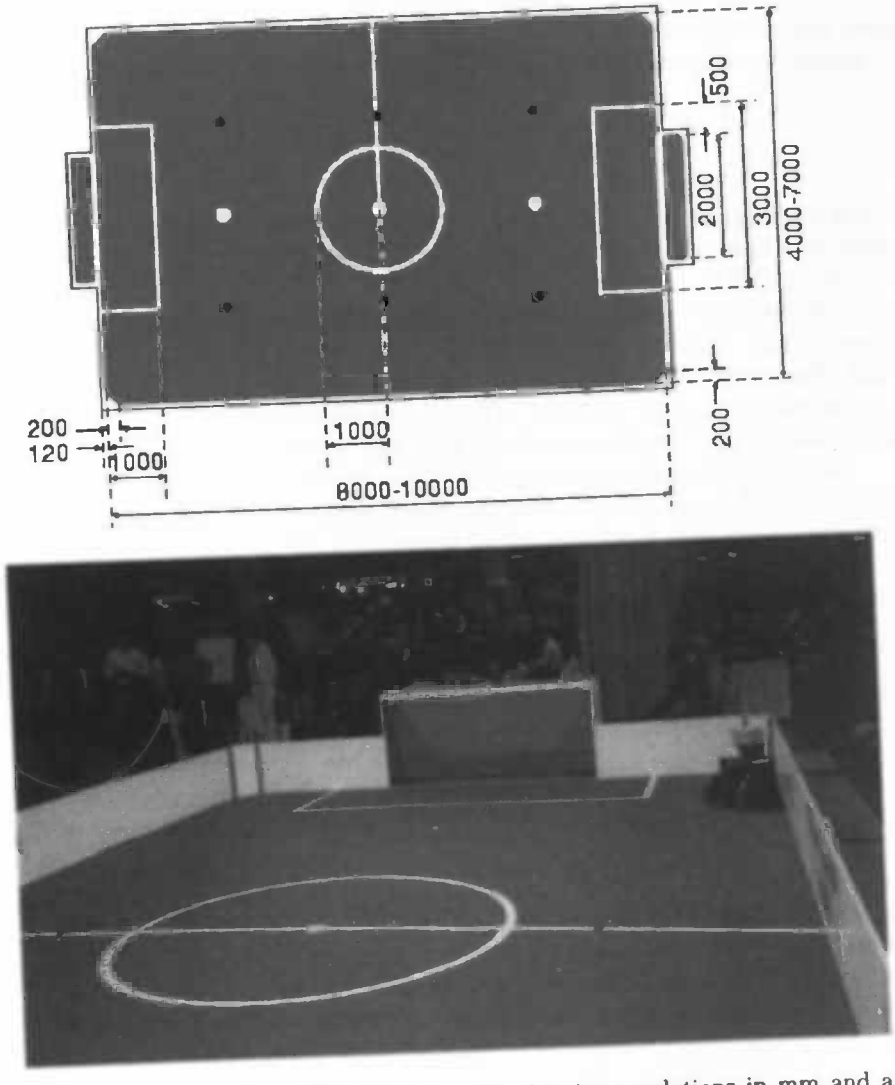

Figure 1: A map of a RoboCup field with the size regulations in mm and a picture of one of the world cup fields.

1.2.2 The rules

The rules are kept as close to the official '*FIFA*' rules as possible for wheeled robots. Some changes are applied to make the rules suitable for the specific physical conditions of the field and the robots or because their execution is still beyond the possibilities of the current methods.

In practice the rules mean that robots are allowed to do anything except for charging on other robots and stay in the penalty area to long (except for the goalkeeper).

1.3 *'S.P.Q.R.'*

The research in this report is done for the '*S.P.Q.R.*'[3] RoboCup team [Iocchi et al. 2001] of the University of Rome *'La Sapienza'*, Italy. Formally they participated in '*ART*' [Nardi et al. 1999] , the Italian national team. *'ART'* participated in several RoboCup tournaments with reasonable results. However, the members of *'ART'* decided to go their own way and the national team was dismembered. The Roman members now founded their own team. They already had some field players, but the goalkeeper of '*ART*' was built by another university. This meant that a goalkeeper for '*S.P.Q.R.*' had to be designed and realized. Working on the software development of the new goalkeeper would be my task.

The goalkeeper was called Kerberos, to the mythological dog that guards the Roman underworld. Its task would be to guard our goal from the opponents scoring.

1.4 Objectives of the project

The main objective of the project was to design a goalkeeper that could actually defend the goal in a RoboCup middle-size league match. This meant that the final test would be the 'reality' of a match.

My focus would be on the information processing software part of Kerberos. The design of the hardware and the drivers for it would be done by other members of our team.

The main part of the information processing I would work on was the vision system. The main goal here was to develop a system that could localize objects and do a prediction of their track. A side goal was the design of a module that could interpret this information and base an action on it. The complexity of this module would be dependent on the time that would be left for it.

2 Hardware design and realization

In this chapter the hardware on Kerberos will be discussed. The choice of the hardware for a robot is very important. It determines the performance in a high extend and is influenced by many factors.

[3] '*S.P.Q.R.*': Senatus Populusque Romano (For the Roman senat and people) and is the sign of the city of Rome.

2.1 Standard computer vs. specialized hardware

Basically there are two approaches in the choice of the processing hardware for a robot. The first one is to use a standard motherboard with a fast processor and do all the calculations in software. The typical way to process for example an image with this approach is first to grab it with a normal frame grabber card in one of the PCI slots and do all further calculations in software. The other approach is to use a smaller and less powerful on board computer and do the heavy calculations with specialized hardware devices. In such a system the hardware part already does a lot of preprocessing on the image before forwarding it to the on board computer. Both choices have their own advantages and disadvantages, which will be discussed here.

- The first difference between the two is speed. Usually hardware solutions calculate much faster than software solution so they are preferred when the highest possible speed is required. However both are subjected to a constant increment in speed, which means that the speed a hardware solution reaches today will be equaled by software within a couple of years.
- The second difference is the flexibility. Hardware solutions are usually highly specialized and can't be used for other tasks then the one they are designed for. This forces you to use a method that can't be changed without changing the hardware. Software on the other hand is very flexible. A new method can be implemented and used on the same system.
- The third difference is the physical size of the the two solutions. With the standard PC approach, the motherboard with the slots filled and the power supply forms a rather big block. That block takes a lot of space and has to be somewhere in the robot. This gives large restrictions to the overall design of the robot. This problem can be solved in some cases by using laptop components, but this restricts the number of input devices that can be used largely. A specialized solution is often designed to take only little space, although this is not always the case.
- Finally there is a big difference in price. Standard PC parts, manufactured in mass production, are relatively cheap. This not only makes it possible to build a system at low costs, but also makes an update affordable. If you want to increase the performance, buying a new and faster processor is sufficient. Specialized hardware and extra small computers on the other hand are much more expensive and to upgrade the system you have to buy completely new components.

In our goal keeper a standard PC motherboard is used. This solution has been chosen for various reasons. The first one is the price: our budget was relatively small so an inexpensive solution was needed. Secondly all the former research has been done on robots with standard PC motherboards, so the already developed software could be reused. Furthermore this former research has shown that with software solutions an appropriate processing speed can be reached.

Finally, the fact that the robot would become rather big was not a problem, since it would still be within the size regulations of RoboCup.

2.2 Self made vs. buying

Apart from the above discussion about standard PC parts against specialized hardware, there are some parts of the robot that are simply not standard and cheaply available. The main examples of these are the chassis, the motor controller and the motors with the wheels. However, again there are two choices. Firstly, there are some specialized companies that sell these parts. The other solution is to build components yourself.

Buying parts is usually expensive because robots aren't much sold yet, so the law of the large numbers is not applicable here. Furthermore their properties are designed for a 'general' robot, that is not necessarily designed for the task you want to use it for. On the other hand they deliver a plug-and-play solution that can be used without the aid of specialists. This is especially a big advantage for research groups with limited technical means or a more high level cognitive interest.

When the expertise is present, self-building the components is a good alternative for buying them. They can be build exactly fit for the purpose they will be used for and the costs are relatively low. On our goal keeper, we chose to build all the non-standard components ourselves. Mainly because we had the knowledge and experience inside the team and access to the needed tools. An extra advantage of course is the low price.

2.3 Our hardware

Outside the more general motivations for choices of hardware, every component has its own, more specific function and needs. Therefore here follows an overview of all the hardware on Kerberos with a small explanation for each component.

2.3.1 Chassis

The strongest and easiest to construct shape is a cube. Therefore the chassis is built out of aluminum cubes. There are three layers inside the robot. The upper layer contains the horizontally built in PC motherboard. The power supply and a compressed air tank are in the middle layer. The lowest layer has room for the motors of the wheels, the batteries and two pneumatic kickers. Three cameras are mounted on top of the robot.

2.3.2 On board PC

The motherboard that is used, has a 800 MHz Pentium III processor with MMX. There are 5 PCI slots and 1 ISA slot. They are all needed because multiple cameras are used, that all need a slot for the grabber card. All together there are two frame grabber cards, a digital camera card, a video card and a wireless

Figure 2: Kerberos and Flavio

communication card in the PCI slots and a motor controller card in the ISA slot.

2.3.3 Communication

For wireless network communication a wavelan card is used. The wireless connection has a speed up to 10 Mbit/sec. If only one team is using a channel and there is not too much noise, this is more than appropriate for the necessary communication.

In a RoboCup tournament, however there are a lot of teams using wavelan and also a high concentration of other wireless devices that can interfere with the communication frequencies (mobile phones, wireless microphones, wireless video connections, etc.). Despite efforts of tournament organizations of not letting too many teams use the same channels and restrict other wireless devices that can interact with the wavelan frequencies, the communication speed during matches often drops down to zero.

Unfortunately there is nothing you can do about this but to take in account that the network is often not reliable. Hopefully in the future a solution to this problem will be available.

2.3.4 Vision

The goal keeper gets all its information of the outside world by vision. In the choice of the camera for a robot there are usually two possibilities.

The first one is to use a 'normal' camera. This means that the view angle is small (about 60°) and the image looks like the image a normal photo camera takes. The advantage of such a camera is that things that are in the image have much detail and little distortion. Disadvantage is that because of the small view angle the ball is easily out of sight.

The other possibility is a camera that has a much larger view angle by using a mirror or a lens. Off-course this option has the advantage that a much larger part of the environment can be seen. On the other hand wide angle images have much more distortion, which makes it difficult to get detailed information.

On the goal keeper both camera types are used. A horizontally mounted camera with an 'omnidirectional' view, which means everything around the robot can be seen and a rotatable stereo camera to watch things in more detail.

Omnidirectional camera For the omnidirectional camera we had to choose between a fish-eye lens and a mirror. With the mirror option the camera would be mounted pointing upwards on the robot, with the mirror hanging above it (figure 3).

This option is particularly interesting because special shaped mirrors exist that can remove certain distortions in the image [Andrew et al. 1999]. Distortion removing in this way does not take any computation time and is therefore very interesting for real-time robotics.

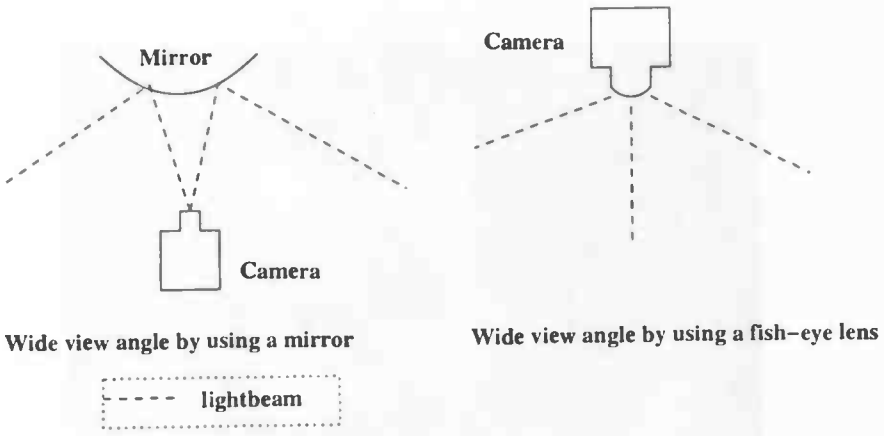

Figure 3: Two constructions for omnidirectional view.

In the end it became evident that it wasn't possible to put the mirror camera on the robot together with the stereo camera, because they both had to be mounted at the same place on top of the robot. Therefore a hanging fish eye camera is used, that fitted on the robot together with the stereo camera.

The fish eye camera has a large circular distortion (figure 4), that has to be dealt with in software undistortions.

For the omnidirectional vision a digital camera was used because more than two analogue framegrabber cards in the PCI slots gave huge hardware problems. The digital camera was also used to obtain a better image quality.

Stereo camera To watch objects in more detail there are two small angle cameras on board. The traditional approach for vision in RoboCup is to derive the position of seen objects with the information of only one camera. In this case the distance of an object to the camera is derived from the height of the object in the image. The closer an object is, the lower it is in the image. This method, however, assumes that the seen object is on the ground.

Since there exist robots now that kick the ball through the air, this assumption holds no longer. If the ball is far away on the ground it has the same height in the image as when it is close and in the air. To always be able to do a correct localization of the ball, we chose to use stereo vision. Therefore two cameras are used. Because they are small angle cameras and therefore have a limited view-field, they are mounted on a step motor to be able to track objects. The step motor is controlled by the on board PC.

2.3.5 Actuators

The final hardware parts to be discussed are the devices to let the robot act.

Firstly there are the wheels. The goal keeper has two powered wheels on its middle axis and two non-powered auxiliary wheels, one in the front and one in

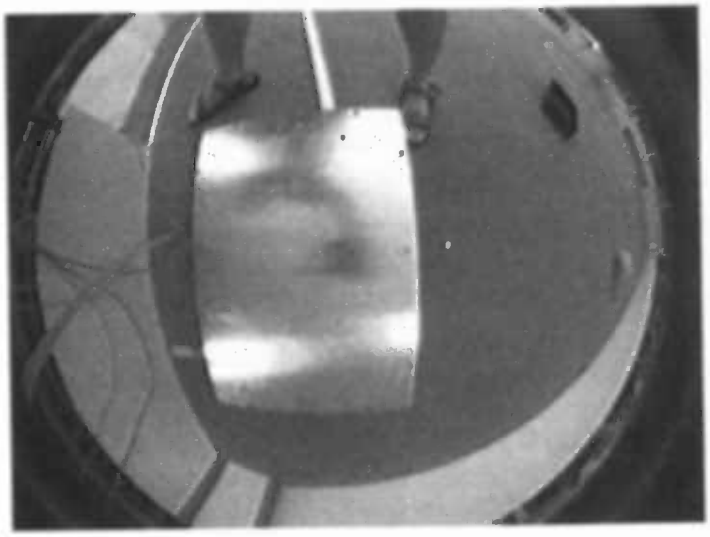

Figure 4: The large circular distortion of a fish-eye camera can be seen by the curved edges of the image.

the back. The powered wheels both have a separate motor and steering is done by giving them different speeds. The auxiliary wheels are only for stability and can move freely in every direction. The powered motors are controlled via a board in the ISA slot of the on board PC. This board is developed and built by our team member Flavio Cappelli [Cappelli 2001]. It does not only control the motors, but also returns their odometry. The wheels itself are small in order to have a fast acceleration.

To kick the ball there are two pneumatic kickers. They are powered by a tank with pressured air that has to be filled before the match. The release of the kickers is done by two electric valves that are controlled by the on board PC via a control board on the serial port, also built by Flavio Cappelli.

3 Software design

A robot that has to act in the real world is a very complex system with a lot of inputs and processing steps. If a system like this is not developed in a well structured way, it is impossible to keep track of all the dependencies. No clear overall architecture not only turns debugging into an even more time consuming activity than it already is, but also makes the system impossible for other people to understand and use.

3.1 The modular approach

The overall architecture that is chosen for the goal keeper is a modular one. This approach assumes that the overall task the robot has to do can be divided in a chain of sub-problems. For each sub-problem there is a module that solves it and sends the result to the next link in the chain. The modular approach has some very big advantages.

1. The division in sub-problems allows a straightforward and clear visualization of the system with diagrams. In this way it is always easy to find out what input a certain module uses and what processing is done on what data.

2. The theoretical modules can be easily implemented in an object oriented programming language and the chain like structure avoids a lot of debugging problems. During implementation it is possible to build a module, check if the output is correct and proceed with the next module. In this way you don't have to debug everything at once at the end.

3. The system is very flexible and easy to change. In a chain where each link receives a certain input, does some processing and forwards the result, it is very easy to replace or change a link.

There are some researchers who believe that a modular system is not the optimal architecture for an intelligent system (see [Steels 1994]). A modular architecture forces to solve problems in a serial way
(perception->modeling->planning->task execution->motor control).
These scientists say a better way is to define certain simple behaviors that receive input from the sensors and have a direct motor control as output. The overall output is the sum of the outputs of all these behaviors. Here the whole can be bigger than the sum of the parts and very complex behaviours can occur.

However the modular approach was chosen because it fitted more in the traditional engineering oriented tradition of the '*S.P.Q.R.*' RoboCup team.

3.2 Layered structure

The modular system on Kerberos can be divided in 5 main parts or layers (figure 5). All the modules in each layer have similar tasks. A short overview is given here. Two layers are in fact hardware layers and are already discussed in the previous chapter, three are software layers and will be discussed in the next chapters.

The first layer is a hardware layer and contains the sensors that can give information about the state of the outside world. The provided information is mostly of a low level and needs to be processed for further use. An example is the image provided by a camera, which contains a lot of visual information for humans, but for a computer it is just an array of pixels.

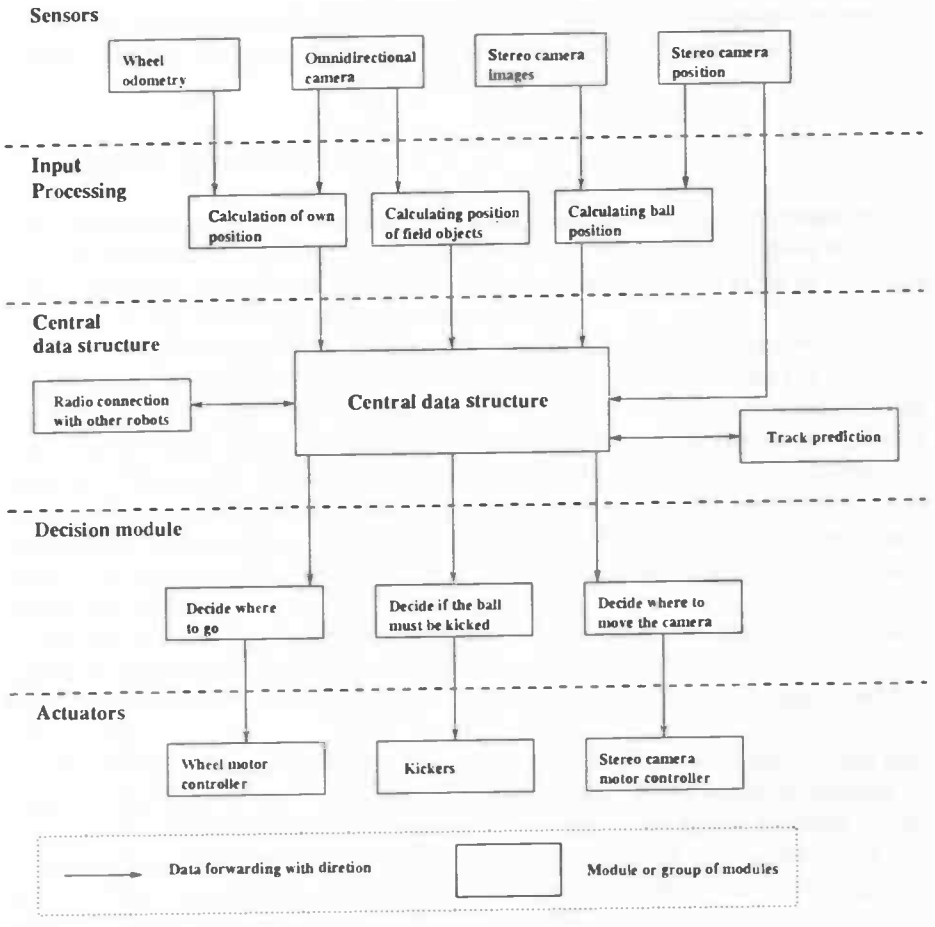

Figure 5: The five main layers

The second layer extracts higher level information from the low level output of the sensor layer. For example, this processing layer calculates the position of the ball from an image of the ball.

The third layer is a database that contains all the high level information that the robot has. It is the representation of the world as the robot sees it and is therefore called Robot Perceptual Space (RPS). In our case this representations are the positions of all the objects in the field.

The fourth layer is a decision layer. It decides, based on the representation of the world in the RPS, what action has to be taken.

The final layer is a hardware layer again. It contains the devices that can modify the state of the world and executes the actions decided on by the decision layer. Examples of these devices are the motors to move the robot and the pneumatic kickers to kick the ball.

During the implementation I used various modules and libraries designed by others, which is always mentioned. Everything else is implemented by myself.

4 Input processing

The sensor layer has an output of very rough low level sensor information. To be able to do something with it, it has to be processed further. This is what the next layer does. It uses the output of the sensors in the input layer to compute the state of the world in higher level knowledge.

The RoboCup world is a highly predefined sub part of the 'real world'. It is exactly known what its basic properties are. The shape and colors of the field are known, so we don't have to bother about finding out what the world looks like. On the other hand we don't know the dynamic properties of the moving objects. This means the objects in the field can be divided into two classes, the objects that have a fixed and known position, like the goals and the lines, and the objects that can move, which are the robots and the ball. Both classes have their own typical properties, which can be used to obtain high-level knowledge about the environment.

In this context the terms absolute and relative position are used. The absolute or global position of an object is its position in the coordinates of the field. The relative or local position is the position in the coordinates of the robot itself.

The known position and the physical properties of the members of the first class makes them perfect for the use as landmarks. If the relative position of some landmarks is known, it is possible to calculate the global position of the robot (self-localization). Because the global position of the members of the second class is not known, they can't be used as landmarks. The knowledge you want to obtain is the location in the field and the moving direction.

Because the main sensors that provide low level information are cameras, the first task to obtain the wanted world knowledge is to recognize which parts of the images are which objects. After that we try to do a self-localization, also based on the odometry of the wheels. Thirdly we try to calculate where all the moving objects are in global coordinates. Finally we try to predict the track of the moving objects. In theory these steps provide us all the high-level information we can obtain from the sensors. The rest of this chapter will contain a more detailed overview of the used methods and their implementations.

4.1 Image processing with MoS^2

For the implementation of the modular approach that is used, a predefined software structure is developed by our team member Danielle Baldassari [Baldassari 2001]. This structure is called MoS^2 (Modular Software Modeling System) and makes it possible to implement, visualize and debug a modular system with C++ in an easy an standardized way. The most important part of MoS^2 is an object

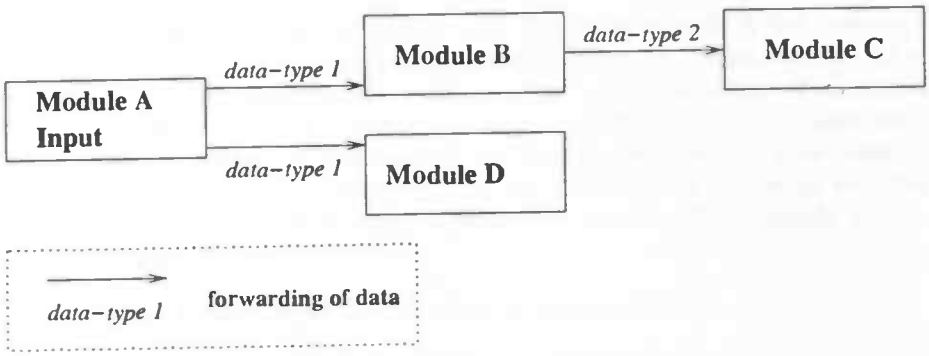

Figure 6: An example of a modular system

hierarchy that gives the data processing links of a modular system a standard form. Each link of the chain or pipeline is derived from the same base classes. There are three of them.

RObject This class provides some general use functions as the creation of debug streams and the returning of the name of the derived class. All modules are derived from this class.

RDataSink This class has methods to receive data from other modules. All modules that receive data to process are derived from this class.

RDataSource This class has methods to forward data to other modules. All modules that have to forward data to other modules are derived from this class.

To make the different modules compatible with each other, also the data that is sent between the modules is put into a standard form. All data types are derived classes of RObject. All modules have a list with the data types they can receive or send. RDatasources can be connected to RDataSinks, provided that their data types are the same. In this way a whole pipeline can be constructed by connecting the right modules with each other. This is implemented by giving each RDataSource a list with the RDataSinks it's connected to. To make the use of this system clear the implementation of the modular system in figure 6 will be given here as an example.

We assume that classes for the modules already exist. They are all RObjects. modules A and B are RDataSources and modules B, C and D are RDataSinks. To create the pipeline, we have to connect the right modules with the right data types:

1. A.addSink(B,data type1)
2. A.addSink(D,data type1)
3. B.addSink(C,data type2)

Now the pipeline is ready and can be run. By giving module A the command to read the input, the whole pipeline is worked through automatically. After module A has read and checked the input, the data is emitted. This means that for all the modules A is connected to (B and D) a function is called with as argument a pointer to the data of A. Each of this modules now does its processing and emits the created data to the modules it's connected to. The modules are worked through with a depth first search. The example will be worked through as follows:

1. Module A reads input and forwards it to module B
2. Module B does processing and forwards result to module C
3. Module C does processing
4. Module A forwards input to module D
5. Module D does processing

With MoS^2 also goes a graphical user interface, called MoS^2builder. In this application built modules can be opened as blocks and connected into a scheme like in figure 6. The so created pipeline can be ran and the parameters of the modules can be adjusted with the aid of tools to watch the data at every point in the chain. MoS^2builder is a great help in adjusting and debugging a pipeline.

Inside the data input data processing layer of the goal keeper there are two main pipelines. One with the stereo camera as input and one with the fish eye camera.

4.2 Object recognition by color

The objects in the field are purely recognized by color. This is possible because every type of object has its own specified color. The search for objects in an image is done in two steps.

4.2.1 Color filtering

The first step to recognize objects by their color is color-filtering. The image that comes from the camera is an RGB image. We want to know for every pixel to what main color of the RoboCup field it belongs.

Herefore there is a special image type, the 8 color image. In this image every pixel is a byte with a value from 0 to 8. Every value corresponds with one of the colors that occur in the RoboCup field.

For all the mapping methods lookup tables are used. In this case a list with all the possible RGB values and the 8 color value they map on. The use of lookup tables is only possible if there are not to many values per pixel. When 24 bits per pixel are used, the lookup table would be $2^{24}bit \cong 2Mb$, which is far to big to use. To have a manageable size of the lookup tables, 15 bits per pixel are used decoded in a byte (5 bits for R,G and B and one empty bit), this

lookup table takes only $2^{15}bit \cong 4kB$. All images that come from the cameras are converted to such RGB 555 images.

The mapping is not as easy as it seems. The main reason are changing light conditions. An object with a certain color maps on different RGB values when the light conditions are changed. For the color mapping we had two methods available.

RGB-set filtering In this method every RoboCup color is defined as a set of RGB values it falls in. To calibrate it an RData tool is available where squares can be chosen out of the image with the mouse and a RoboCup main color can be assigned to it. This gives a list of RGB values that belong to a certain main color. When filtering an image this list can be used to look up to which RoboCup main color an RGB value is mapped.

The RGB-set method can do a very good color mapping with little noise. However it is very sensitive to changing light conditions. When it gets darker or lighter, the filter becomes useless.

HSV-threshold filtering The second method is HSV-threshold filtering. Before filtering, all the RGB values are transformed to the HSV-color space. In this color space colors are not described by a red, green and blue parameter, but by a hue, a saturation and a value parameter. This parameters describe colors more in the terms humans do. When it gets lighter, the actual color (hue) doesn't change, but only the intensity (value). When a color is mixed with more other colors (the saturation gets higher), it gets closer to white. When the value gets lower, a color gets closer to black. In theory this means that RoboCup main color can be defined as a part of the hue scale and black and white with the saturation and value parameters. This principle is used to map every HSV value to one of the 8 color image colors.

The big advantage of this method is that it is very robust with light changes. The properties of the parameters take care of this. The disadvantage is that the edges between the different hue values are not very sharp and a lot of noise appears on them.

Because the light conditions in a RoboCup field are well defined and no large changes appear, the RGB-set filter is used for the color filtering on the goal keeper.

4.2.2 Blob extraction

After the color filtering it is known for every pixel to which color group it belongs. We now assume that objects appear as blobs of adjacent pixels of the same color group, a robot for example is a blob of magenta and black pixels. To find all the objects of a certain kind, the image is scanned for blobs of pixels of the corresponding color(s). With some filtering on the size of the blobs, a reasonable overview of the objects in the field is obtained. However this method has some disadvantages.

- Individual differences between objects can't be seen. A magenta and black blob is just a robot, which robot is completely unknown.
- When two objects of the same kind partly overlap each other they become one object.

For this reason a more refined object recognition mechanism should be a priority point in future research.

4.3 Stereo vision

As written before, stereo vision is chosen to localize objects that are in the air, in particular a high kick of the ball. The processing of stereo images has, apart from the problems of normal image processing, many extra factors to take into account. In this part a short overview of the used stereo theory is given, after this various methods to apply this theory are discussed, finally an overview of the stereo vision pipeline that is running on Kerberos is given.

4.3.1 Stereo theory

The reconstruction of the distance of an object out of the information given by two cameras is done by using triangulation. This method makes some assumptions about the cameras.

1. The two cameras have a pinhole model. This means that the lens is an infinitely small hole.
2. The two axes of the camera are parallel and the projection planes are the same.
3. The object is infinitely small, so there is only one light beam that projects the object on the projection plane of the camera.

Figure 7 illustrates this assumptions and is an upper view of the two cameras.

The triangulation method that is used, is based on the conformity of triangles. The following triangles are conformative.

1. The triangle with the sides [C1 C] and Z and the triangle with the sides [P1 A1] and [C1 A1].
2. The triangle with the sides [C2 C] and Z and the triangle with the sides [P2 A2] and [C2 A2].

They are formed by a line crossing two parallel lines. This creates two triangles with two similar angles, which means they are con formative. The value we want to know is Z, the distance from the object to the pinhole axis. We know that:

$$\frac{Z}{f} = |\frac{C_x - C_{1x}}{P1_x - A1_x}| \tag{1}$$

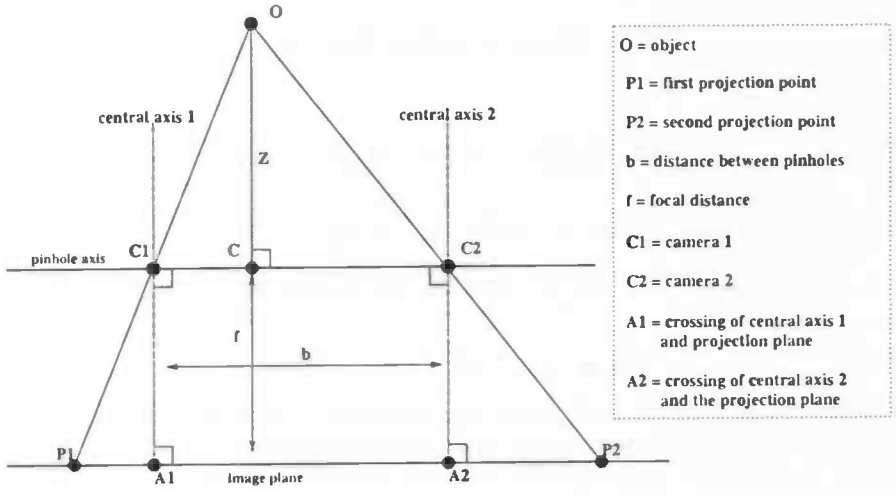

Figure 7: The theoretical scheme to do triangulation with two pinhole cameras.

and

$$\frac{Z}{f} = |\frac{C2_x - C_x}{P2_x - A2_x}| \tag{2}$$

f = the focus distance of the camera

Z = the distance from the pinhole axis to the object

C = the point where the perpendicular of the pinhole axis through the object starts.

C_x = the x coordinate of point C

$C1$ **and** $C2$ = the pinholes

$P1$ **and** $P2$ = the projection points

$A1$ **and** $A2$ = The points where the camera axes cross the projection plane

$C_x - C1_x$ and $A1_x - P1_x$ always have the same sign because they represent the size of two vectors with the same direction.

$$\Rightarrow \frac{Z}{f} = \frac{C_x - C1_x}{P1_x - A1_x} \tag{3}$$

$C2_x - C_x$ and $A2_x - P2_x$ always have a different sign because they represent the size of two vectors with opposite directions.

$$\Rightarrow \frac{Z}{f} = -\frac{C2_x - C_x}{A2_x - P2_x} \tag{4}$$

$$\Leftrightarrow \quad Z * (A1_x - P1_x) = f * (C_x - C1_x) \tag{5}$$

and

$$Z * (-A2_x + P2_x) = f * (C2_x - C_x) \tag{6}$$

$$\Leftrightarrow \quad Z * (A1_x - P1_x - A2_x + P2_x) = f * (C2_x - C1_x) \tag{7}$$

$C2_x - C1_x$ is the signed distance between the two pinholes

$$\Rightarrow \quad Z = \frac{f * b}{A1_x - P1_x - A2_x + P2_x} \tag{8}$$

This means that with a given camera setting the distance to an object in 3D is a function of the distance between the projections of that point in the left and the right image. This distance is called the disparity.

When Z is computed, the 3D x coordinate X of the object can also be calculated (Figure 7). The next formula gives the 3D x position relative to the left camera ($C1$):

$$X = C_x - C1_x = \frac{Z * A1_x - P1_x}{f} \tag{9}$$

The 3D y coordinate of the object relative to the camera can be calculated in a similar way as the x coordinate, using the side view of the cameras.

4.3.2 Errors (occlusions/movements/synchronization)

To apply the described theoretical stereo method, a lot of difficulties have to be overcome. In the first place, the assumptions that are made in the theory do not hold in reality. However with various computational methods they can be approximated. Furthermore various types of noise should be taken into account, introduced by the technical limitations of the camera, the limitations of the computational devices. In this part all these errors will be discussed.

Not holding assumptions The first group of errors is caused by the difference there always is between theory and reality.

Lens problems For our purpose a pinhole camera is not useful. Because the opening for the light is very small (as small as possible for a good image) the exposure time has to be quite long to have a good image[4]. The robot however operates in a rapidly moving environment and absolutely needs a short shutter time to avoid blurred images. Therefore a glass lens is used, which lets through a much larger amount of light. Disadvantage of such a lens is that it introduces

[4]On www.photographytips.com Home > Equipment & accessories > Camera > Pinhole camera an overview of the advantages and disadvantages of pinhole and glass lens cameras is given.

a circular distortion in the image. This means that the light beam doesn't go straight from the object to the projection plane but is slightly curved, specially at the edges of the image. The error that is introduced in the triangulation method is dependent on the size of the distortion.

It is possible to remove the circular distortion with a projection of the image points. When considering the circular distortion as a function on every light beam through the lens, it is possible to remove it by applying the reverse transformation. This procedure is called rectifying the image.

Camera positions The second assumption for the triangulation method doesn't hold in reality either. In practice it is hardly possible to make the projection planes the same and the axes of the camera parallel. The parameters are very small and hard to measure and adjust.

Also this error can be removed by software methods. To make the assumptions holding two transformation have to be done. Firstly the images have to be transformed to correct for the differences in internal camera parameters. After this the images look like they are made by two cameras with exactly the same model. The next step is to correct for the error in the position of the two cameras according to each other. Here the images are rotated and translated to the same projection plane.

Large objects The assumption that the object is an infinitely small point can't be satisfied either. In reality you want to know the position of larger objects, like the ball. Because the theory just tells us how to reconstruct a distance from two points we have to be careful here. The methods to overcome this problem will be discussed later.

Camera limitations The perfect stereo camera would have an infinitely small shutter time, an unlimited resolution, a perfect image and the cameras would take their images at exactly the same moment. Unfortunately such a camera does not exist and we have to do with those that are available. This introduces some errors and limitations in the processing.

Quantization error Inside a camera there is an array of photo cells on the image plane. This array is used to measure distances in the image plane. Every photo cell represents one pixel of the image. This means that the resolution of the image can't be smaller than the size of these photo cells. So every position in the image has a quantization error of 0.5 pixel.

The actual error in the 3D position caused by an error of 0.5 pixel varies with the distance and the pixel size. With an image resolution of 320x240 pixels the 3D error varies from the order of one centimeter with an object distance of 50 cm till the order of one meter with an object distance of 8 meters (about the RoboCup field field length).

The actual values of the pixels have a quantization error too because they are digitized.

Shutter time If the shutter time is not infinitely small it will also introduces noise. An object or the camera that moves while the shutter is open can smear the image. The cause of this effect is that the same point of an object passes over more than one photo cell. This means that that the size of this error is a linear function of the speed in pixels of an object with reference to the camera. If the movement of an an object stays under one pixel there will be hardly any smearing, but if the movement goes over several pixels the effect will be much larger. In this context there are two causes for a smeared image.

1. An object that moves itself. The speed in pixels is a function here of the speed of the object in the field and the distance to the camera. With constant field speed the pixel speed increases when the object is closer and with constant distance the pixel speed increases when the field speed increases.

2. The cameras that are moving. The pixel speed is a function of the angular speed of the cameras and the distance to the object in this case. When the angular speed increases, the pixel speed increases and when the distance to an object increases, the pixel speed decreases.

The effect of this noise can be compared with the effect of a bigger pixel size. The faster the movement the bigger the equivalent pixel size.

A good way to reduce this noise is to track the object with the camera. In this case the relative movement of the object and the camera stays close to zero and little smearing will occur.

Synchronization Moving objects or cameras can also introduce a synchronization error. This error occurs when the two cameras don't take their images at exactly the same moment. The object to be localized could have moved between the two captures, which means that it's not at the same position in both images and a localization error is introduced. This noise will only be significant when very fast movements occur. Tracking the object with the camera is again a good solution.

Image noise The last problem due to camera limitations we have to deal with is image noise. It is the result of all kinds of electronic noise inside the camera and affects the overall image quality. The size of it is directly dependent on the quality of the camera and the frame grabber card. The exact form and size is dependent on the specific hardware used, but in general it introduces an uncertainty in the RGB value of a pixel.

Computational limitations As said before the robot has to act in a real time environment. This means that the whole process of capturing the images, analyzing them and computing the position of seen objects has to be done several times per second. Therefore we have a limited time for all the computations to be done. Because most image processing methods are a linear function of the

number of pixels, working with low resolution images gives a large increment in speed. However, by decreasing the resolution of an image also the quantization error is increased.

4.3.3 The problem of correlating objects

To apply a triangulation a point in both images of the stereo image pair, representing the same point in the real world, is needed. What we want to know, however, is the position of an object much larger that a point. To overcome this problem, two methods are possible. These two methods and their problems have been verified with specific experimental tests.

Correlating points In this method it is tried to do a triangulation for each point of the object and then take the average of the calculated positions as the position of the object.

The image position of the objects is already determined with object recognition by color (see 4.2). The pixels of the object are taken as points. To determine which point in the left image corresponds to which point in the right image, a correlation is done. Hereby for each pixel of an object in the left image a correlation window with itself and the pixels around is taken. This window is used to do a cross correlation with the pixels of the right image. In this way the area in the right image can be found that has the biggest correlation with the correlation window. The center pixels of the correlating areas in the left and the right image are considered the same point in the real world. Now the triangulation method can be applied on these points.

A great disadvantage of this method is that its effectiveness is very dependent on the features of the objects to correlate. An object with a black and white pattern on it can be correlated much better than just a white object. In the latter case just for the borders a decent correlation can be found.

Moreover it can happen that when two objects are close in the image but far in depth, one of them 'takes away' the correlation from the other. When trying to correlate the edge points of an object, it is possible that an other object lays inside the correlation window of this points. When the edges of this second object are much sharper then those of the first, it can happen that a correlation is found of the second object. A good example of this is seen when a clear, thin line is behind the ball. In figure 8 is shown what such a situation might look like. The upper block is a part of the left image, the lower block of the right image.

The black bars correspond to the vertical line, the dashed blobs at the right are the ball. The small dashed square in the upper image is the pixel for which we want to find the corresponding pixel in the other image. The small dashed square in the lower image is the corresponding pixel found with correlation. However, this is a match on the black bar because it has stronger features than the ball. It's clear that the wrong pixel is found.

The described effect introduces an unacceptable large error in the computation. The size of this error is about half the width of the correlation window

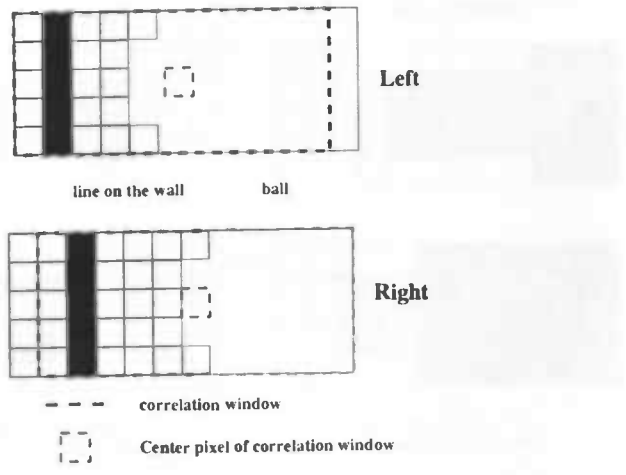

Figure 8: An example of two objects close to each other that can give problems with correlation.

minus 2 pixels, which is about 3 pixels. When a 320x240 pixel resolution is used, this error goes from a few centimeters when the object is close, to several meters when the object is at the other side of the field.

Correlating a whole object The other method to determine the real world position of an object is to represent the object as points in both images and do the triangulation on these points. To obtain a point representation of an object some kind of average has to be taken. Here is chosen to use the center of mass, this means that simply all the positions of the pixels of the object are averaged. A sub pixel approximation of the position is obtained in this way because the average of many pixels are taken.

This method works very good when the whole object is seen by both cameras. However, when there is another object that partly covers the object we want to do the triangulation on, there is a problem. This problem occurs because the object recognizing module just delivers the center of mass of the seen part of an object, which is different from the center of mass of the whole object.

Figure 9 illustrates what happens to the centers of mass of an object when it is partly covered.

In figure 9 an object is seen that has a disparity of 2 pixels. Without another object occluding it, the centers of mass have a disparity of 2 pixels and the triangulation will give the right distance. However, when another object (occluding object) is partly in front of our object (occluded object) there is a problem. The object in front of our object has a bigger disparity and therefore a bigger shift in the right image then our object. This means that in the right image a bigger part of the object we want to know the disparity of is occluded than in the left image. Therefore the disparity of the centers of mass are differ-

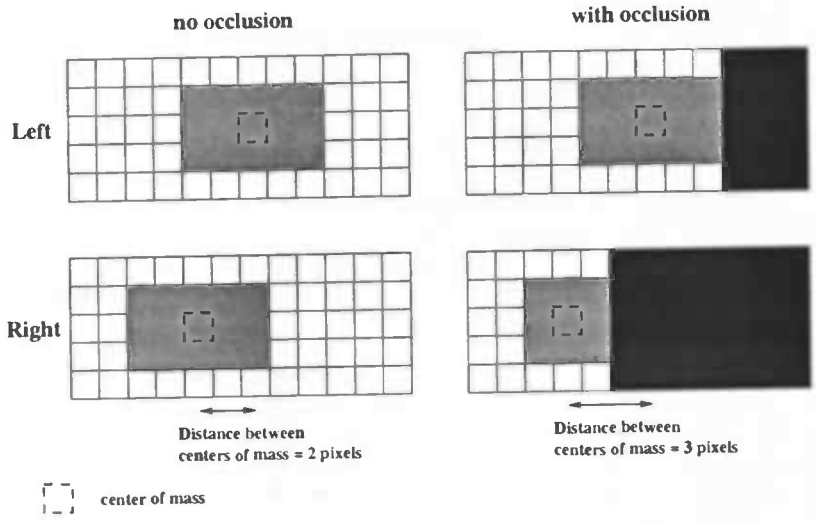

Figure 9: The effect of an occlusion on the center of mass.

ent from the disparity of the objects and a wrong distance will be found with triangulation.

This error can be corrected. When an object is occluded, it has a different size in the left and the right image. By looking in which of the two images the object is larger, we can even know on which side it is occluded. By using a model of the object, it is possible to do a correction for the occlusion.

It is assumed that the un-occluded object appears as a rectangular in the image. The occluding object is also assumed a rectangular that is shifted over the object we want to find the disparity of. Figure 10 shows the model for an occlusion on the right.

When in the two images the same part of the object is seen, it is possible to do a correct triangulation again. This is can be done because in this case the centers of mass are shifted equally in the two images. Therefore it is calculated what the center of mass in the right image would be if the same part was visible as in the left image.

$$x_{ccm} = x_{vcm} + \frac{W_l - W_r}{2} \tag{10}$$

x_{ccm} = the corrected center of mass

x_{vcm} = the center of mass of the visible part of the object

W_l and W_r = the widths of the object in the left and the right image

The so obtained corrected center of mass can be used for triangulation. Occlusions at the left side can be corrected in a similar way.

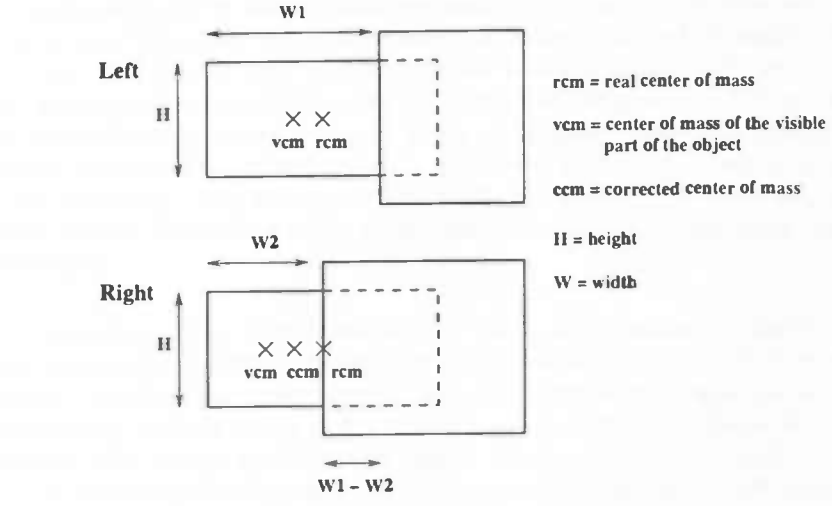

Figure 10: Model of the object to correct for occlusions at the right side.

Off course the assumption that the object and the occlusion are rectangles doesn't hold in reality. However all the objects in the RoboCup field are relatively close to rectangles and in experiments the correction method has proved to be appropriate for its task.

In the goal keeper is chosen to use the latter approach. It takes less calculations (no correlation and no reconstruction for every point), which is faster and it is more precise because of the ability for correction.

4.3.4 The SVS tool

With many of the stereo computations the SVS-tool is used [Konolige et al. 1999]. SVS is a software tool for stereo vision. It includes several functions that cover a large part of the methods needed for stereo calculations.

Calibration The first function offered is a calibration. This method calculates the circular distortion correction that is needed to be able to use the pinhole model automatically. Furthermore the translation and rotation functions to line out the two stereo images are calculated automatically.

Once calibrated, the computed variables can be used for every image pair of the stereo camera. For this purpose there is a rectify function that takes as input the original images and gives as output the undistorted outline images. Because the MMX part of the motherboard is used, this goes very fast.

Unfortunately the function is designed for gray-scale 1 byte per pixel decoded images. This means it can't be used for RGB 555 images, which are decoded in two bytes per pixel. However for the 8 color images it is useful, because they are also decoded with 1 byte per pixel. When undistorting these images, there

is a phenomenon we have to take into account. It can happen that two pixels of the original image are mapped on one pixel in the undistorted image. Therefore the rectify function does an interpolation between pixel values in some cases. In a gray-scale image this is a perfect solution of the problem. When taking the average of a black point (value 0) and a white point (value 255) you get a gray point (value 127), which is correct. However, an 8 color image is a symbolic representation. The average of black (0) and white (7) is cyan (3), which is not correct. This effect needs to be taken into account when using the rectify function.

Correlation The next function SVS offers is a correlation function. This function takes the stereo image pair as input and tries to correlate every pixel with a pixel of the other image. For this correlation a window around the concerning pixel is taken and a match for it is tried to be found in the other image. The output is a disparity image, this is an image in which every pixel has the value of the disparity to the corresponding pixel in the left image. This function is only used in tests, where the problems described in 4.3.3 where discovered and was decided not to use this method.

3D reconstruction Finally there is a 3D reconstruction function. Given a point in the image and its disparity this function can reconstruct the 3D position of it relative to the stereo camera.

4.3.5 Scheme of the components

After having considered the needed theory and the errors that can occur, now the actual implementation of all the modular components will be discussed.

In figure 11 the whole chain is shown. As can be seen only the ball is localized with stereo vision. This has various reasons.

1. The ball is the only object that can go through the air, so only here stereo localization is necessary.
2. There are no modules yet that can distinguish between individual robots. This means it not yet possible to know which robot in the left image is which robot in the right image so there are no objects to do a triangulation on. Because there is only one ball, both the balls in the left and the right image are the ball and a triangulation can be done.
3. The stereo camera has a narrow view angle and therefore has to track an object by moving. Only one object can be tracked at the time. It is logical to take the most important object in the field, which is the ball.

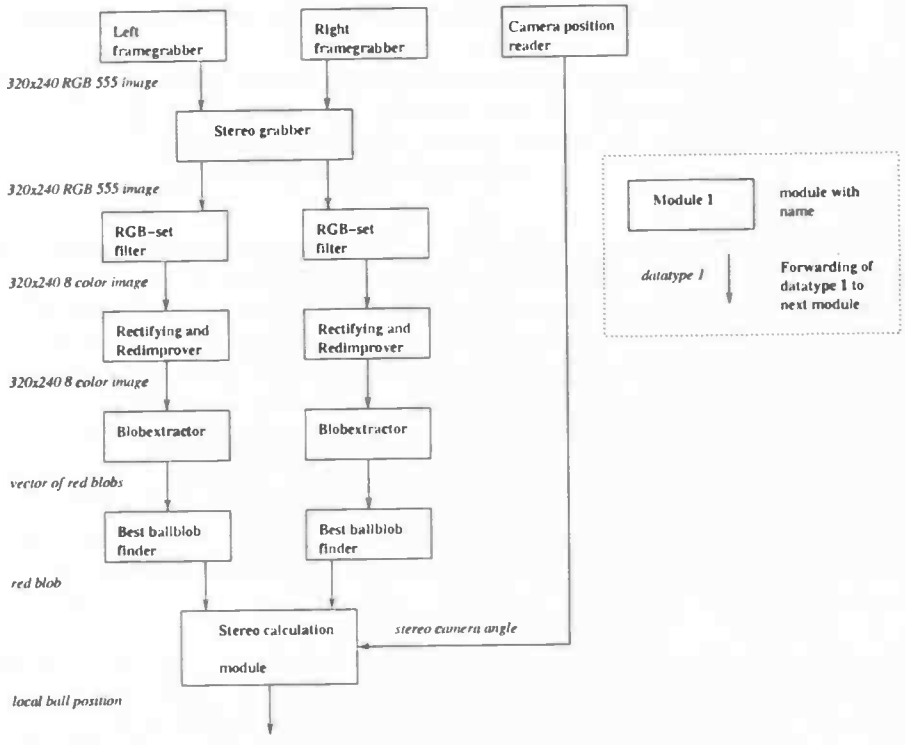

Figure 11: The modules for the stereo localization of the ball.

Frame grabbers The frame grabbers grab a 320x240 pixels RGB image with 5 bits for every color(RGB 555 image). The size of the image gives a reasonable resolution but is still small enough to do real time calculations. This module is implemented by Daniele Baldassari.

The used cameras are not synchronized because they have no linking abilities. The grabbing are two separate processes that fill an array every time a new image is taken from the camera. Yet we want the two images as close to each other in time as possible. Therefore the stereo grabber module is put in the pipeline. This module calls the functions that read the image arrays directly after each other. This means the arrays are read at about the same moment and the time difference between the two images is maximal $\frac{1}{framerate} = \frac{1}{25Hz} = 0.04\,s$.

RGB-set filtering Now the image from the frame grabber is converted to an 8 color image. Because we only need the ball, the filter is calibrated in such a way that everything becomes black, except for the ball, that is red. Also this module is implemented by Daniele Baldassari.

Rectifying and red improvement After the image is converted to an 8 color image the image is rectified with the aid of SVS. This introduces some noise in the 8 color image, caused by the interpolation between pixels that SVS does. This introduces strangely colored pixels at the border of the ball. However, we know that everything that is not black in the image is red and this knowledge is used to make every pixel that is not black red again in the rectified image.

Another kind of noise that can be clearly seen here is the image noise. This results in small black parts and bad edges in the objects. This noise is filtered away with a kind of averaging filter. When a pixel is red, all the pixels around it are made red to. The disadvantage of this filter is that the red noise in the rest of the image is enlarged too, but the ball still stays the largest red blob in the image.

Blob extraction From the rectified and improved image are now taken all the red blobs. They are all put in an array and forwarded to the next module. This module is implemented by Daniele Baldassari.

Best ball blob finder From the array of red blobs, of which many are just noise, now has to be selected the one that is the ball. As a feature the size of the blobs is taken. It is known that the ball is the largest red object in the field. So the largest blob of the array is taken. When the ball is behind another robot or just too far away, it is possible that some noise becomes the largest blob. Therefore there is a minimum size for the largest blob to be accepted. It can happen that in this way the ball is not recognized when it is at the other side of the field, but that is not a big problem because in this case there is no danger for the goal.

Also the shape of the ball is used. Every blob has a value for roundness, that goes from 0 (a line) to 100 (a circle). Even when the ball is partly covered

by a robot, it still has quite a big roundness. Therefore the largest blob is also tested for roundness and refused as the ball when it is to low.

Stereo calculation The ball blobs of the left and the right image finally arrive at the stereo calculation module. Here firstly the disparity of the centers of mass of the blobs is calculated. This disparity is corrected, based on the dimensions of the blobs. The corrected disparity is used to reconstruct the 3D position of the ball with aid of the SVS reconstruct function.

Now that we know the position of the ball relative to the stereo camera, we have to obtain the local position of the ball. The relative position of the camera to the robot is used to rotate and translate the ball position to local coordinates. Hereby the position of the camera step motor is taken into account. The so obtained values are the output of the stereo pipeline.

4.3.6 Performance

The stereo localization has a reasonable performance. When the camera is not moving, the localization error goes from about 5 cm when the ball is close to the robot (about 50 cm) to about 1 m when the ball is at the other side of the field (about 8 m). This is an error of about 10% of the measured distance. The main reason for this error is image noise. For reacting behavior this error is not too big because the absolute error gets smaller when the ball gets closer.

The correction for occlusions works well. The error with occlusions is not significantly bigger than without.

Moving the camera introduces extra noise in the localization of the ball. At this moment the camera has just one speed, top speed. This causes so much smearing in the image that no useful information can be taken out of it. A better solution would be to follow the ball smoothly, unfortunately there has not been enough time to implement this behavior. At this moment the noise is reduced by moving the camera as little as possible. The few bad images by the short moving are taken for granted.

4.4 Omnidirectional vision

Objects that are not in the air are localized with the omnidirectional fish eye camera. The original plan was to localize all the objects in the field and do a self localization with the camera information. However the fish eye camera arrived just shortly before the tournament, which meant a big part of the planned functions where left away because of time shortage.

4.4.1 Self-localization

The first part that is left out is the self localization of the robot by vision. We wanted to use a method where the local view of the lines of the field is fitted into a global map of the field with aid of the Hough transform([Iocchi et al. 2000] and [Grisetti 2001]). When the local view is correctly fitted into the global

map, the global position and orientation of the robot is also known. The lack of localization by vision forced us to rely completely on the odometry of the wheels for the global position and orientation of the goal keeper.

Odometry is a very unreliable source for localization. Its error is not so large on small movements, but they add up in time. This makes the position more uncertain with every movement the robot takes. Because the goal keeper just moves in a very small area, we hoped the localization with odometry would be precise enough for the duration of one half, after which we could reposition the robot. Anyway a good self-localization system should be the biggest priority point for the further development of Kerberos.

4.4.2 Ball localization

The ball is the only moving object in the field that is localized with the omnidirectional camera. Considering our time shortage this seemed the best solution because of various reasons.

- The task of the goal keeper is to prevent the ball from going into the goal. For this reason it needs to know where the ball is. Whether there are other robots near the ball is less important. The omnidirectional camera offers the perfect sensor to see the ball wherever it is.
- The stereo camera is guided by the omnidirectional camera. The first one gives the direction in which the ball is seen. This direction is unchanged when the ball is in the air. The latter rotates to that direction and gives a more precise view.
- The methods in RoboCup are not yet so far developed that robots do complicated tactical maneuvers. Therefore just watching the ball gives an appropriate knowledge for defending the goal.

The scheme for ball localization with the omnidirectional camera is given in figure 12.

Frame grabber The digital camera was delivered with framegrabber software [Konolige 2001]. This software had to be integrated with RData to produce a module that could be used in a pipeline. The main task here was converting the 24 bit RGB image (8 bit for R, G and B) from the digital camera to an RGB 555 image.

Filtering, blob extraction and selection Once a module was created that delivered an RGB 555 image, the next parts of the pipeline could be literally copied from the stereo ball localization. RGB-set filtering, red blob extraction and the choice of the red blob that represents the ball happen in exactly the same way as in the stereo pipeline.

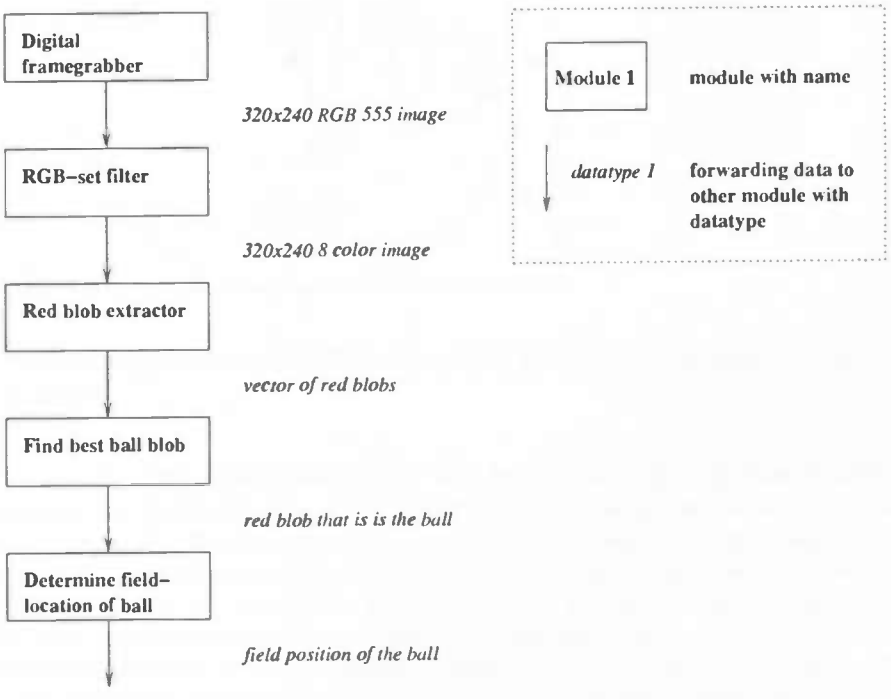

Figure 12: Scheme of the modules for the omnidirectional camera pipeline.

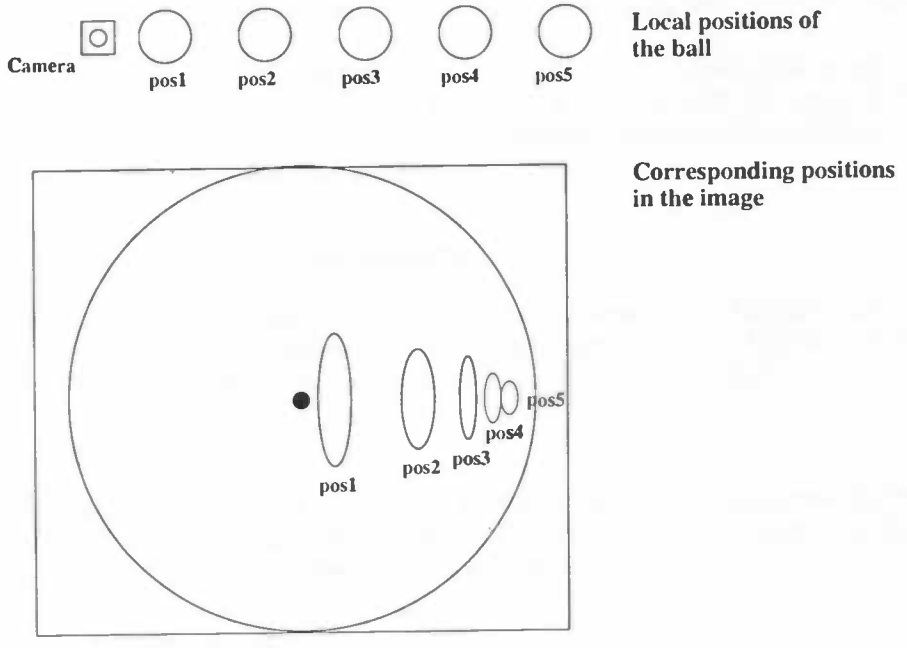

Figure 13: Appearances of different local ball positions in the image of the fish eye camera.

Calculating the local position of the ball The only step that is different from the stereo pipeline is the conversion from image coordinates to local coordinates. For this conversion a simple model is used. The omnidirectional camera is assumed to hang perfectly horizontally and the circular distortion is assumed equal to all directions. This means that the angle from the middle of the image to the image position of the ball is the same as the angle of the local position of the ball. Only the radius is different. This is illustrated in figure 13.

To determine the local position of the ball, it is first calculated in polar coordinates. The angle is the angle from the center of the image to the center of mass of the ball blob. The radius is obtained by multiplying the radius in the image with some calibration function. This calibration function is built by measuring some values experimentally and interpolate for the values in between. Finally the so obtained position in polar coordinates is converted to cartesian coordinates.

4.4.3 Performance

The main noise in the omnidirectional pipeline is that the assumptions that the camera is perfectly horizontal and the distortion perfectly circular do not hold. This introduces an error in the angle to the object of about 5 degrees. The error

in the radius of the ball is about 10% of the measured distance when the ball stays on the ground.

As said before a ball going through the air cannot be localized appropriately with this system. In this case only the angle is correct (with the error of 5 degrees), but the radius has a huge, structural error. However for guiding the stereo camera these values are good enough.

5 Central data structure

The next layer is the central data structure. In this layer all the high level information about the environment is stored, merged and combined. Furthermore a prediction for the future values of some of them is tried to be done.

5.1 RPS

The data processing heart of the goal keeper is the Robot Perceptual Space (RPS). Through this database all high level information passes and its content represents everything the robot knows about its world.

5.1.1 Requirements

The database was completely built from scratch and this gave the possibility to integrate it optimally with the input and input processing layers. It had to meet various requirements.

1. The RPS had to be self updating. This means that its contents are regularly checked, merged and old data is thrown away.
2. The RPS can be accessed by multiple processes. Therefore it had to be built to handle the problems that come along with that.
3. The RPS had to be integrated with RData. It had to be possible to directly connect an input pipeline to it.
4. The RPS is going to be used on all the robots of the team, which all have small differences in the sensors they use. Moreover it is possible that completely new methods are used in the future. Therefore the RPS must have a general structure that allows adding parts and parameters in an easy way.

5.1.2 The general structure

The basic thought of the general structure is that adding data members to the RPS is not done by modifying the object, but by creating a derived class. The base class contains the data members that are used by all computers involving the match. The derived classes become more and more specific for individual robots. The individual RPS for the goal keeper has three derived levels.

1. The Robot Coordinator Perceptual Space (RCoordinatorPS), which is the base class. The coordinator is the coach of the team, a computer outside the field that is connected by radio to all the players. It can have a variety of tasks, like planning the strategy or send start and stop commands to the robots. The coordinator is no robot, so it has no sensors except for the information the players are sending over the network. Therefore it has data members to store this information. It might need the dimensions of the field and some things about the state of the game (score, played time, etc), so there is space for these parameters. Finally there is a module that can check the data members.

2. The first derived class is the Robot Ego Perceptual Space. This class is meant to be the base class for all the players. It has all the data members of the coordinator, because it can use the information from the network too. The big difference is that a robot has a position in the field. Further it has several sensors to be able to perceive its environment. Because this class is a general class for all the robots, it only has members that are useful for all robots. These are the own position, the positions of the teammates, the positions of the opponents and the positions of eventual objects in the field that can not be classified (for example the referee).

3. The next derivation level is the individual level, the Robot Goal keeper Perceptual Space (RGoalPS). There is space for data coming from sensors that are specific for a certain robot. In the case of the goal keeper the specific features are that there are:

- More than one cameras that can localize the ball
- There is a moving camera.

Figure 14 gives an overview of the RPS and its structure and components.

5.1.3 Built for extension

The data members of the RPS are designed for extension. There is counted on the fact that in the future new methods will be used and therefore new features will be added to the data members. For this reason most data members are derived classes of one data member base class and they occur in the RPS as pointers. This allows the adding of extra features to data members without changing the general structure of the RPS.

5.1.4 Dynamic change

The parameters of the data members are changed dynamically. They are all RDataSinks and can therefore be connected easily to the end of an input pipeline. In this way parameters will be changed automatically when some data arrives at the end of the pipeline.

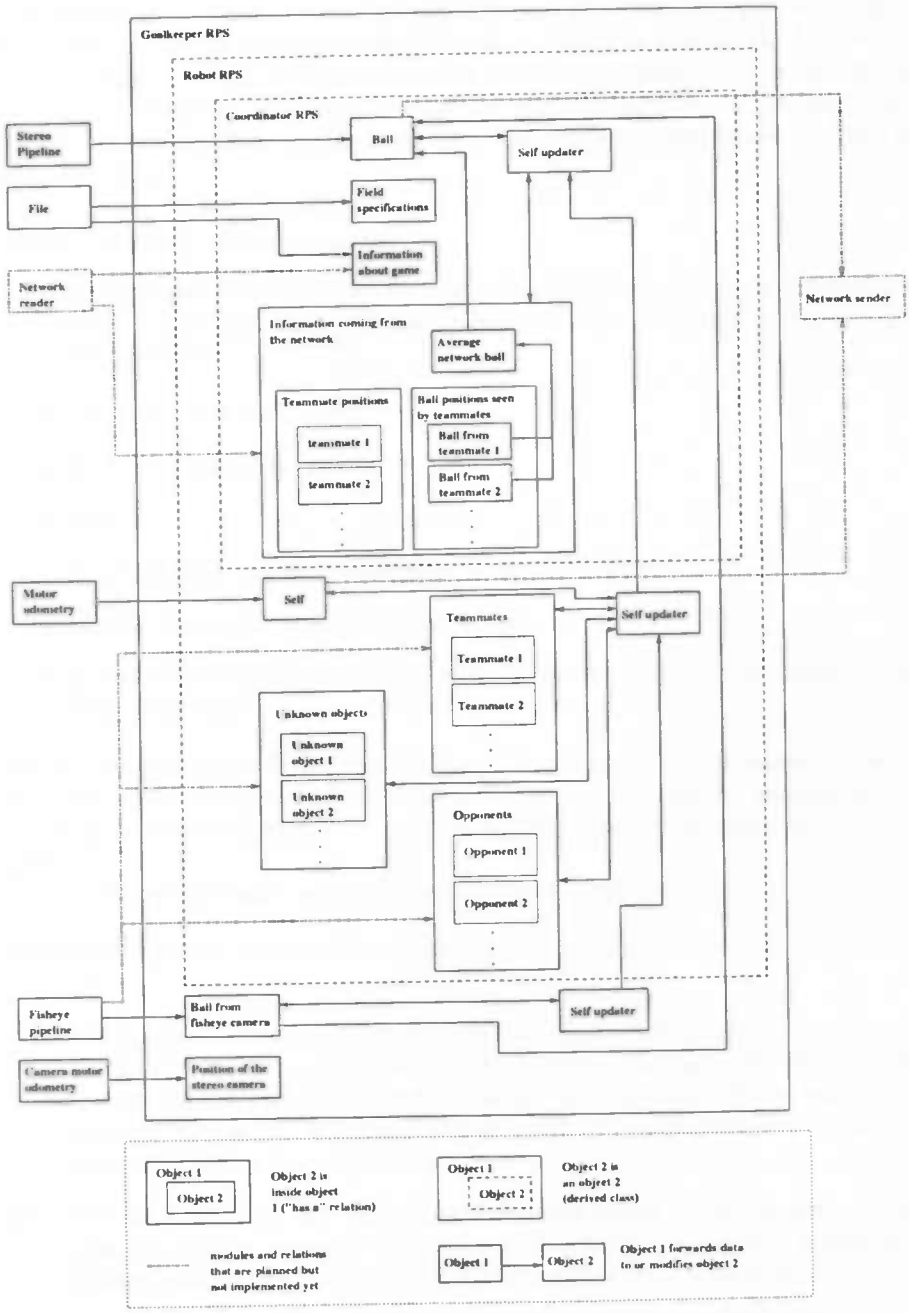

Figure 14: Overview of the RPS.

Because the input pipelines and other methods that use the RPS can be in different threads, a protection is needed for more than one processes working on the same variable at the same time. This protection is a traffic light for threads in every data member. When a thread is modifying a data member, the traffic light is put on red which means all other threads will see it is occupied and wait. This construction allows multiple non synchronized processes to use the RPS at the same time.

5.1.5 Moving objects classes

All data members that store information about moving objects in the field are derived from one base class because they have the same basic features. The main features are:

- Local position and speed
- Global position and speed
- Size
- A model that can do a prediction of the future position
- Source (from which sensor is this object coming)
- a TimeToLive (the time the last observation will still be valid, also see the next subsection)

When new data arrives through the input pipeline, all the parameters will be changed automatically. Additional data that is needed, like the position of the robot to do the local/global conversion, will be taken from other parts of the RPS.

The following classes representing moving objects are used:

RPSBall contains information about the ball. There are no extra data members.

RPSNetworkGoods is a vector to store the information coming from the network. The positions of all the teammates and the ball positions seen by the teammates can be put in this vector. The positions of the teammates are also forwarded to RPSGoods, in which it's combined with sensor information about the teammates.

RPSGoods is a vector that contains information about the teammates. Every position in the vector is a pointer to an RPSGood object, which is a derived class of RPSObject without any added data members.

RPSEvils represents the opponents in a similar way as in RPSGoods.

RPSUnknown represents the objects in the field that are not classified. This happens in a similar way as in RPSGoods.

RPSSelf represents the physical position of the goal keeper itself (or said in another way, the local position of the field) . It is derived from RPSObject with extra data members for the rotation speed, the orientation and there is a variable that says if the robot is lost or not.

5.1.6 Not moving object classes

Apart from the moving object classes, there are some classes for non moving objects.

RPSField contains the dimensions of the field the robot plays on currently.

RPSGamestate can contain all kinds of general information like the network addresses of the teammates, the time to play etc. This object is not fully implemented yet.

5.1.7 Self update

RPS updates itself. This self update process has two functions.

The first one is a control and cleaning function. All the information is put in RPS is actual knowledge. However, when it's not updated, it becomes useless. Therefore every data member of the RPS has a variable called TimeToLive. This variable indicates how long an observation of an object will still be valid. When a data member is updated, the TimeToLive is set to a maximum value. The self update process now takes some time from the TimeToLive every cycle. When no new updates are coming, the TimeToLive becomes smaller then zero and the concerning data member is not valid anymore and will be removed. In this way the RPS is kept clean from outdated information.

The second task of the self update function is to merge data. At this moment this is only done with the ball. The ball can be seen in three different ways and moreover a prediction of its position can be done when it's not seen. These four ways have a clear priority order.

1. The stereo camera gives the most precise information about the ball.
2. The fish eye camera gives less precise information when the ball is in the air.
3. The network information is relatively slow and received information is useless for to base fast interceptions on. Still it is useful to have an idea of where to look at.
4. A prediction assumes that the ball is not touched by other robots and just continues its track, which is very often not the case.

The merging of the ball information is based on these priority rules. If it is possible to apply the first rule, it is used, otherwise the second rule is used etc.

5.1.8 Partial use

The RPS is built as a database that can contain all possible information about the world the robot acts in. However a lot of input methods are not implemented yet. Therefore the RPS is just partially used in the goal keeper. At this moment only the ball positions coming from the cameras, the own position and the dimensions of the field are used and updated in the goal keeper. In the future more should be connected.

5.2 Track prediction

One of the possibilities of the RPS is linking a model to a moving object. When activated, this model is updated every time a new observation arrives. Such a model can calculate the speed of the object and eventually do a prediction of where it is going to.

5.2.1 Linear model theory

Currently only one model is available in RPS. This is a linear model. It assumes that the object moves in a straight line with a constant speed. The model takes a number of positions from the past and calculates the best straight line and speed through them.

The model is a simplified Recursive Least Squares Parabolic Fit [Kimura 1992]. This algorithm was originally designed to calculate the track of an object thrown through the air. It assumes the X, Y and Z direction of the track can be represented as a parabolic equation and the best function is searched through all the data points.

We assume the object moves in the plane of the field (so 2D) and both X and Y directions can be represented as first degree equations.

$$x = p_x + t * v_x \tag{11}$$

$$y = p_y + t * v_y \tag{12}$$

This linear constants can be represented in matrix form as:

$$C_{lin} = \begin{bmatrix} v_x & p_x \\ v_y & p_y \end{bmatrix} \tag{13}$$

When N data points are used, the total square error is defined as:

$$J = \sum_{t=1}^{N} X_e\left[t_i\right]^2 \tag{14}$$

$$= \sum_{t=1}^{N} \left(X\left[t_i\right] - C_{lin} \bullet T\left[t_i\right]\right)^T \left(X\left[t_i\right] - C_{lin} \bullet T\left[t_i\right]\right) \tag{15}$$

$X_e[t]$ = the difference between the real and the model values on time t
$X[t]$ = the matrix with the data points on time t
$T_t = \begin{bmatrix} t \\ 1 \end{bmatrix}$ = the matrix to convert the parameters to a position on time t
With some calculation this gives us for the x direction the next equation:

$$J_x = \sum_{i=1}^{N} x_{t_i}^2 - 2x_{t_i}v_x t_i - 2x_{t_i}p_x + t_i^2 v_x^2 + 2t_i v_x p_x + p_x^2 \tag{16}$$

If we want to minimize the error, we have to equal the derivatives to p_x and v_x to zero.

$$\frac{dJ_x}{dp_x} = -\sum_{i=1}^{N} x_{t_i} + v_x \sum_{i=1}^{N} t_i + \sum_{i=1}^{N} p_x = 0 \tag{17}$$

$$\Leftrightarrow \quad p_x = \frac{\sum_{i=1}^{N} x_{t_i} - v_x \sum_{i=1}^{N} t_i}{N} \tag{18}$$

$$\frac{dJ_x}{dv_x} = -\sum_{i=1}^{N} x_{t_i}t_i + p_x \sum_{i=1}^{N} t_i + v_x \sum_{i=1}^{N} t_i^2 = 0 \tag{19}$$

$$\Leftrightarrow \quad v_x = \frac{\sum_{i=1}^{N} x_{t_i}t_i - p_x \sum_{i=1}^{N} t_i}{\sum_{i=1}^{N} t_i^2} \tag{20}$$

We now have two equations and two unknowns, by combining (18) and (20) we get:

$$v_x = \frac{N\sum_{i=1}^{N} x_{t_i}t_i - \sum_{i=1}^{N} x_{t_i}\sum_{i=1}^{N} t_i}{N\sum_{i=1}^{N} t_i^2 - \left(\sum_{i=1}^{N} t_i\right)^2} \tag{21}$$

$$p_x = \frac{\sum_{i=1}^{N} x_{t_i} - v_y \sum_{i=1}^{N} t_i}{N} \tag{22}$$

p_y and v_y are calculated in a similar way.

These formulas produce the parameters of the linear model as a function of a row of data points. A nice thing of this model is that it takes the same computation time every step. All that has to be done is update the summations and calculate the formulas for the linear parameters again.

5.2.2 Sliding model

The model is used as a sliding model. This means that it's always calculated over the last N data points. This can be done in two ways. Start a new model at every data point and throw the model away that started N data points ago. This means you always have N models. The second possibility is using just one model. In this case a method is needed to remove old data points from the model.

The second possibility is used. The used method starts with building the model from the first data point on. When the model is N data points big, an extra processing step is taken every cycle. This step is removes the first data point from the model. In this way every cycle a new data point is added to the model and an old one removed, so it stays the same size.

The influence of the data points is inside summations. Therefore methods were needed to remove the first points from the used summations.

The time dependent summations always have 0 as their first element and the other elements are time differences from the start of the model. When removing the first element, the second element becomes the start of the model. Therefore all the data points in the model must be converted to time differences to the second data point with aid of the difference between the first and the second data point Δt.

$$\sum_{i=1}^{N} t_i = \sum_{i=2}^{N} (t_i + \Delta t) \quad \Leftrightarrow \quad \sum_{i=2}^{N} t_i = \sum_{i=1}^{N} t_i - (N-1) * \Delta t \tag{23}$$

$$\sum_{i=1}^{N} t_i^2 = \sum_{i=2}^{N} (t_i + \Delta t)^2 = \sum_{i=2}^{N} \left(t_i^2 + 2t_i\Delta t + \Delta t^2\right) \tag{24}$$

$$\Leftrightarrow \quad \sum_{i=2}^{N} t_i^2 = \sum_{i=1}^{N} t_i^2 - (N-1) * \Delta t^2 - \sum_{i=2}^{N} t_i \tag{25}$$

Removal of the first element of the other summations is more simple. For $\sum_{i=2}^{N} x_i$ and $\sum_{i=2}^{N} x_i^2$ the first element can be simply removed by subtracting it from the sum. For $x_i t_i$ the first element in the model is zero again and conversion is done in the following way:

(5.12)

$$\sum_{i=1}^{N} x_i t_i = \sum_{i=2}^{N} (x_i t_i - x_i \Delta t) \tag{26}$$

$$\Leftrightarrow \quad \sum_{i=2}^{N} x_i t_i = \sum_{i=1}^{N} x_i t_i - \Delta t * \sum_{i=2}^{N} x_i \tag{27}$$

The summations with y in it are processed in a similar way as the ones with x in it. This methods use the same computational time every cycle.

5.2.3 Implementation

The whole linear sliding model was implemented using integers where possible. In this way it takes as little computational time as possible.

To be able to remove data points from the summations, the points are stored in a queue until they are removed.

Every cycle a check on the standard deviation is done. When it is too small, the ball is considered not moving. The standard deviation is done by using

the square mean/mean square method because the needed values are already available from the model parameters.

5.2.4 Performance

The model gives a reasonable prediction of the direction and speed of an object. However, the performance is relative to the number of points that is used.

When a lot of points are used, the prediction is very stable because the noise is filtered out by averaging. The disadvantage of many points is that changes of direction are adapted very slowly by the network and possibly even too late to react.

On the other hand a short model is very sensitive to noise. It gives a prediction that is less precise, both in direction and speed. However it has the advantage that it reacts very fast to changes of direction of the object.

In the goal keeper a model of 10 data points is used. This gives a prediction that has a reasonable precision together with a fast enough reaction speed.

The calculation of the model takes so little computational time (about .01 ms) that it can be easily connected with all the moving objects without slowing anything down.

6 Decision module

Now that all the information collecting, processing and storing methods are connected, the next and last step is using this information to make a decision about what to do. This task can be divided in three subtasks. The first one is to decide where to go, the second if the kickers need to be used and finally where the stereo camera should move.

6.1 Where to go

The first decision that needs to be taken is what the optimal position is to prevent the opponents from scoring.

The fastest direction to move for the goal keeper is forward and backwards because it can drive in this directions without having to rotate first. Therefore the goal keeper is put in the goal with the sides parallel to the goal. In this way the two sides of the goal can be reached in the fastest way.

The basic movement of the goal keeper goes over a base line at some distance in front of the goal (figure 15). Originally the plan was to make the trajectory a little curved at the edges to block a ball that is coming from the sides optimally. This was abandoned by a lack of time.

The task of this module is to calculate the optimal interception point for the ball. Other players are not taken into consideration because at the moment they are unseen. In order to calculate the optimal interception point, the situations that can occur in the field are divided into three groups.

1. Unknown ball position In this case the goal keeper stays at its current position. It is very well possible that the ball is just temporary out of sight. However, when the ball is not seen for some time, the goal keeper goes back to the middle of the base line, the base position (figure 15). This is the optimal position to defend the goal when nothing is known about the ball. The returning to this position is handled by a planning and control algorithm built by our team members Alessandro Farinelli [Farinelli 2001] and Alessandro Castelpietro [Castaldo 2001]. This algorithm moves the robot to a given position and orientation while avoiding possible obstacles (which can't be seen at the moment).

2. The ball is far away In this case the optimal interception point is the crossing of the base line and the line through the ball position and a point centered behind the goal (figure 15). Far away means further away than a certain predefined distance. This point simulates the position of a real goal keeper that makes the corner of the goal small. It follows the movement of the opponent with the ball without leaving another part of the goal completely unprotected. The moving to the calculated position is handled by the same algorithm as mentioned in the last paragraph.

3. The ball is close When the ball is close, the track prediction of the ball is used. The interception point is the crossing of the ball track and the base line. This is equivalent with standing in the track of the ball in order to stop it. The robot never passes the edges of the base line. Because the ball is so close, the goal keeper has to move as fast as possible. Therefore the control mechanism of Alessandro Farinelli and Alessandro Castelpietro is abandoned here and the robot just moves backward or forward, its fastest way to move. However the price here for speed is that no repositioning is done and the robot can easily get off the base line during fast actions. This error is corrected when the ball is on a save distance from the goal again and the robots is in one of the first two situations again.

This behavior scheme is very simplistic and needs to be refined further in the future.

6.2 Using the kickers

The next decision that needs to be taken is whether the kickers should be used or not. This decision is a direct response function. Some area is defined around the goal keeper where the kickers can reach the ball. When the ball is inside this area, the kickers are extended.

The idea behind this behavior is that when the ball is so close to the goal the goal keeper can reach it, there is direct danger for a goal and the ball needs to be kicked away as fast as possible.

Also this behavior is in fact a little to simplistic. There are situations possible where it is better to wait for a moment and kick the ball in a more intelligent way. For example in a direction without opponents or to wait with kicking in

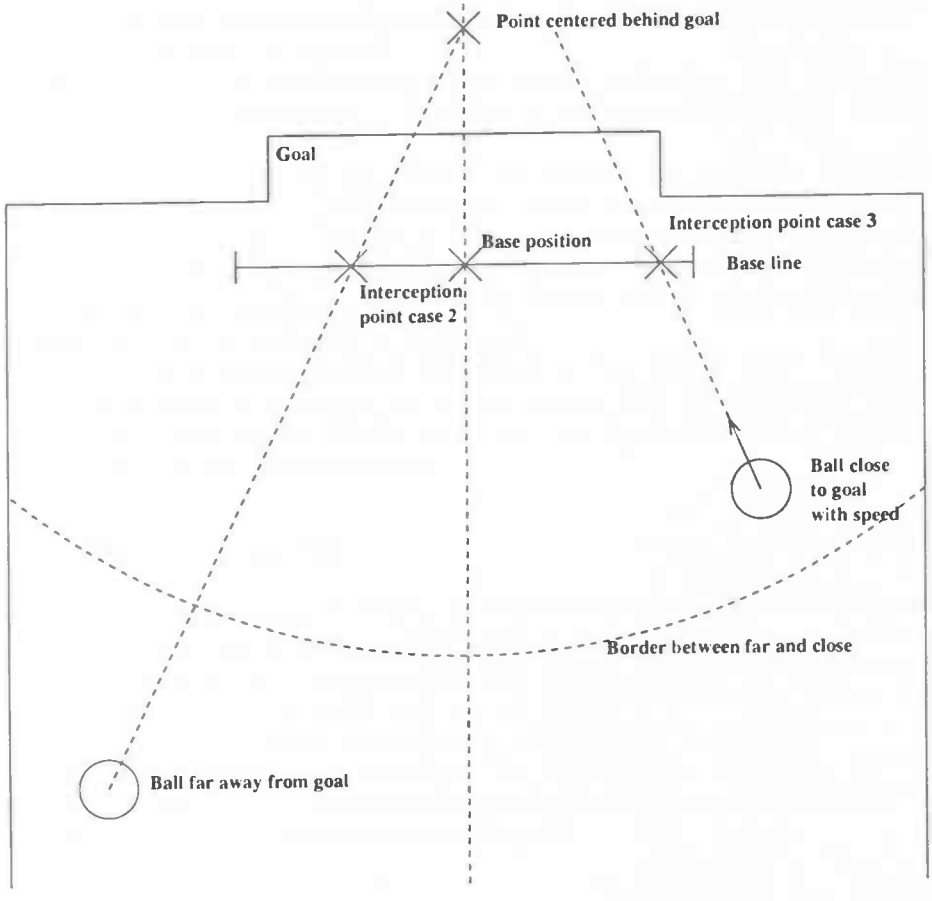

Figure 15: The interception points

order to save pressured air when the ball is stuck. So in the future also this algorithm needs to be refined more.

6.3 Moving the stereo camera

The last decision that needs to be taken is where to move the stereo camera. The best behavior here would be to follow the tracked object smoothly. When moving the camera with the same speed as the object, the object has no rotational speed with reference to the camera and a lot of noise and synchronization problems by moving are avoided.

However, also the final moving stereo camera and motor were ready just shortly before the tournament. Therefore he smooth behavior is not implemented yet.

As a temporary solution the camera just moves to the angle the ball is on top speed. To avoid too much movement, which causes noise, there is a no move area in the middle of the image. When the ball is in this area, the camera doesn't move. When the ball is outside this area, the camera moves till the ball is in the center of the image. This position is inside the no move area and the camera can stay motionless for a while again.

This methods makes a part of the images useless. Images taken while the camera is rotating on top speed, are useless because they are to smeared. However these images are not filtered away and the temporary noise in the ball position is taken for granted for now.

7 The real world

Subparts that are working in theory, in a simulation or even in real world tests are one thing, all the subparts cooperating in a working robot is another. Even when a system is built systematically from scratch and all the subparts are tested, there is still very much that can go wrong. In a complex system like a robot there are so many dependencies and factors of importance that it is almost impossible to keep track of them all. Therefore the real test for a robot is the hard reality. Can Kerberos do its task in the circumstances of a RoboCup match and defend the goal in a reasonable way?

7.1 The match

After months of work and a final sleepless night, the moment of truth appeared, the first real RoboCup match. To my great disappointment and embarrassment the on board computer crashed after several seconds and didn't move for the rest of the match. This was a very bad start but from now on it could only get better. During the time between the next matches we worked almost constantly in order to get Kerberos working and later to get it working better. This resulted in a reasonable behavior by the end of the qualifying rounds. Some nice saves where made. Unfortunately there where also numerous problems with the rest

of the team, which resulted often in a goal keeper against the complete other team situation. This resulted in 43 goals against us in five matches and not going to the next round. Nevertheless it was a very instructive and useful experience. Now the most important practical problems of Kerberos are known. This knowledge can be used in the future to improve its behavior and show what it's worth in future tournaments. Therefore an overview will be given of the good and bad things about the system, together with the accounted problems.

Wireless network The reason the robot crashed the first matches were problems with the wireless network. When a match starts, for some reason the wireless network gets almost completely stuck immediately. When some process is giving feedback (for example via cout), which arrives via the network at the remote login window, it risks to crash because the feedback can't be sent appropriately over the network. This problem was solved by sending absolutely no feedback over the remote login connection.

Hardware A normal PC is not built for the rather violent collisions that can occur in RoboCup. At one moment apparently one of the frame grabber cards was moved in its socket and one of the cameras gave no more image. Most collisions where against the wall, due to bad localization. This means that with better localization a big part of the risk can be taken away. Also better mounting of the cards in the slots can reduce this risk.

Another problem was that the wheels didn't have enough grip on the ground to do a fast acceleration. This can be solved by using wheels with better grip.

The last problem was that the stereo camera motor sometimes returned the wrong angle.

Apart from these three points the hardware was very reliable and no other big problems occurred.

Localization The already mentioned slipping of the wheels caused huge errors in the localization by odometry. This method was definitely not appropriate and a better localization method is needed absolutely.

Omnidirectional camera The main problem of the omnidirectional camera was image noise. When the ball is more than about three meters away, the noise is bigger than the ball in the image. This means that the ball can only be detected when it is closer than three meters. This reduces the time the goal keeper has to act substantially.

Stereo camera There was a lot of image noise in the stereo images too. Because the orange of the ball and the magenta of the robots where mapped on partially overlapping RGB values, it was impossible to distinguish between them.

The noise introduced by the movement of the camera wasn't a big problem. The ball doesn't move very fast around the field and stays usually in the no-move zone of the stereo image, which meant the camera didn't have to move very often.

Reaction speed The limiting factors in the reaction speed where limited view of the omnidirectional camera and slipping of the wheels. Because of the limited view, the ball was often seen too late and hereby the tactical positioning started too late. As a result Kerberos had to move very fast. However because of the wheel slipping very fast acceleration was not possible. This would only result in a lot of wheel slipping before the robot starts moving.

Reasoning The reasoning algorithm worked well for the situations taken into account. The goal keeper responded in the right direction, only too slow sometimes, as said before.

There where two situations for which no appropriate response was programmed for. The first one was a player trying to push the ball behind the goal keeper between the goal keeper and the goal. This can be solved by a replacement of the base line by the initial idea, in order to leave no room between the back border and the goal keeper. The second situation is a ball that is behind the goal keeper, but not yet in the goal. For now the robot just does nothing, which worked well, but is not a very elegant solution. Some algorithm to take the ball away should be designed for this situation.

For the kicking algorithm, the moment of kicking turned out to be very important. When the ball is kicked too early or too late, it looses much of its effect and power. There should also come an algorithm that can determine if the ball is stuck (for example between the goal keeper and an opponent). In this case there should be no kicking in order to save air pressure.

7.2 Future work

Because so many subsystems are incomplete or not working well enough, in this part an overview is given for future work and modifications. The overview is given in order of importance.

7.2.1 Localization

The most important part that's missing is a good localization module. The goal keeper moves fast and always acts close to the border, which will result in many collisions without good localization. Moreover the tactical decisions will not be effective if the robot isn't in the position it thinks it is.

7.2.2 Hardware

Secondly, there are some hardware parts that need to be replaced or modified. The wheels need better grip, for a faster interception. The motor controller of

the stereo camera needs to be looked at because it seemed that sometimes the wrong angle was returned. Finally the cameras should be replaced by ones with a better image quality in order to classify the RoboCup main colors better.

7.2.3 Decision algorithms

The decision algorithms are built with a lot of time pressure in a very short period. Basically they work, but they are far from perfect. Therefore they will need attention in the future.

7.2.4 Recognizing more than the ball

At this moment the only moving object in the field recognized is the ball. The positions of other robots can be very useful for the defense of the goal too, specially when opponents start to attack smarter.

7.2.5 Wireless network

At this moment the wireless network is just used for remote login to the robots. In the future it should be used for information exchange between the robots and the coordinator. This should be done in a client-server way to avoid crashing the robot because of network problems.

7.2.6 Object recognition

The object recognition by color that is used now is a fast but primitive method. In the future methods should be used that can not only classify, but also recognize individual objects. Here the work of our teammember Davide Troiani can be used. He did research on methods to recognize the ball by correlation methods [Troiani 2001].

7.2.7 Intelligence

The next step will be to put more intelligence in the goal keeper. As the opponents get smarter and smarter, the defending tactics need to be more and more refined. Here can be thought about more cooperation with teammates, for example not just kicking the ball away, but to a teammate or work together with a defender.

7.3 Conclusion

The goal of developing a working RoboCup goal keeper was reached. Kerberos proved to be able to defend the goal in a reasonable way. The overall system is built in a clear and robust way, ready for future expansions. The subsystems can be divided in two groups.

The first group are those that where made with plenty of time and planning. They are optimally integrated in the overall structure and there functionality is well underbuilt theoretically.

Unfortunately there is also a second group. This are the sub systems that where developed on the last moment. Because of the time shortage, they are often just ad hoc solutions, without much theoretical background and built to work in the simplest way. In the future they should be replaced with more scientific methods.

When leaving the ad hoc parts away, there still is a large well structured basis left. This is a robust foundation and framework for future development, that will still go on for the next few years.

In fact the goal keeper will never be ready. There will always be parts that could be better, faster or smarter. But this is the challenge that makes RoboCup so interesting.

References

[Andrew et al. 1999] R. Andrew Hicks Ruzena Bajcsy, 'Reflective surfaces as computational sensors' , *GRASP Laboratory Department of Computer and Information science University of Pennsylvania, 1999*

[Baldassari 2001] Daniele Baldassari, 'A modular software modeling system for mobile robots', master thesis (in Italian) *Computer Science Department University of Rome 'La Sapienza'*, 2001

[Cappelli 2001] Flavio Cappelli, 'A real time motor controller for a mobile robot', master thesis (in Italian), *Computer Science Department University of Rome 'La Sapienza'*, 2001.

[Castaldo 2001] Alessandro Castaldo, 'Techniche di generazione automatica di piani eseguibili robot mobili', master thesis (in Italian) *Computer Science Department University of Rome 'La Sapienza'*, 2001.

[Farinelli 2001] Alessandro Farinelli, 'Trajectories Planning In Dynamic Environment', master thesis (in Italian) *Computer Science Department University of Rome 'La Sapienza'*, 2001.

[Grisetti 2001] Giorgio Grisetti, 'Localizzazione Robusta ed Affidabile per Robot Autonomi in Ambiente Dinamico', master thesis (in Italian) *Computer Science Department University of Rome 'La Sapienza'*, 2001.

[Iocchi et al. 2000] Luca Iocchi and Danielle Nardi, 'Self localization in the RoboCup Environment', *Dipartimento di Informatica e Sistemica Universita di Roma 'La Sapienza'*, 2000.

[Iocchi et al. 2001] Luca Iocchi, Daniele Baldassari, Flavio Cappelli Alessandro Farinelli, Giorgio Grisetti, Floris Maathuis and Daniele Nardi, '*S.P.Q.R.* Wheeled Team', Springer-Verlag, 2001

[Kimura 1992] H. Kimura, 'Adaptive Visual Tracking Algorithms for 3-D Robotic Catching', M.S. Thesis, *Department of Mechanical Engineering, Massachusetts Institute of Technology*, Camebridge MA, August 1992.

[Kitano et al. 1998] H. Kitano and M. Asada, 'RoboCup Humanoid Challenge' in: *Proc of International Conference on Intelligent Robotics and Systems* (IROS-98), Victoria, 1998.

[Konolige et al. 1999] Kurt Konolige and David Beymer, 'Small Vision System Users Manual', SRI International, 1999.

[Konolige 2001] Kurt Konolige, 'DCAM Digital Camera Capture Software Users Manual', Videre Design, 2001.

[Nardi et al. 1999] D. Nardi and others, 'ART-99: Azzura Robot Team', in: *RoboCup-99: Robot Soccer World Cup III*, pages 695-698, Springer-Verlag, 1999

[Steels 1994] L. Steels, 'The artificial life roots of artificial intelligence', *Artificial Life Journal*, Vol 1,1. MIT Press, Cambridge.

[Troiani 2001] Davide Troiani, 'Stereo ball localization using correlation', master thesis (in Italian) *Computer Science Department University of Rome 'La Sapienza'*, 2001.