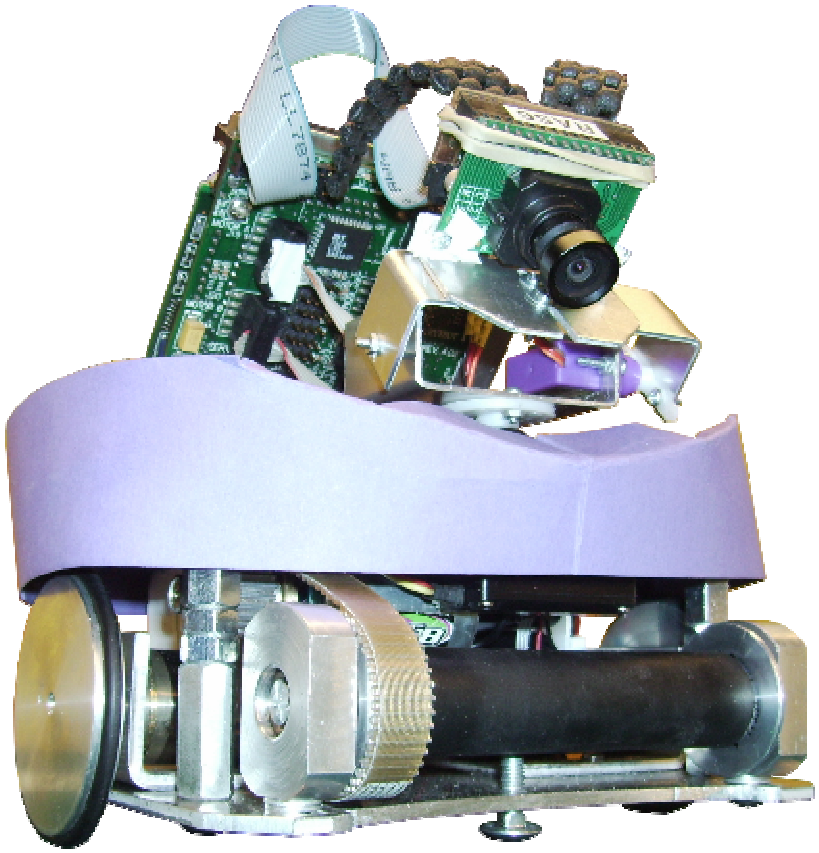
Royal Institute of Technology
2D1426 Robotics and Autonomous Systems

R2D2 DS Edition
Project Report



Spring 2007

Christian Illanes, illanes@kth.se
Fernando Porta, fernando@kth.se
Stephan Puls, puls.stephan@gmx.de
Sylvain Rodhain, sylvainrodhain@hotmail.com

Abstract

This report describes the robot R2D2 Dark Side Edition, which was build as part of the course 2D1426, Robotics and Autonomous Systems, in spring 2007. The rules of the competition and the provided hardware will be introduced. Our ideas will be presented in detail as well as what actually was implemented. After all we draw a summary and conclusion.

Contents

1 Introduction

1.1 Project description

This project aimed at giving us a sufficient approach of the conception of an entirely automatic system. This aim was embodied by the creation of an independent embedded robot, supposed to “play a soccer game” (the specific rules applied will be considered later on).
To reach those ambitious objectives, we were dispensed lectures on general robotic points of interest, a seminar by the leader of an industrial project, and supported by Anders Dovervik. We were also provided with several elements “ready to use” for creating our robot.

The project was articulated around 3 main events:

- Lab0 was meant to help us get involved in the project, having done by then a system able to respond to any stimulus, and modify its activity.
- Lab1 was meant to test we were already running a robot able to score alone on the field, in a systematic way.
- The final competition was eventually to test the performance of the different teams against each other.

The project was also probably to give us the experience of a team work, and test our management of a multi-disciplinary field through the use of the competences of several participants.

1.2 Rules and settings

The entire set of rules can be found at:
http://www.csc.kth.se/utbildning/kth/kurser/DD2426/rules/

There is not a lot to say about the rules themselves. Though, we consider that the fact the corners were added to the previous settlement, and removed slightly before the competition, even if this has not been of any trouble for us, was unfair. The rules were also looking quite undefined, with several modifications to come later than at the beginning of the course.

1.3 Development environment

The programming was done on a workstation provided in the lab room for the course. Matlab was used for calculations regarding Look-up tables for the image processing. The robot programming was done in C and compiled with a gcc compiler specific to the given EyeBot controller. For documentation, internal communication and file archiving a

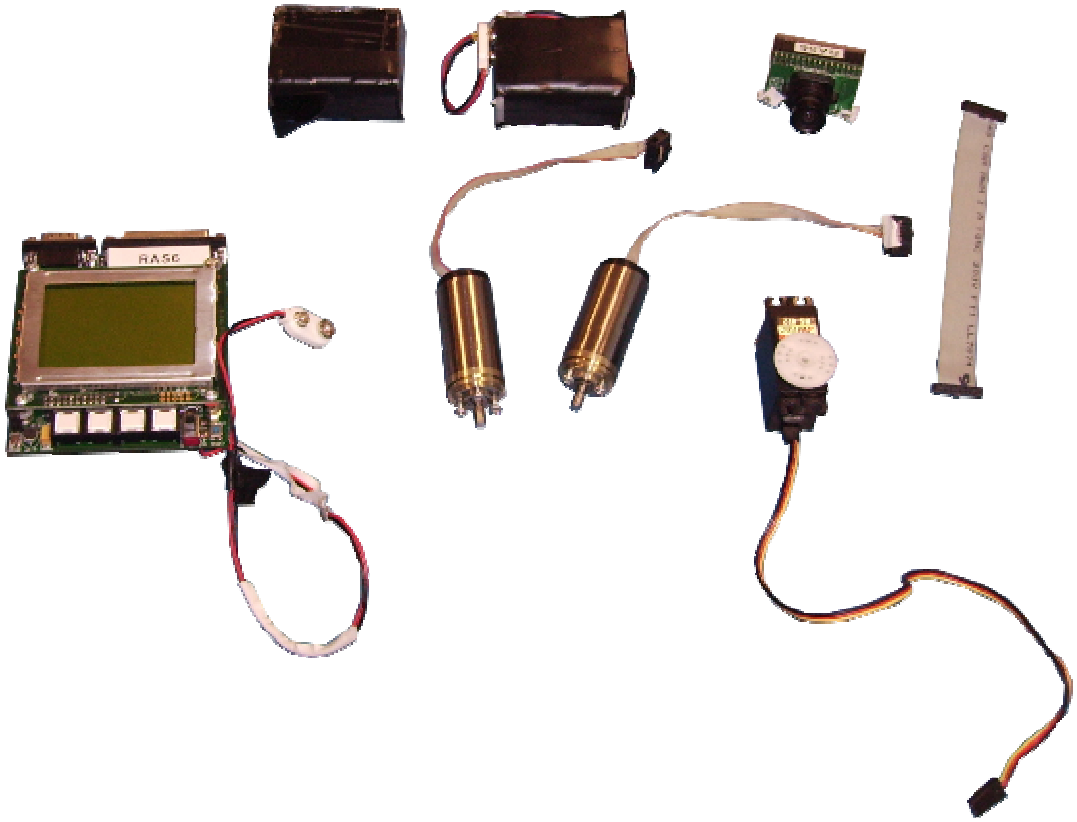
private Wiki was used. A CVS repository was used at the beginning, but was later abandoned.

2 Hardware

One part of this course was to physically build the robot with all its mechanical parts. This chapter is going to mention what material was given from the beginning and the parts that where created during the course. Also an explanation will be given to mention our thoughts and ideas about the decisions taken with respect to the construction choused.

2.1 Given parts

For this course all groups were given the same material to start with. Besides building material such as aluminium sheets, screws, nuts and other parts for assembly, there were motors and wheels, a servo, camera, battery packs and of course the EyeBot as the processing unit.

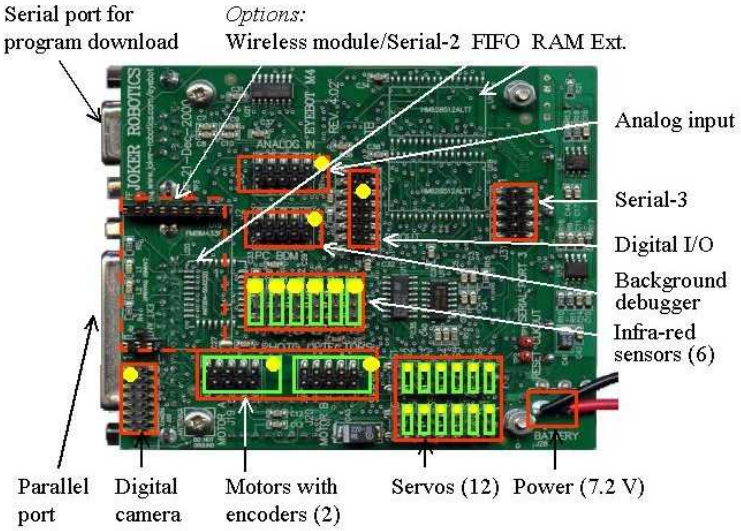


2 battery packs, 1camera, 1 EyeBot, 2 mini motors, 1 servo, 1 camera cable

Given parts:

- Aluminium sheets (never used)
- 2 motors (Faulhauber/minimotor 2224006SR)
- 2 wheels (one broken)
- 1 camera (color camera, 176x144 pixels)
- 1 microcontroller board – EyeBot (defect)
- 2 battery pack
- 1 servo (Hitech HS422)

The EyeBot has a 35MHz, 32-bit Motorola Microcontroller and runs an operating system called RoBIOS. Furthermore the board has a large graphics LCD (128x64 pixels), 2MB RAM, digital camera interface and digital and analog I/O pins. The programming was done in C.

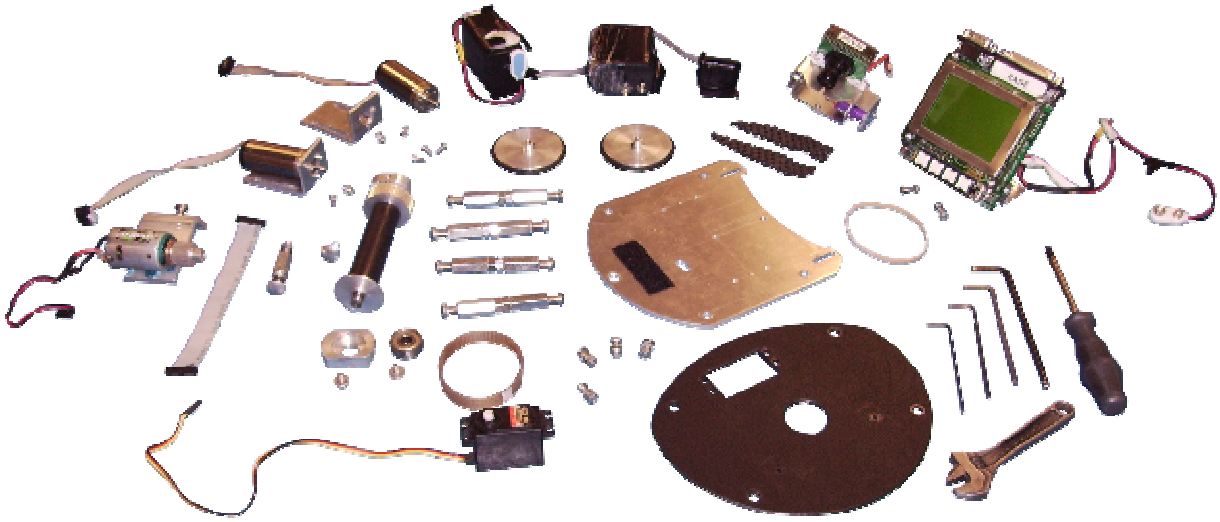


The back of the EyeBot controller

2.2 Construction of the robot

Almost every mechanical part of the robot was constructed with our own materials. The only allowable colors were those colors that were not in conflict with the ball, the goal or the field.

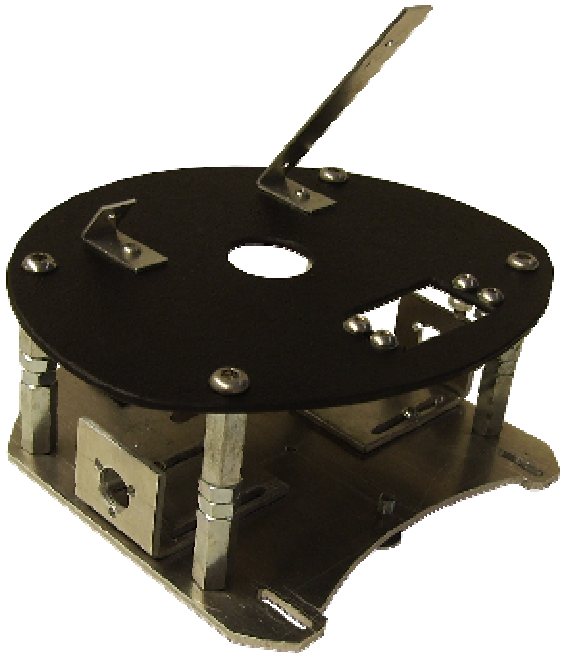
To work up the whole robot we had access to a professional workshop at Hjälpmedelcentralen syd, which we are very thankful for.



The entire robot dissembled and the tools needed for assemble it

2.2.1 Chassis

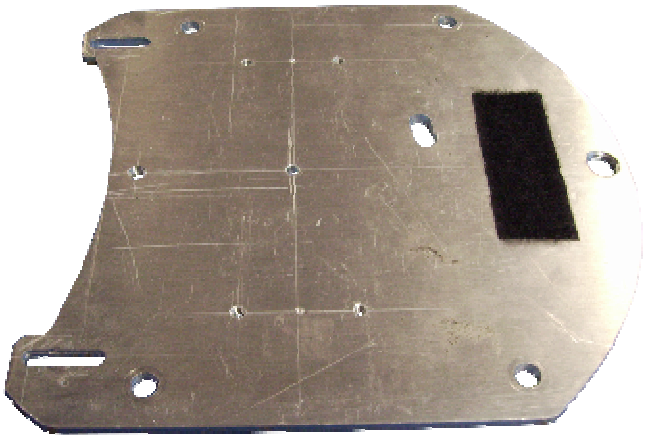
The main idea of the shape was to make it as circular as possible with the wheels along the center line so the robot could turn on the spot without getting stuck into the wall or some other obstacles. For getting static stability we added two extra contact points, one static in the front and one in the back with suspension. Two extra contact points was needed because of the symmetry. We chose to have the suspension in the rear part because we wanted to have the roller at the same level as the ball as often as possible, and for the case when it was not at the same level it is preferably to have it a little above.



Chassis

2.2.1.1 Bottom plate

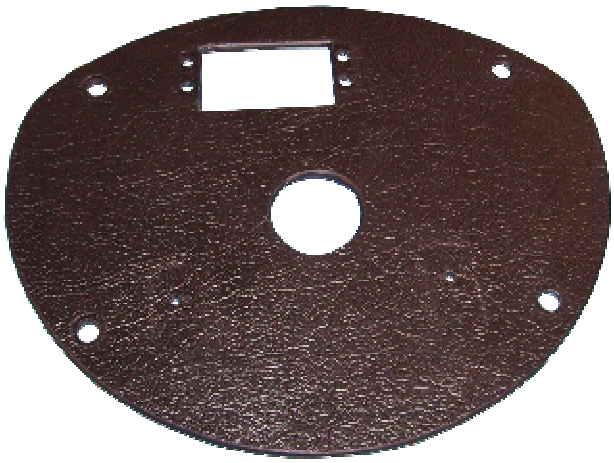
For the bottom plate we chose a 3 mm thick aluminium plate for better stability instead of the 1 mm plate given (see chapter 2.1 given parts).



Bottom plate

2.2.1.2 Top plate

This plate was constructed in abs plastic because of the simplicity to work with it. We did not know a priori what we where going to add to it.



Top plate

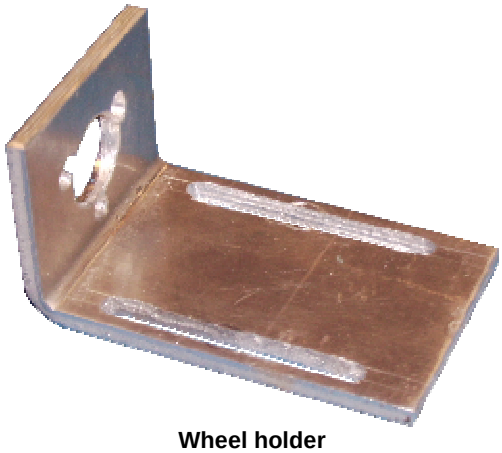
2.2.1.3 Distance pillars

To make the height of the top plate easy adjustable we used distances, screws and nuts. This gave us the possibility to test different robot configurations, such as having the camera at different heights.

Distance pillar

2.2.1.4 Wheel holders

As we had to create our own wheel holders we created them in such a way that they can be moved sideways to adjust the wheels so they have the same distance from the center point. It is important to have this configuration accurate so the turning of the robot also is accurate. We wanted to have the chassis as close as possible to the field without having any screws besides the contact points brushing against the field. This wheel holders was made of 3 mm thick aluminium plates.



Wheel holder

2.2.2 Suspension

To not lose contact with the ground, as we had more than 3 contact points, we used suspension on the fourth one situated at the back. The material used was two springs, one on each side of the bottom plate. The cylindrical alike construction is to hold the suspension vertical and is mounted on the bottom plate.

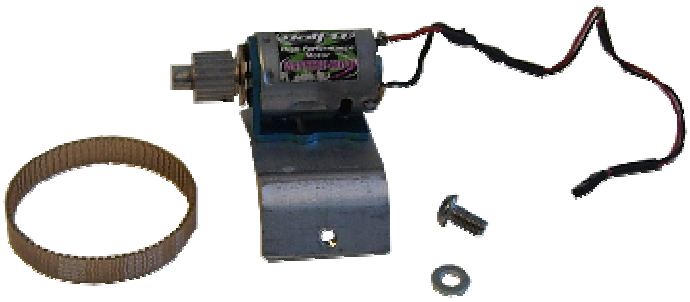
Suspension

2.2.3 Roller

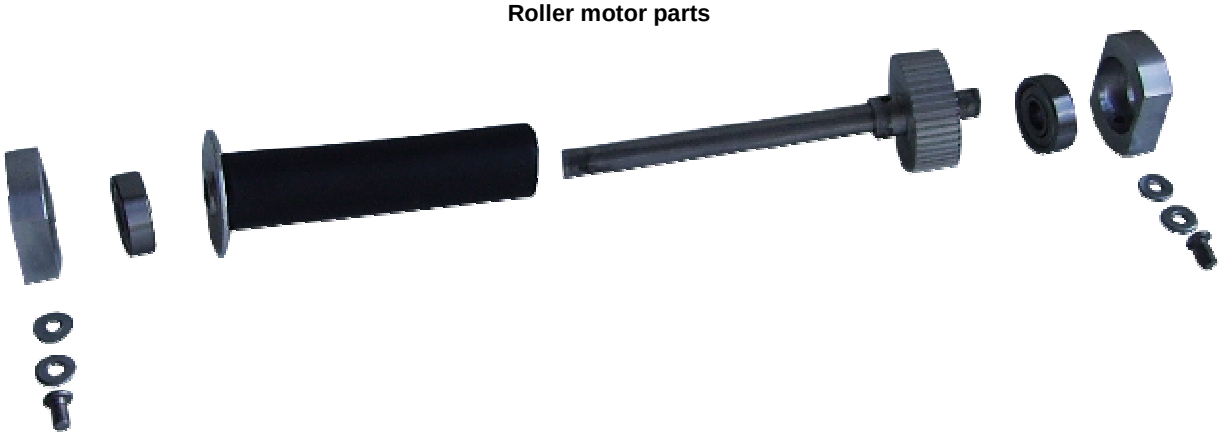
Many ideas on how the robot should work were based on the assumption that we had a mechanism that was able to hold the ball near to the robot chassis while the robot was driving around in the field. The requirements were that it should be possible to drive forward, backward and turn without loosing the ball. It should also be able to get the ball from a corner without any major efforts. One of those mechanisms that could do all that was a good working roller with good qualities. But this was not an easy task as it seemed at a first glance.

We spent many days in finding materials with good friction against the ball. Besides the friction, we wanted a material with so little texture as possible to avoid the bumpiness. We found out that a silicon tube was the best solution. Unfortunately the tube was blue and resulted in a color conflict with one of the field goals. After a lot of searching, we finally found that a bike wheel rubber tube was satisfactory enough, but not equally good as the silicon tube.

Another issue with the roller was the transmission part. Even here we did try out different types of constructions. The final version uses bearings to minimize the friction of the rotation and a transmission belt with gear wheels.



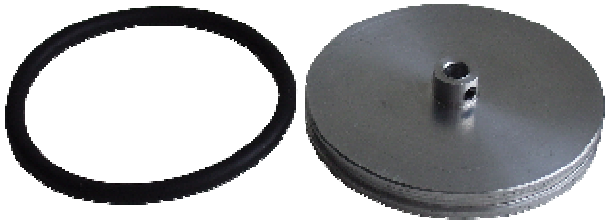
Roller motor parts



Roller parts

2.2.4 Wheels

We made our own wheels when we discovered that one of the given wheels was broken. Those ones where made a bit different with the help of a turn, but we kept the diameter dimension. One improvement of this construction compared to the given one is that it is easier to take it off and on. According to the rules, the wheels must have O-rings and where therefore constructed this way.



O-ring **Wheel**

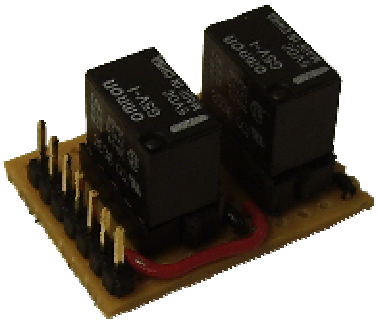
2.2.5 Roller controller

We wanted to be able to turn on the roller only when the ball was near the robot. For this we needed to have a form of motor controller circuit that responded to signals from the EyeBot controller. A simple relay switch circuit was built for enable the roller to turn on and off the backspin from the EyeBot.

Experimenting with the roller we found out that by changing the direction of the roller the ball was kicked fast over the field bouncing between the walls, as we had a really good roller construction. This gave us the idea of having the kicking function embedded into the roller. For this a bidirectional motor control circuit had to be build.

A transistor-based H-bridge circuit was tested but did not work as good as expected as it took power from the roller motor, making it slower.

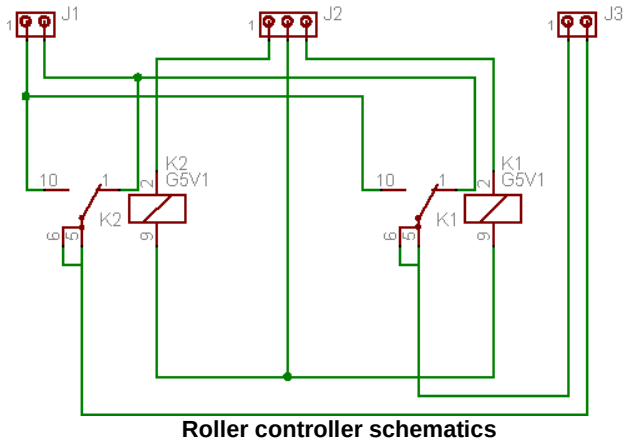
The final bidirectional motor control circuit was a simple 2 relay circuit. This solution can not control the speed of the motor, but it was not a problem for us as we wanted it to spin as fast as possible.



Roller controller

2.2.5.1 Schematics

The controller takes to signals from the EyeBot that controls the relays switching between positive and negative DC power.



Roller controller schematics

Part list:

- J1 - DC Power
    1. Positive
    2. Negative
- J2 - EyeBot Signals
    1. Signal A
    2. Signal GND
    3. Signal B
- J3 - Motor connector
- K1 and K2, Relay, G5V1 (http://www.relay100.com/upfile/20066231172433421.pdf)

2.2.5.2 Roller programming

To send signals to the motor controller we inserted signal cable to the first ports on the digital input and output port on the EyeBot (see EyeBot picture in chapter 2.1). The EyeBot library[1] contains the functions to change and read the digital i/o port:

```
OSWriteOutLatch(int latchnr, BYTE mask, BYTE value);
```

and

```
OSReadOutLatch(int latchnr);
```

The signals were defined as follows:

A	B	Output
0	0	Off
0	1	Dribbler
1	0	Kicker
1	1	Not defined (Off)

So for example for turning the dribbler on the function call would look like:

```
OSWriteOutLatch(0, 0xFC, 0x01);
```

For turning the kicker on:

```
OSWriteOutLatch(0, 0xFC, 0x02);
```

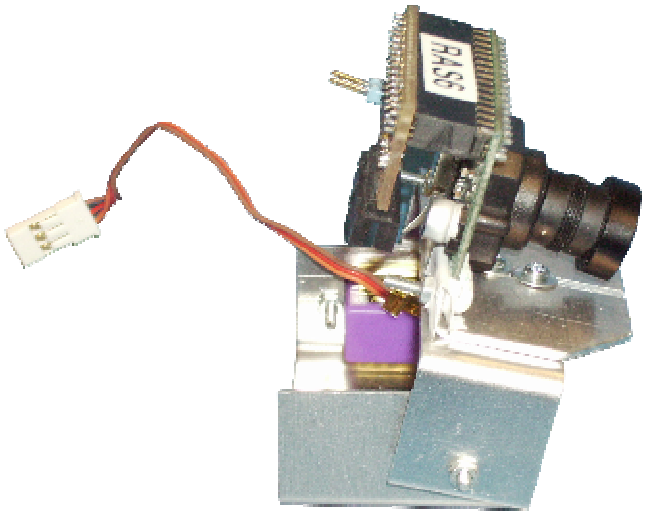
And for turning the roller off:

```
OSWriteOutLatch(0, 0xFC, 0x00);
```

[1] EyeBot - Online Documentation, http://robotics.ee.uwa.edu.au/eyebot/doc/API/library.html

2.2.6 Camera holder

As we wanted to be able to pan and tilt, a special holder for the camera was constructed. The cam holder requires two servos. One was given, Hitech HS422, and the other one was bought by us, BlueBird BMS306. The holder is made of 1 mm aluminium plate and consists of 3 modules. The first one is for panning and is attached to the given Hitech servo that in its turn is attached to the top plate. The second module is for tilting and is controlled by our BlueBird servo. This servo is mounted in the panning module. The third module is just the camera holder.



Camera holder with panning and tilting function

2.2.7 Ears

The ears might seem not to have any function, but it is wrong. It gives the robot a nice personality and makes it funny, which is good for sales reasons. But most important of all is that it acts as a cushion if the robot hits the wall when looking down.

2.2.8 Optical odometry

From the start we decided that we wanted to see if we could use an optical mouse for getting a more accurate pose than using VW-drive. Using an optical mouse we hoped to minimize the risks of dead reckoning. The optical mouse connection was tested but we did not have time to implement it on the robot because we had problems with our EyeBot.

For making the optical mouse to work, we decided to have an extra 8-bit microcontroller, Microchips PIC16F84 (http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010230), with a serial interface giving the EyeBot the actual pose. The PIC was chosen since we already had one from a previous course.

The mouse that we tested was a Sweex Optical Mini USB Mouse with a USB to PS/2 adapter. Using the adapter to connect to the PIC worked with our mouse, but as noted on a web-page (The seeing-eye mouse, http://imakeprojects.com/projects/seeing-eye-mouse/) it dose not work with all USB mice.

The PS/2 mouse has a data and a clock signal and sends packages of 3 bytes for sending X and Y movement data. Mice with scrolling wheels can send an additional byte for the Z movement. A good wiki page (Computer-Engineering, http://www.computer-engineering.org) was found, explaining the PS/2 mouse protocol and interface. We used an assembler example program from an article about communication with a PS/2 mouse using a PIC16F84 (Communication with a PS/2 mouse using a PIC 16F84 - http://www.fiacopetti.it/ps2_16f84_en.htm) and a test program in c by the same author of the wiki page.

The tests were carried out having the mouse moving on the field connected to the PIC circuit communicating serially with a laptop instead of the EyeBot. Having the mouse moving slow we got good values but some times we got totally wrong movement values. Also when moving the mouse faster (robot speed) we got wrong and sometimes getting the contrary values of the mouse movement. This could have been caused by the field fabric pattern making it difficult for the optical sensor to se changes. Maybe having a better mouse the problems could have been solved. But having problem making the EyeBot work, this part had a lower priority.

3 Implementation

3.1 RoBIOS

RoBIOS is the operating system running on the EyeBot controller board. It enables the programmer to use the camera, the motors and servos with ease and allows easy downloading of own programs to the robot. To achieve this usability the OS incorporates a programming interface which provides means to make photos or controll the motors. There are functions to print images to the LCD screen, it supports multitasking and ways to work with all sorts of sensors. There is also a VW-Drive implemented in order to controll the driving through the motors.

3.2 VW-Drive

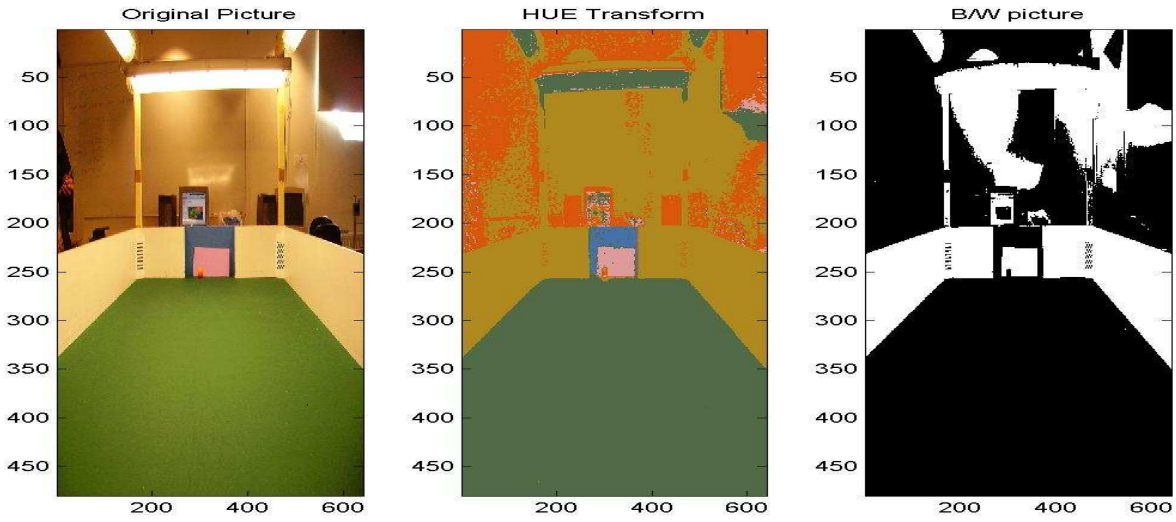
The VW-Drive gives the programmer the means to concentrate on the interesting things of robot programming. In order to stear the robot one can use functions like VWDriveStraight or VWDriveTurn. As arguments one uses speed and distances. After all this is a fairly comfortable way to get the robot moving. But there were also problems with it. We could use only a rate to gather pictures of 3 frames per second while driving, because the VW-Drive needs quite a bit of processing time on the EyeBot and therefor slows down other systems. At least one other group could not use the VW-Drive at all, because they got only scrambled pictures while driving.

3.3 Image processing

3.3.1 Visual recognition system

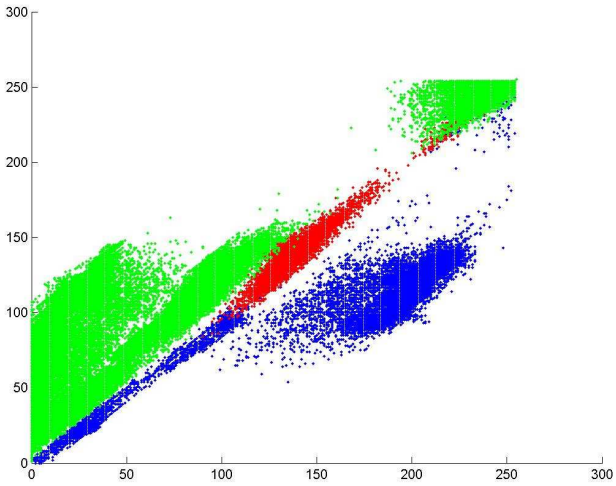
One main problem the robot has to deal with is the recognition of the elements in its field of vision. Those elements, thanks to the limited environment are at the number of 7 (carpet, ball, wall, enemy, blue and yellow goal, corner signs removed later on), and they were chosen so that their colors would be easily differentiable (respectively green, orange, white, purple, blue, yellow, black).

The obvious way to distinguish the elements was to use the colors. Therefore, the first approach we had has been to take several pictures of the field (with a regular 5 million pixels camera first), and try to compute the HUE values for the different elements. The results were not bad, but we didn't use the real camera mounted on the Eyebot, nor the entire set of colors, as we just wanted to have a first approach. A black-and-white computation gave us a quite good recognition for the corners.



Hue transformation and segmentation of a 5 million pixel picture

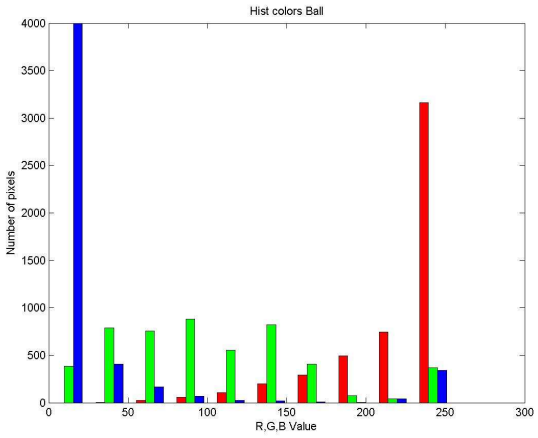
A second approach tried in parallel was to observe the data in 2D projections of the R.G.B. space. The separation between colored areas, and hence objects was made using hyper-planes perpendicular to one of the three axis (Red, Green or Blue), i.e. simple lines in the 2D projection plane. The results were, once again, not so bad, but there was a particular lack of recognition in some areas between the ball and the yellow goal, and we thought the recognition was made to focus mainly on the ball in a first time.



Projection of the colors of the carpet (green), enemy (red) and blue goal (blue)

The third and final approach was implemented on the robot because of its facility to use: a look-up table, in which was stored a defined value (from 1 to 6 for each element), computed on a personal laptop was integrated to the code, as a constant table. Therefore the operations required on the Eyebot were limited to the lecture of a value stored in memory, while any change could be made and uploaded easily in parallel.

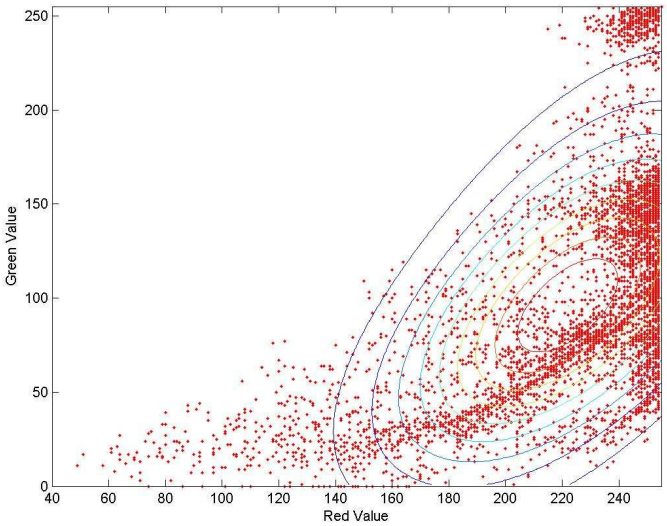
The first way to create the LUT has been to use the same system as previously, defining hyper-plane of the R.G.B. space, and storing the values in a 3D matrix. Though, the results could be improved, so we decided to use other algorithms for computing the values. The simple Gaussians were giving really bad results, and after observing the distributions of the data points, one could notice the distributions were similar to multi-Gaussian distributions.



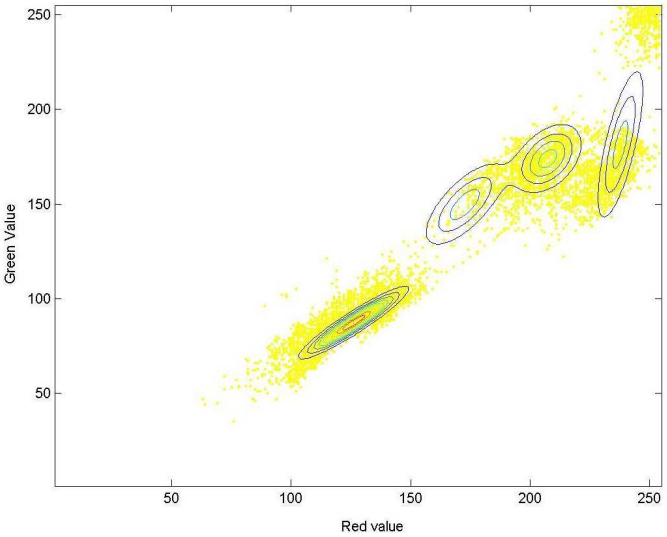
Histogram of the pixel's R.G.B. intensities used for learning the color's distribution of the ball

The calculation of the Gaussian could have been done using the expectation-maximization algorithm, but a much simple way was to define manually the limits of the bins, used later on to build the Gaussians, making the learning much faster. Then,

making the hypothesis that 3D bins are uncorrelated, we simply define the probability density as the sum of the 3 dimensional weighted with priors Gaussians. We store then for each possible color in the LUT the number of the element with the maximal probability (using again priors on the probability of each object).



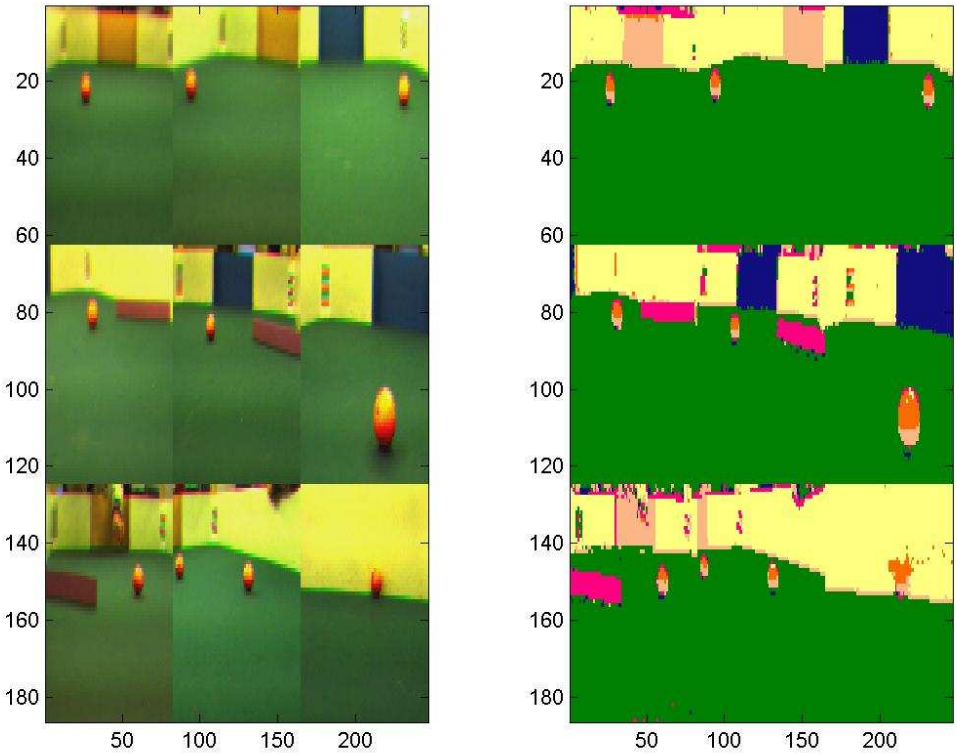
Projective representation of the R.G.B space of the colors of the ball and its multi-Gaussian model



Projective representation of the R.G.B space of the colors of the yellow goal and its multi-Gaussian model

Then the initial system was set, and as soon as we could take pictures with the camera of the Eyebot, we could train the system with real images. The first LUT was big (255*255*255 bytes); we thus reduced it to a smaller size, picked then as 25*25*25 bytes, and considered only the values of the LUT at the intersections of the 3D grid defined every 11 units of intensity for each color. For every pixel in the picture taken by the camera, we were considering the inferior rounding value, and looking for its corresponding element in the LUT. Afterwards, but unfortunately too late, we had the

idea to use a LUT of size 32*32*32 bytes, so that for every pixel we would get from the camera, the R.G.B. value obtained being a number between 0 and 255, we could get the stored value in the LUT by deleting the 3 last binary digits. For example, when considering the red intensity 205, in the case of a LUT divided in 32 equivalent spaces, each space contains 8 units of intensity, and therefore, 205 is stored in the matrix as 25. It turns out that for 205, written in binary 11001101, the 5 first digits are equal to 11001 that make 25. This trick would save quite some time probably.



Results of the recognition process, to a set of images taken with the Eyebot Camera.

The results were quite satisfying, even if bottom half of the ball was always represented as yellow goal. This was for sure annoying, but could not be solved as there was, from the strict point of view of the colors, absolutely no difference between the two elements (the shadow of the ball itself was modifying the perceived color). It was then just a matter of, choosing the bottom half to look like the ball, hence taking the risk to have the yellow goal to be considered as a ball, or the other way around. As the top half of the ball was looking fine, we preferred to be able to detect the yellow goal properly, and add simple system counting the number of pixels to decide whether what we perceived as yellow was or not the yellow goal.

An important problem we had to face when considering the creation of the LUT was that the entire environment would be different on the day of the competition, so the whole system was made to be automatic, with MATLAB-almost-friendly-interface so that a regular user (in case the programmer would not be available on the day of the competition) would be able to use the scripts independently on his computer. The script

is, from a set of complete pictures, allowing you to cut separately the different parts of the images, storing their color values, creating then the R.G.B. histograms, and letting the user define the bins. It is then computing the Gaussian Mixture representation, and filling in the LUT. Eventually, it is writing the LUT in C code, and saving the file for an easy transfer on the robot. This script has been used on the day of the competition, providing in 20 minutes, the cutting-off of approximately 20 images, and the computing of the LUT.

We consider the set of images taken during the competition, around 20, was too small to give a sufficient training, and the Gaussian models obtained were probably deficient. That could explain the strange behavior of the robot, which went toward the wall at the beginning of the seeding process.

3.3.2 Estimation of ball distance

The problem here was quite simple: being given a picture, and selecting a pixel in it, can we determine how far the robot is from that point? This was also a main part in our vision system, as we put a lot of effort in the building of a system allowing tilting and panning for the camera.

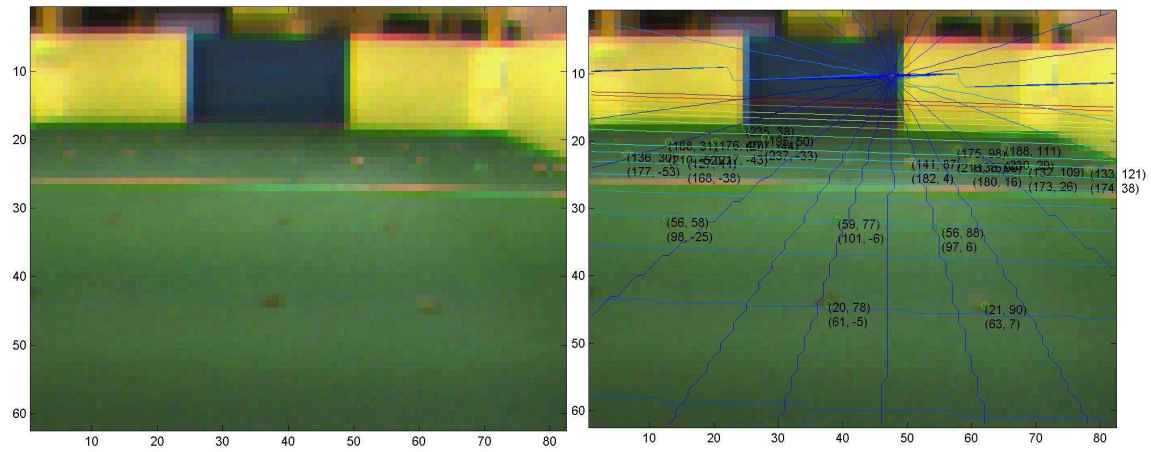
The first approach has been to make a simple homography between the plane of the field and the plane of projection in the camera. This was done by solving an over-determined system, relating the homogenous coordinates of a point in the field, to its image in the camera:

$$\begin{bmatrix} X & Y & Z \end{bmatrix}' = C * \begin{bmatrix} x & y & 1 \end{bmatrix}'$$

Where x, y, represent the coordinates of the pixel, and X, Y, Z the homogenous coordinates in the Euclidian Space. We could then obtain the real position in the field, by normalizing:

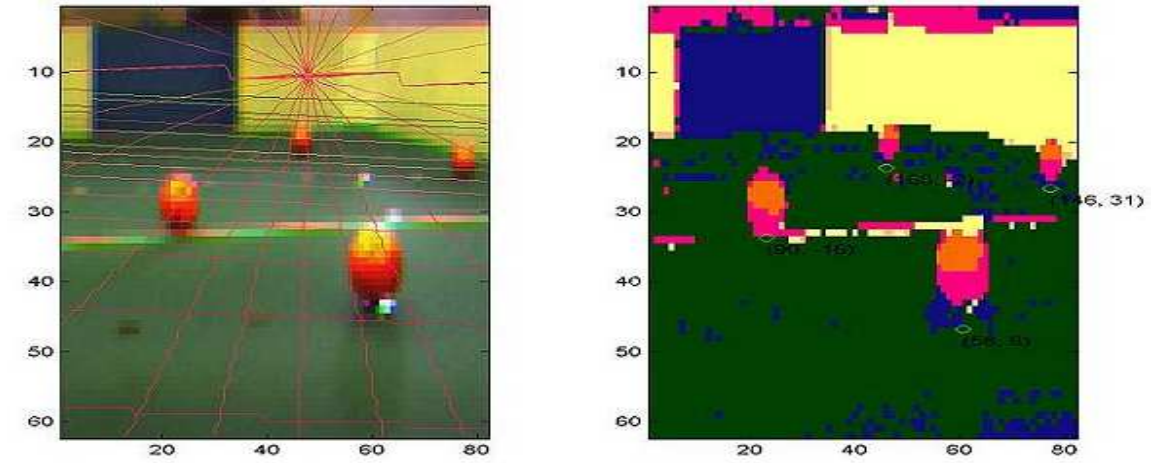
$$X_{REAL} = X/Z$$
$$Y_{REAL} = Y/Z$$

We had for solving this system a simple picture of the field, with pieces of tape dispatched all over the table, in a determined way:



Homography estimation of the distance ahead, and to the sides

And we could then estimate the positions of some balls:



Example of the calculation of balls' positions

This system was working with a simple position of the camera, but when trying to extend it to the tilting and panning functions, it turned out that we were facing another problem. For panning, there was no difficulty, as a rotation of the camera could be, in the frame of the camera, be estimated as the opposite rotation of the field.

Therefore, we could consider the new equation:

$$\begin{bmatrix} X & Y & Z \end{bmatrix}' = R(\theta) * C * \begin{bmatrix} x & y & 1 \end{bmatrix}'$$

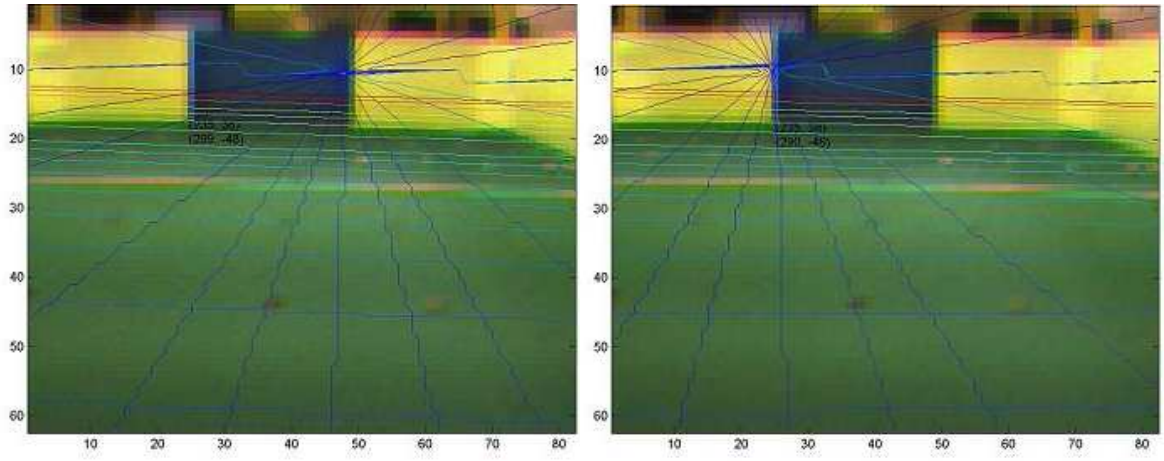
Instead of

$$C * \begin{bmatrix} x & y & 1 \end{bmatrix}' = R(-\theta) * \begin{bmatrix} X & Y & Z \end{bmatrix}'$$

With:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And θ is the angle of rotation of the camera.



Camera oriented forward to the robot **Artificial rotation of the robot to the left**

Though, we could not solve the tilting of the camera, as the homography was not working…
This is, I think, due to the fact that the real projective formula is:

$$[x \quad y \quad 1]' \sim C' * R_Z(\theta_Z) * R_X(\theta_X) * [X \quad Y \quad Z \quad 1]'$$

Where C' is what I roughly thought the inverse of C.

Unfortunately, we can calculate C' but we cannot then compute its inverse C, as C' is not a square matrix, but can compute the pseudo inverse $pinv(C)$.
Though, when we would need:

$$C * C' = I$$

We have, and the matrices don't commutate:

$$C' * pinv(C) = I$$

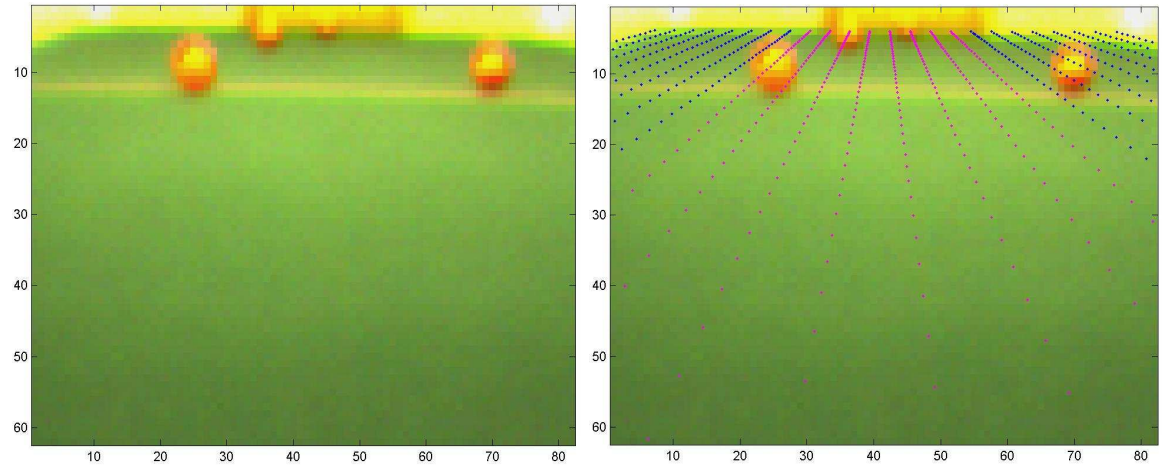
So we were blocked with a system that could, given a point on the field, calculate its image in the camera, but not from a pixel in the camera, give the distance to the robot.
We then had the idea to use specific orientations of the camera (i.e. a set of specifically defined angles for both tilting and panning), and calculate all the possible positions, which would be stored in a LUT.
For every pixel in the camera, the system would access the LUT, and consider the distance to the robot. Thus, the system requires one LUT for each couple of angles for panning and tilting, (θ_Z, θ_X). We were using small windows frames, but still, we required:

$$N_{Number\ \ of\ \ couples} * X_{resolution} * Y_{resolution} * 2 \sim N_{Number\ \ of\ \ couples} * 5kb$$

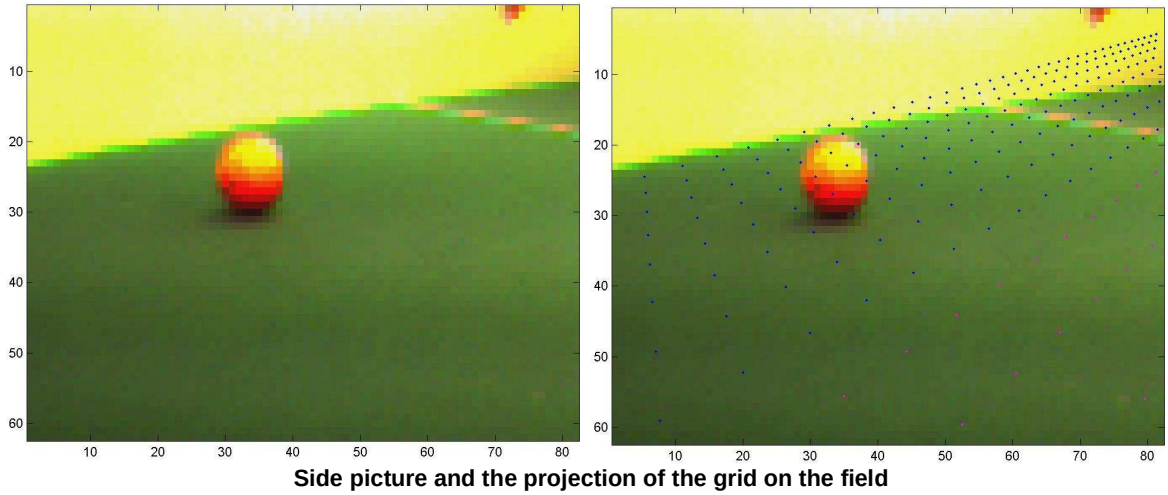
In addition, we decided to include the angle toward the considered pixel, comparatively to the orientation of the robot. So we solved the over-determined system with homogenous coordinates in both the camera frame, and the Euclidian Space, to

compute the C' matrix, and then picked 3 different angles for the panning, and 2 for the tilting, i.e. a total of 6 couples of angles, that we could reduce to 4 for symmetry reasons.

We then applied a grid to the field, with a 5 cm precision, and computed the resulting image in the camera. We started saving the largest values (i.e. those in the very top of the image), and then those closer, as we preferred to have an under-estimation of the distance rather than an over-estimation: pixels that are chosen again by another point of the grid store the latest value that was projected on them.



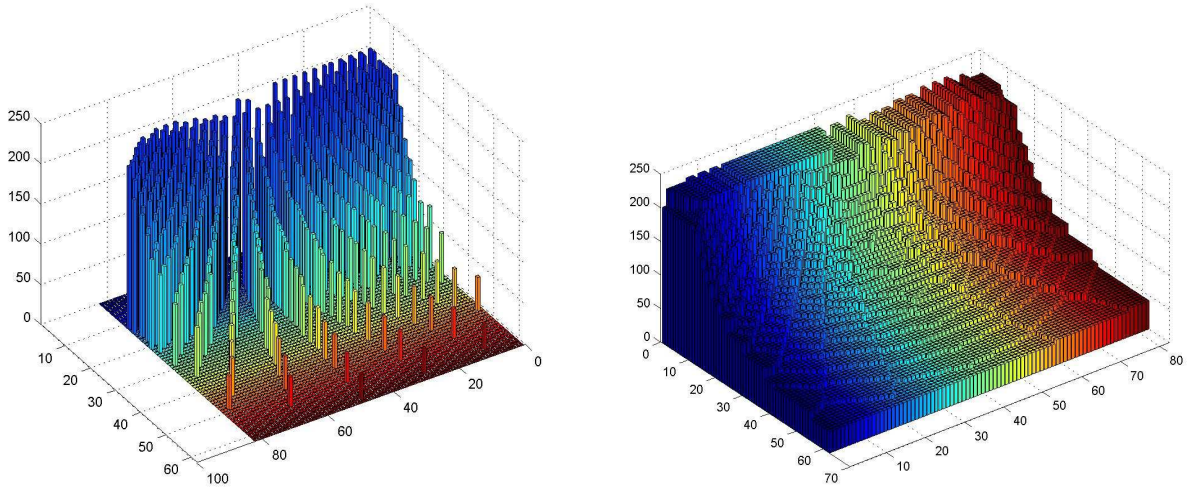
Frontal picture and the projection of the grid on the field



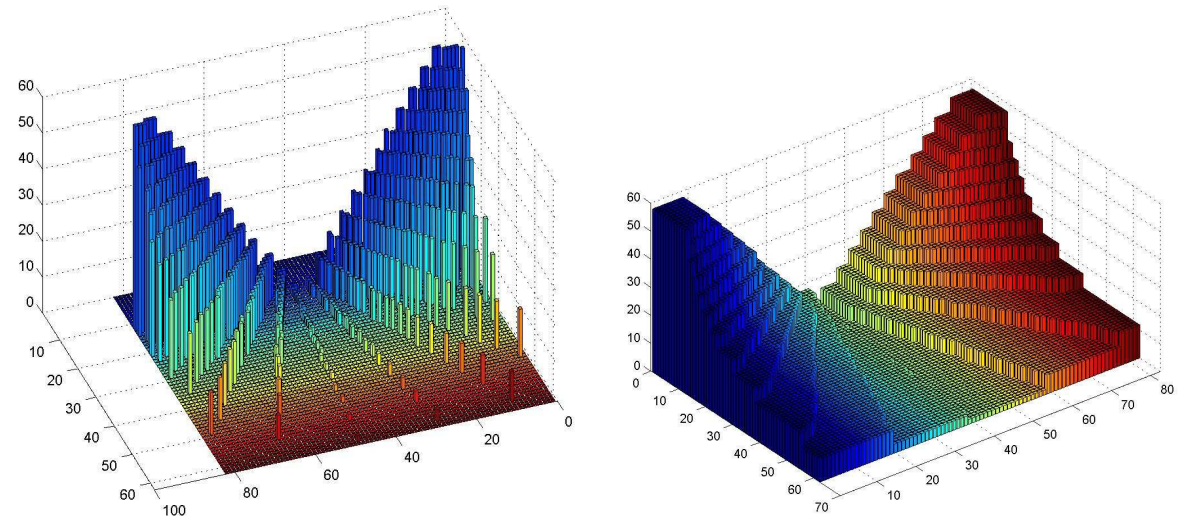
Side picture and the projection of the grid on the field

One can see there is a noticeable error on the side estimation of the distance. This is, I think, due to the fact that the translation of the camera center compared to the rotational axis Z it not taken into account. We unfortunately did not have enough time to solve this matter.

Another problem we had was that not all pixels were being “hit” by a point in the 3D grid, therefore, our LUT was having empty values, that we had to fill in; after several attempts to use interpolation, and extrapolation functions, we opted for a simple nearest neighbor method, as in no case we could pretend to reach a higher precision than 5 cm (nor needed it) considering the errors on the calculation of the C’ matrix.



Estimation of the distance ahead, depending on the pixel coordinates



Estimation of the distance to the sides (absolute value), depending on the pixel coordinates

The interest of this system was, as for the color recognition, that everything was being compiled separately, and exported on the Eyebot in C language. The system had absolutely no calculation to do, and if we had had more time to improve this system, we could have probably reached a system with 1 cm precision (or reaching the limits of the camera precision at least, very precise in the foreground, less in the background).

Another advantage is as we did not know, when this system was made, if we were to use global localization or not, the system can be used in both cases, as it is giving a relative position of the ball to the robot, what could be extended to absolute position. We did not have the time to exploit the possibilities of the positioning-LUT for the competition.

This system is also not limited to the ball, and one could think of using the vertical edges between a goal and the wall, or between the carpet and the wall, to make a triangulation, to improve the estimation of the position of our own robot (a Hough Transform is however definitely not something to think of).

3.4 Worldmodel

We did not implement any global positioning system due to the lack of time, but it was intended to have one so we could estimate and update the robot pose for a better planning. We could estimate our absolute pose by using our method for estimation of object distances as mentioned in chapter 3.3.2 Estimation of ball distance.

3.5 Navigation

Two trajectory algorithms were tested, driving patterns with Bezier curves and open loop trajectory control with circles and straight lines. Both this systems was tested in Matlab and in the EyeBot simulator.

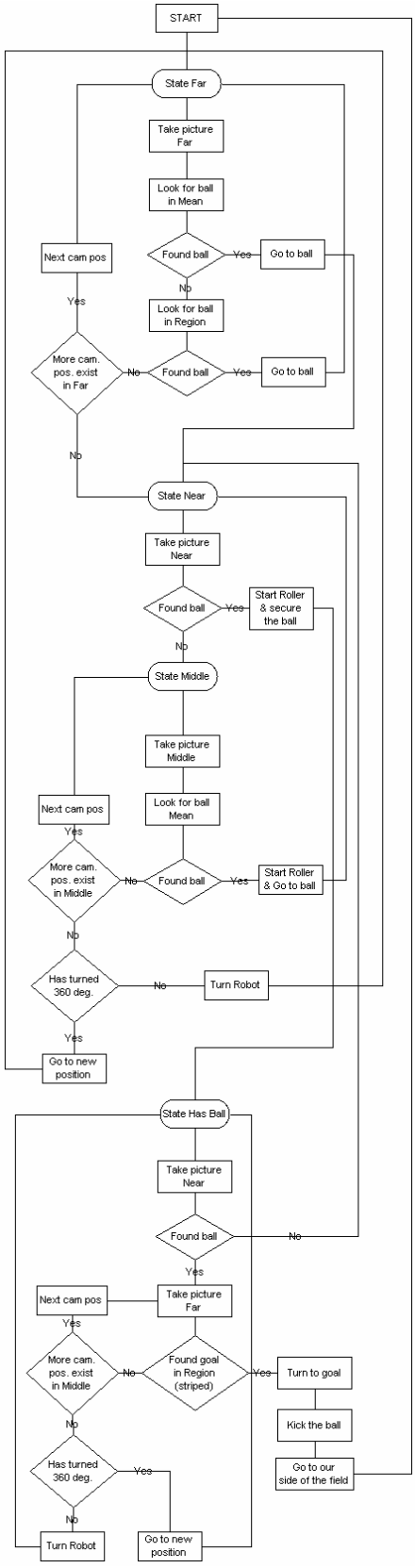
Having a state of the art roller we discarded both systems and used a even more simple algorithm, turning on spot and drive straight to the ball respectively goal.

With simplicity comes robustness.

3.6 Statemachine

Our statemachine have 3 main parts, locating the ball, locating the goal and score a goal. Having the statmachine outlined simplified the development of the code and the debugging of the program.

Searching for the ball is our hardest task since we had to control the pan and tilt servos for the camera. To optimize the searching two different algorithms were used. The so called Region algorithm is the slower one but gives more accurate values when looking for chosen object, the ball in our case, far away from the robot. Things that are near are easier to find and could therefore use a faster algorithm that we call Mean.



A detailed statemachine

4 Results

Our group has been efficient at the beginning of the construction. We had the opportunity to access an additional workshop, with professional tools. We quickly obtained a robot that was driving forward perfectly and turning on itself without deviations.

We left open-room for special features, and then focused on the construction of camera device. Out of different sources from internet, and after several deficient realizations, we picked a rather simple system, but working perfectly fine. It turned out this system was biased by the cable linking the Eyebot to the camera. This was solved later on by creating our own cable for the camera, longer and more flexible.

At that stage, we reached the lab 0, which has been quite simple to pass, as all the mechanical work had already been done. We made the robot to respond the simple stimuli of pressing buttons and change from a straight forward trajectory to a rotation on the spot. Some time before lab 0, the recognition system was defined, and almost working. In addition, early before the lab 0, we changed the original wheels because one of the wheels was defect and could not be attached to the motor axis. With the old ones we had to face a deviation when it came to rotation on the spot.

At this point we had discovered that our EyeBot controller did not work as it should. Beside having loose EyeBot parts at the beginning (the LCD screen was not attached to the rest of the EyeBot). The EyeBot got frozen in the middle of the running program. Then the EyeBot was not able to start, forcing us to flash it with the BDM cable booting it from DOS. The problem continued the whole course and slowed down our programming development. This was bad because there were no spare EyeBot. We may have located the problem to the ROM memory as it often gets frozen during the saving of a program to the ROM memory.

The lab 1 has been a difficult step, because of the lack of time to face the requirements. The mechanical parts of the robot were all working fine by then, but we had no strategy for scoring a goal. The choice was to make a dummy robot, with no control loop, that would allow trajectory correction. Therefore, we were hypothetically able to score once, but extremely sensitive to rotation on the spot errors, and odometry errors.

On the way to the competition, we did our last improvement considering mechanical matters. We added an electronic device allowing us to use the roller also as a kicker, with amazingly efficient results, though this idea was discovered by mistake/experimenting. We also improved our strategies and decided for a specific state machine; eventually the system allowing the estimation of the distances was finished, and the final coding was done.

During the seeding part of the competition the robot made a nice score by a kick three seconds to late, even though the EyeBot managed to stall and the LUT was not

optimized for the lightning in the competition room. Our late goal did not classified us for the rest of the competition :´(

5 Conclusion

During our programming phase we had quite a bit of trouble with the EyeBot. We had to flash it the hard way many times, with a parallel port BDM cable and with a special bootable CD, as well as reflashing it by just easy downloading the RoBIOS to the EyeBot. We experienced sometimes, that the EyeBot screwed up our programs by writing randomly in the memory. This way it is often hard to debug own code, if the error might be caused through the hardware itself.

Some things were not implemented on the robot, for example the usage of the distance-estimation-LUT could have increased the speed of movement of the EyeBot and maybe also given us a absolute pose system.

After all our system was kept simple in its layout. It was a good choice to go with the statemachine, it made the programming fairly modular and therefore made debugging easier.

The camera made the project not easier. The auto-brightness 'feature' made the image processing quite interesting. To much time went in trying to correct the inconsequent images to for example distinguishing the yellow goal and the ball. Maybe a better control board for the camera could have been used. The AVRcam board (http://www.jrobot.net/) make it a lot easer to work with image recognition without losing any theory.

The EyeBot controller was quite limited in the meaning that the Motorola microcontroller does not support floats, it only emulates these calculations. The good thing with the controller is that it has a lot of in and out ports like, servo controller, motor controller, serial ports, exc… But it is maybe time to look at some more modern solution as a Intel XScale solution, for example Gumstix (http://gumstix.com) that is praiseworthy, has a lot of ready made solutions and a lot of documentation. A Nano-IPX with a robot module also could be an alternative.

Finally we think that our robot managed to perform quite well even if it did not manage to qualify during the seeding. The course gave us the insight of the difficulty in building an autonomous robot.

6 Sources

6.1 C

6.1.1 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "eyebot.h"
#include "lut.h"
//#include "lutFrontFar.h"
//#include "lutFrontClose.h"
//#include "lutLeftFar.h"
//#include "lutLeftClose.h"

#ifndef TYPES_H // This is taken from the eyebots types.h
#define TYPES_H
#define BYTE unsigned char
#define imagecolumns 82
#define imagerows 62
typedef BYTE colimage[imagerows][imagecolumns][3];
typedef BYTE image[imagerows][imagecolumns];
#endif

#include "minmax.h"
#include "ioImage.h"
#include "imageManip.h"
#include "region2.h"
#include "getMean.h"
#include "ourdrive.h"
#include "roller.h"

#ifndef BOUNDARIES
#define LEFTBOUNDARY 15
#define RIGHTBOUNDARY 65
#define GOALLEFTBOUNDARY 30
#define GOALRIGHTBOUNDARY 50
#define FIELDWIDTH 1.165f
#define FIELDHEIGHT 2.353f
#endif

#ifndef BALLTHRESHOLDMEAN
#define BALLTHRESHOLDREGIONPART 0
#define BALLTHRESHOLDMEANFAR 10
#define BALLTHRESHOLDMEANMIDDLE 15
#define BALLTHRESHOLDMEANNEAR 0
#endif

void playR2D2sound(){
                int i, freq;
                for(i=0; i<10; i++){
                                freq = (rand()%8000) + 100;
                                AUTone(freq,100);
                                while(!AUCheckTone()){}
                }
}

int relocate(VWHandle vw, ServoInfo* servotiltInfo, PositionType* robotPos){

                Object grass; grass.type = 1; grass.lastXScreen = 0; grass.lastYScreen = 0;

                servotiltInfo->state = 0; servotiltInfo->current = servotiltInfo->max;
                SERVOSet(servotiltInfo->handle, servotiltInfo->current);
```

```
		OSWait(16);

		Region region; initializeRegion(&region);

		image bin, ret;
		colimage pic;

		VWSetSpeed(vw, 0.0, 0.0);

		//check if there is enough space to drive
		CAMGetColFrame(&pic,0);
		getMeanFromLUT2(pic, &grass, 10, imagerows);

		if(grass.xScreen>LEFTBOUNDARY && grass.xScreen<RIGHTBOUNDARY && grass.yScreen<50){
			if(ourDriveStraight(vw,0.15,0.3) == MOTORSSTALLED){
				// ROBOT HAS STALLED
				//obviously there seems not enough space to drive.
				if(ourDriveStraight(vw,-0.15,0.3) == MOTORSSTALLED){
					if(ourDriveTurn(vw, M_PI, M_PI/2) ==
MOTORSSTALLED){
						// if stalled here to... then I dont what to
do
						playR2D2sound(); // Call for help
					}
				}
			}
		}

		return 1;
}

int newPosition(VWHandle vw, ServoInfo* servotiltInfo, PositionType* robotPos){
		LCDSetPrintf(4,0,"newPosition");
		return relocate(vw, servotiltInfo, robotPos);
}

int turnToGoal(VWHandle vw, PositionType* robotPos){
		LCDSetPrintf(4,0,"turnToGoal");

		float v;
		float turn;

		VWGetPosition(vw, robotPos);
		//float x = (robotPos->x - FIELDWIDTH/2); // OLD VERSION
		//float y = (FIELDHEIGHT - robotPos->y); // OLD VERSION
		float y = (robotPos->x - FIELDWIDTH/2);
		float x = (FIELDHEIGHT - robotPos->y);

		if(x == 0 && y == 0){
			v = 0;
			turn = 0;
		}
		else{
			v = atan(x/y);
			turn = -(robotPos->phi+v);

			if(robotPos->phi > -v && robotPos->phi < M_PI - v) turn = turn;
			else turn = (2*M_PI-robotPos->phi)-v;

			if( turn >= 2*M_PI )  turn = turn - (2*M_PI);
			if( turn <= -2*M_PI ) turn = turn + (2*M_PI);
		}

		if(ourDriveTurn(vw, turn, M_PI/9) == MOTORSSTALLED){
			// MUST CHECK IF STALLED

		}

		return 1;
}
```

```
int goHome(VWHandle vw, PositionType* robotPos){
		// Go to our side of the field
		if(ourDriveStraight(vw, -0.2, 0.2) == MOTORSSTALLED){
				// MUST CHECK IF STALLED
				VWSetSpeed(vw, 0.0,0.0);
		}
		// Turn 180 deg

		// Search for our goal if find go there

		// if not found keep searching a maximum of 360 deg

		return 1;
}

int main(){
		int r,c,i,j,k,key;
		float startAngle;
		int nrPhiPos = 4, nrPhiPosCounter = 0; // nrPhiPos = 3 (original value)
		int panCounter;
		float velocityScaleFactor = 2;
		float angleVelocityScaleFactor = 0.5;

		Object ball; ball.type = 1; ball.lastXScreen = 0; ball.lastYScreen = 0;
		Object enemy; enemy.type = 2;
		Object yellowGoal; yellowGoal.type = 3;
		Object blueGoal; blueGoal.type = 4;

		int goalChoice = 0;
		Object oponentGoal; oponentGoal.type = blueGoal.type;
		Object ourGoal; ourGoal.type = yellowGoal.type;

		Region region; initializeRegion(&region);

		image bin, ret;
		colimage pic;

		StateMachine statemachine; statemachine.state = START;

		VWHandle vw;
		vw = VWInit(VW_DRIVE,1);  //init v-omega interface
		if(vw == 0) { LCDPutString("VWInit Error!\n"); return 0; }
		VWStartControl(vw,7,0.3,7,0.1);
		VWSetSpeed(vw, 0, 0);
		PositionType robotPos; robotPos.x = 0.5825; robotPos.y = 0; robotPos.phi = 0;
		VWSetPosition(vw, robotPos.x, robotPos.y, robotPos.phi);

		ServoHandle servopan;
		servopan = SERVOInit(SERVO7);
		ServoInfo servopanInfo;
		servopanInfo.min = 85; servopanInfo.max = 170; servopanInfo.center = 128;
		servopanInfo.current = servopanInfo.center; servopanInfo.handle = servopan;

		servopanInfo.state = 1;
		servopanInfo.nrOfCamPos = 2;
		servopanInfo.servoData[0] = servopanInfo.center;
		servopanInfo.servoData[1] = servopanInfo.min;
		servopanInfo.servoData[2] = servopanInfo.max;
		SERVOSet(servopan, servopanInfo.center);

		ServoHandle servotilt;
		servotilt = SERVOInit(SERVO8);
		ServoInfo servotiltInfo;
  //servotiltInfo.min = 82;		// 29deg.(mid of view) We will see 0.0104 - 0.1658 m infront of us
		servotiltInfo.min = 85;			// 35deg.(mid of view) We will see 0.0242 - 0.208 m infront of us
		servotiltInfo.max = 230;		// 66deg.(mid of view) We will see from 0.12 m to the walls
		servotiltInfo.center = 170;		// 35deg.(mid of view) We will see 0.0242 - 0.208 m infront of us
		servotiltInfo.current = servotiltInfo.max;
		servotiltInfo.state = 0;
```

```
servotiltInfo.handle = servotilt;
SERVOSet(servotilt, servotiltInfo.max);  // Range for tilting 100 - 192

CAMInit(NORMAL);
CAMSet(FPS3_75, 0, 0);

// Stabilization of the camera
for(i=0; i<50; i++){
                LCDClear();
                LCDSetPrintf(0,0,"start in %ds", (50-i)/3);
                CAMGetColFrame(&pic,0);
}

LCDClear();
LCDMenu("RUN","GOAL","","EXIT");
key = KEYGet();
while(key != KEY4){
                LCDClear();
                LCDMenu("RUN","GOAL","","EXIT");
                switch(key){
                                case KEY1:
                                                LCDClear();
                                                LCDMenu("","","","END");

                                                /*
                                                 *              0: FAR (MEAN/REGION)
                                                 *              1: nrPixels: 314
                                                 *              2: x:0.234 y:0.634
                                                 *              3: Go to ball / Went to ball / Not found
ball
                                                 *              4:
                                                 *              5:
                                                 */


                                                while(key != KEY4){
                                                                //OSWait(100);
                                                                LCDClear();
                                                                switch(statemachine.state){
                                                                                case START:

LCDSetPrintf(0,0,"START");

VWGetPosition(vw, &robotPos);

startAngle = robotPos.phi;

statemachine.state = FAR;

panCounter = 0; // Reset pan position.

nrPhiPosCounter = 0;

break;
                                                                                case FAR:

LCDSetPrintf(0,0,"FAR (MEAN)");

SERVOSet(servopan, servopanInfo.servoData[panCounter]);

SERVOSet(servotilt, servotiltInfo.max);

OSWait(16); // Value taken from servo datasheet


CAMGetColFrame(&pic,0); // Take picture

getMeanFromLUT2(pic, &ball, 30, 62); // Look for ball in mean (tweek this values)
```

```
		LCDSetPrintf(1,0,"nrPixels: %d", ball.nrPixelsFound);

		if(ball.nrPixelsFound > BALLTHRESHOLDMEANFAR){ // LOOK FOR MEAN
		LCDSetPrintf(2,0,"x:%d y:%d", ball.xScreen, ball.yScreen);


		LCDSetPrintf(3,0,"Go to ball");
		if(panCounter == 0){
				if(ball.xScreen > GOALLEFTBOUNDARY && ball.xScreen < GOALRIGHTBOUNDARY){
						ourDriveStraight(vw, 0.2, 0.1*velocityScaleFactor);
				}
				else{
						if(41-ball.xScreen > 0){ // LEFT screen side
								ourDriveTurn(vw, M_PI/36,
M_PI/3*angleVelocityScaleFactor);
								ourDriveStraight(vw, 0.1, 0.1*velocityScaleFactor);
						}
						else{ // RIGHT screen side
								ourDriveTurn(vw, -M_PI/36,
M_PI/3*angleVelocityScaleFactor);
								ourDriveStraight(vw, 0.1, 0.1*velocityScaleFactor);
						}
				}
		}
		else{
				if(panCounter == 1){ // LEFT camera position
						ourDriveTurn(vw, M_PI/9, M_PI/3*angleVelocityScaleFactor);
						ourDriveStraight(vw, 0.05, 0.1*velocityScaleFactor);
						panCounter = 0;
				}
				else{ // RIGHT camera position
						ourDriveTurn(vw, -M_PI/9, M_PI/3*angleVelocityScaleFactor);
						ourDriveStraight(vw, 0.05, 0.1*velocityScaleFactor);
						panCounter = 0;
				}
		}
		LCDSetPrintf(3,0,"Went to ball");
```

```
                panCounter = 0; // Reset pan position.
                statemachine.state = NEAR;
                break;
                                                                                                }
                else{ // LOOK FOR REGION
                LCDSetPrintf(0,0,"FAR (REGION)");

                binarize(pic, &ball, bin);
                findRegion(bin, &region, ret, 5, 30);
                cleanupRegion(ret, &region, 5, 30);
                getMean2(ret, &ball, 5, 30);

                LCDSetPrintf(1,0,"nrPixels: %d", ball.nrPixelsFound);

                if(ball.nrPixelsFound > BALLTHRESHOLDREGIONPART && ball.nrPixelsFound < 2007){
                                LCDSetPrintf(2,0,"x:%d y:%d", ball.xScreen, ball.yScreen);

                                LCDSetPrintf(3,0,"Go to ball");
                                if(panCounter == 0){
                                                if(ball.xScreen > GOALLEFTBOUNDARY && ball.xScreen <
GOALRIGHTBOUNDARY){
                                                                ourDriveStraight(vw, 0.3, 0.1*velocityScaleFactor);
                                                }
                                                else{
                                                                if(41-ball.xScreen > 0){ // LEFT screen side
                                                                                ourDriveTurn(vw, M_PI/36,
M_PI/3*angleVelocityScaleFactor);
                                                                                ourDriveStraight(vw, 0.2,
0.1*velocityScaleFactor);
                                                                }
                                                                else{ // RIGHT screen side
                                                                                ourDriveTurn(vw, -M_PI/36,
M_PI/3*angleVelocityScaleFactor);
                                                                                ourDriveStraight(vw, 0.2,
0.1*velocityScaleFactor);
                                                                }
                                                }
```

```
                }
                else{
                        if(panCounter == 1){ // LEFT camera position
                                ourDriveTurn(vw, M_PI/9,
M_PI/3*angleVelocityScaleFactor);
                                ourDriveStraight(vw, 0.2, 0.1*velocityScaleFactor);
                                panCounter = 0;
                        }
                        else{ // RIGHT camera position
                                ourDriveTurn(vw, -M_PI/9,
M_PI/3*angleVelocityScaleFactor);
                                ourDriveStraight(vw, 0.2, 0.1*velocityScaleFactor);
                                panCounter = 0;
                        }
                }
                LCDSetPrintf(3,0,"Went to ball");

                panCounter = 0; // Reset pan position.
                statemachine.state = FAR;
                break;
        }
        else{
                LCDSetPrintf(2,0,"Not found ball");
                if(panCounter < servopanInfo.nrOfCamPos){
                        panCounter++;
                        statemachine.state = FAR;
                        break;
                }
                else{
                        panCounter = 0; // Reset pan position.
                        statemachine.state = NEAR;
                        break;
                }
        }
                                                                                }
        break;
                                                        case NEAR:
```

```
LCDSetPrintf(0,0,"NEAR");

SERVOSet(servopan, servopanInfo.center);
SERVOSet(servotilt, servotiltInfo.min);
OSWait(16); // Value taken from servo datasheet

CAMGetColFrame(&pic,0); // Take picture
getMeanFromLUT2(pic, &ball, 0, 62); // Look for ball in mean (tweek this values)

LCDSetPrintf(1,0,"nrPixels: %d", ball.nrPixelsFound);

if(ball.nrPixelsFound > BALLTHRESHOLDMEANNEAR){ // LOOK FOR NEAR
LCDSetPrintf(2,0,"x:%d y:%d", ball.xScreen, ball.yScreen);


rollerDribbler(); // Roller ON


LCDSetPrintf(3,0,"Go to ball");
if(ourDriveStraight(vw,0.10,0.3*velocityScaleFactor) == MOTORSSTALLED){
                        VWSetSpeed(vw, 0, 0);
}
LCDSetPrintf(3,0,"Have ball");


panCounter = 0; // Reset pan position.
statemachine.state = HASBALL;
break;
                                                                                                                    }
else{
panCounter = 0; // Reset pan position.
statemachine.state = MIDDLE;
break;
                                                                                                                    }
break;
                                                                                                    case MIDDLE:
LCDSetPrintf(0,0,"MIDDLE");

SERVOSet(servopan, servopanInfo.servoData[panCounter]);
SERVOSet(servotilt, servotiltInfo.center);
OSWait(16); // Value taken from servo datasheet
```

```
CAMGetColFrame(&pic,0); // Take picture
getMeanFromLUT2(pic, &ball, 0, 62); // Look for ball in mean (tweek this values)

LCDSetPrintf(1,0,"nrPixels: %d", ball.nrPixelsFound);

if(ball.nrPixelsFound > BALLTHRESHOLDMEANMIDDLE){ // LOOK FOR NEAR
LCDSetPrintf(2,0,"x:%d y:%d", ball.xScreen, ball.yScreen);


LCDSetPrintf(3,0,"Go to ball");
if(panCounter == 0){
                if(ball.xScreen > GOALLEFTBOUNDARY && ball.xScreen < GOALRIGHTBOUNDARY){
                                ourDriveStraight(vw, 0.1, 0.1*velocityScaleFactor);
                }
                else{
                                if(41-ball.xScreen > 0){ // LEFT screen side
                                                ourDriveTurn(vw, M_PI/36,
M_PI/3*angleVelocityScaleFactor);
                                                ourDriveStraight(vw, 0.02, 0.1*velocityScaleFactor);
                                }
                                else{ // RIGHT screen side
                                                ourDriveTurn(vw, -M_PI/36,
M_PI/3*angleVelocityScaleFactor);
                                                ourDriveStraight(vw, 0.02, 0.1*velocityScaleFactor);
                                }
                }
        }
        else{
                if(panCounter == 1){ // LEFT camera position
                                ourDriveTurn(vw, M_PI/9, M_PI/3*angleVelocityScaleFactor);
                                panCounter = 0;
                }
                else{ // RIGHT camera position
                                ourDriveTurn(vw, -M_PI/9, M_PI/3*angleVelocityScaleFactor);
                                panCounter = 0;
                }
        }
        LCDSetPrintf(3,0,"Went to ball");
```

```
                panCounter = 0; // Reset pan position.
                statemachine.state = NEAR;
                break;
                                                                                                                }
                else{
                if(panCounter < servopanInfo.nrOfCamPos){
                                panCounter++; // Next cam. pos.
                                statemachine.state = MIDDLE;
                                break;
                }
                else{
                                // IF ROBOT HAS TURN 360 deg
                                if(nrPhiPosCounter >= nrPhiPos){
                                                nrPhiPosCounter = 0;
                                                newPosition(vw, &servotiltInfo, &robotPos);
                                }
                                else{
                                                nrPhiPosCounter++;
                                                if(ourDriveTurn(vw, 2*M_PI/nrPhiPos, M_PI/3*angleVelocityScaleFactor) ==
MOTORSSTALLED){
                                                                newPosition(vw, &servotiltInfo, &robotPos);
                                                                nrPhiPosCounter = 0;
                                                }
                                }
                                panCounter = 0; // Reset pan position.
                                statemachine.state = FAR;
                                break;
                }
                                                                                                                }
                break;
                                                                                                case HASBALL:
                LCDSetPrintf(0,0,"HASBALL");
                //turnToGoal(vw,&robotPos);
                panCounter = 0;
                statemachine.state = FINDGOAL;
                break;
```

```
                                                                    case FINDGOAL:
LCDSetPrintf(0,0,"FINDGOAL (BALL)");

SERVOSet(servopan, servopanInfo.center);
SERVOSet(servotilt, servotiltInfo.min);
OSWait(16); // Value taken from servo datasheet

CAMGetColFrame(&pic,0); // Take picture
getMeanFromLUT2(pic, &ball, 0, 62); // Look for ball in mean (tweek this values)

LCDSetPrintf(1,0,"nrPixels: %d", ball.nrPixelsFound);

if(ball.nrPixelsFound > BALLTHRESHOLDMEANNEAR){
LCDSetPrintf(2,0,"x:%d y:%d", ball.xScreen, ball.yScreen);

SERVOSet(servopan, servopanInfo.servoData[panCounter]);
SERVOSet(servotilt, servotiltInfo.max);
OSWait(16); // Value taken from servo datasheet

LCDSetPrintf(0,0,"FINDGOAL (GOAL)");

CAMGetColFrame(&pic,0);
binarize(pic, &oponentGoal, bin);
findRegion(bin, &region, ret, 0, imagerows);
cleanupRegion(ret, &region, 0, imagerows);
getMean(ret, &oponentGoal);

LCDSetPrintf(1,0,"nrPixels: %d", oponentGoal.nrPixelsFound);

if(oponentGoal.nrPixelsFound > 50 && oponentGoal.nrPixelsFound < 800){
                LCDSetPrintf(2,0,"x:%d y:%d", oponentGoal.xScreen, oponentGoal.yScreen);

                // If goal is in middle of the camera view kick the ball
                servopanInfo.state = panCounter;

                switch(servopanInfo.state){
                        case 0: // CENTER
```

```
                                        if(oponentGoal.xScreen > GOALLEFTBOUNDARY &&
oponentGoal.xScreen < GOALRIGHTBOUNDARY){
                                                kickBall(vw);
                                                goHome(vw, &robotPos);
                                                statemachine.state = START;
                                                break;
                                        }
                                        else{
                                                if(41-oponentGoal.xScreen > 0){
                                                        if(ourDriveTurn(vw,
M_PI/36, (M_PI/18)*angleVelocityScaleFactor) == MOTORSSTALLED){
                                                                //
MUST CHECK IF STALLED
                        newPosition(vw, &servotiltInfo, &robotPos);
                                                        }
                                                }
                                                else{
                                                        if(ourDriveTurn(vw, -
M_PI/36, (M_PI/18)*angleVelocityScaleFactor) == MOTORSSTALLED){
                                                                //
MUST CHECK IF STALLED
                        newPosition(vw, &servotiltInfo, &robotPos);
                                                        }
                                                }
                                        }
                                        break;
                                case 1: // LEFT
                                        if(ourDriveTurn(vw, M_PI/6,
(M_PI/18)*angleVelocityScaleFactor) == MOTORSSTALLED){
                                                // MUST CHECK IF STALLED
                                                newPosition(vw, &servotiltInfo,
&robotPos);
                                        }
                                        panCounter = 0;
                                        break;
                                case 2: // RIGHT
                                        if(ourDriveTurn(vw, -M_PI/6,
(M_PI/18)*angleVelocityScaleFactor) == MOTORSSTALLED){
```

```
                                                                        // MUST CHECK IF STALLED
                                                                        newPosition(vw, &servotiltInfo,
&robotPos);
                                                                }
                                                                panCounter = 0;
                                                                break;
                                        }
                                }
                                else{
                                        // LOOK FOR MORE CAM POS
                                        if(panCounter < servopanInfo.nrOfCamPos){
                                                panCounter++;
                                        }
                                        else{
                                                if(nrPhiPosCounter >= nrPhiPos){
                                                        nrPhiPosCounter = 0;
                                                        newPosition(vw, &servotiltInfo, &robotPos);
                                                }
                                                else{
                                                        if(ourDriveTurn(vw, 2*M_PI/nrPhiPos,
(M_PI/18)*angleVelocityScaleFactor) == MOTORSSTALLED){
                                                                // MUST CHECK IF STALLED

                                                        }
                                                }
                                                panCounter = 0; // Reset pan position.
                                                statemachine.state = FINDGOAL;
                                                break;
                                        }
                                }
                                                                                                }
                                else{
                                rollerOff();
                                panCounter = 0;
                                statemachine.state = NEAR;
                                                                                                }
                                break;
```

```
                                                                        }
                                                                        key = KEYRead();
                                                                }
                                                                key = 0;
                                                                break;
                                                        case KEY2:

                                                                while(key != KEY4){
                                                                        LCDClear();
                                                                        LCDMenu("YLW","BLUE","","END");
                                                                        LCDSetPrintf(0,0,"OPONENT GOAL");
                                                                        if(goalChoice == 0){

                LCDSetPrintf(1,0,"BLUE");
                                                                        }
                                                                        else{

                LCDSetPrintf(1,0,"YELLOW");
                                                                        }
                                                                        key = KEYGet();
                                                                        switch(key){
                                                                                case KEY1:

                goalChoice = 1;

                oponentGoal.type = yellowGoal.type;

                ourGoal.type = blueGoal.type;

                break;
                                                                                case KEY2:

                goalChoice = 0;

                oponentGoal.type = blueGoal.type;

                ourGoal.type = yellowGoal.type;

                break;
                                                                                case KEY4:
                                                                                        key
= KEY4;

                break;
                                                                        }
                                                                }
                                                                break;
                                                        case KEY3:
                                                                break;
                                                        default:
                                                                break;
                        }
                        key = KEYGet();
                }

                VWSetSpeed(vw, 0, 0);
                VWRelease(vw);

                CAMRelease();
                SERVORelease(servopan);
                SERVORelease(servotilt);

                return 0;
}
```

6.1.2 minmax.h

```
#ifndef MINMAX
#define MINMAX

float maxf(float a, float b)
{
		if(a>b)
				return a;
		else
				return b;
}

int max(int a, int b)
{
		if(a>b)
				return a;
		else
				return b;
}

int min(int a, int b, int c, int d, int e)
{
		if((a<=b) & (a<=c) & (a<=d) & (a<=e))
				return a;
		else if((b<=a) & (b<=c) & (b<=d) & (b<=e))
				return b;
		else if((c<=a) & (c<=b) & (c<=d) & (c<=e))
				return c;
		else if((d<=a) & (d<=b) & (d<=c) & (d<=e))
				return d;
		else
				return e;
}

#endif
```

6.1.3 ioImage.h

```
#ifndef IOIMAGE
#define IOIMAGE

#ifndef TYPES_H // This is taken from the eyebots types.h
#define TYPES_H
typedef BYTE unsigned char
#define imagecolumns 82
#define imagerows 62
typedef BYTE colimage[imagerows][imagecolumns][3];
typedef BYTE image[imagerows][imagecolumns];
#endif

void readImage(const char* fileName, colimage dest){
		FILE *in = fopen(fileName,"r");
		int i,r,c,val;
		char tmp[8];

		for(i=0; i<4; i++){
			fscanf(in, "%s", tmp);
		}

		for(r=0; r<imagerows; r++){
			for(c=0; c<imagecolumns; c++){
				for(i=0; i<3; i++){
					fscanf(in, "%d", &val);
					dest[r][c][i] = val;
				}
			}
		}

		fclose(in);
}

void readImageGrey(const char* fileName, image dest){
		FILE *in = fopen(fileName,"r");
		int i,r,c,val;
		char tmp[8];

		for(i=0; i<4; i++){
			fscanf(in, "%s", tmp);
		}

		for(r=0; r<imagerows; r++){
			for(c=0; c<imagecolumns; c++){
				fscanf(in, "%d", &val);
				dest[r][c] = val;
			}
		}

		fclose(in);
}

void writeImage(const char* fileName, colimage src){
		FILE *out = fopen(fileName,"w+");
		int i,r,c;

		fprintf(out, "P3\n%d %d\n255\n", imagecolumns, imagerows);

		for(r=0; r<imagerows; r++){
			for(c=0; c<imagecolumns; c++){
				for(i=0; i<3; i++){
					fprintf(out, "%d ", src[r][c][i]);
				}
				fprintf(out, "\n");
			}
		}
```

```
		fclose(out);
}

void writeImageGrey(const char* fileName, image src){
		FILE *out = fopen(fileName,"w+");
		int r,c;

		fprintf(out, "P2\n%d %d\n15\n", imagecolumns, imagerows);

		for(r=0; r<imagerows; r++){
				for(c=0; c<imagecolumns; c++){
						fprintf(out, "%d ", src[r][c]);
						fprintf(out, "\n");
				}
		}

		fclose(out);
}

#endif
```

6.1.4 imageManip.h

```
#ifndef IMAGEMANIP
#define IMAGEMANIP

#ifndef TYPES_H // This is taken from the eyebots types.h
#define TYPES_H
typedef BYTE unsigned char
#define imagecolumns 82
#define imagerows 62
typedef BYTE colimage[imagerows][imagecolumns][3];
typedef BYTE image[imagerows][imagecolumns];
#endif

#include "ourStructs.h"

void blurImage(colimage src, colimage dest){
                int tmp[imagerows+2][imagecolumns+2][3];
                int i,r,c;

                // Copy the source image to the center part of tmp
                for(r=0; r<imagerows; r++){
                                for(c=0; c<imagecolumns; c++){
                                                for(i=0; i<3; i++){
                                                                tmp[r+1][c+1][i] = src[r][c][i];
                                                }
                                }
                }

                // Make the borders black.
                for(r=0; r<(imagerows+2); r++){
                                for(i=0; i<3; i++){
                                                tmp[r][0][i] = 0;
                                                tmp[r][imagecolumns+1][i] = 0;
                                }
                }
                for(c=0; c<(imagecolumns+2); c++){
                                for(i=0; i<3; i++){
                                                tmp[0][c][i] = 0;
                                                tmp[imagerows+1][c][i] = 0;
                                }
                }

                // Blur the image with a gaussian alike mask
                int mask[3][3] =    {{1,1,1},
                                                                                                {1,4,1},
                                                                                                {1,1,1}};
                int sum = 0;
                int m, n, result;
                for(m=0; m<3; m++){
                                for(n=0; n<3; n++){
                                                sum += mask[m][n];
                                }
                }
                for(r=1; r<(imagerows+1); r++){
                                for(c=1; c<(imagecolumns+1); c++){
                                                for(i=0; i<3; i++){
                                                                result = 0;
                                                                for(m=0; m<3; m++){
                                                                                for(n=0; n<3; n++){
                                                                                                result += tmp[r-
1+m][c-1+n][i] * mask[m][n];
                                                                                }
                                                                }
                                                                result /= sum;
                                                                dest[r-1][c-1][i] = (unsigned char)result;
                                                }
                                }
```

```
		}
}

void binarize(colimage src, Object* object, image bin){
		int r,c;
		object->xScreen = 0; object->yScreen = 0; object->nrPixelsFound = 0;

		for(r=0; r<imagerows; r++){
				for(c=0; c<imagecolumns; c++){
						if(lut[(int)maxf((src[r][c][0])/10-1,0)][(int)maxf((src[r][c][1])/10-
1,0)][(int)maxf((src[r][c][2])/10-1,0)] == (object->type))
						{
								bin[r][c]=15; // White
								object->xScreen += c;
								object->yScreen += r;
								(object->nrPixelsFound)++;
						}else{
								bin[r][c]=0; // Black
						}
				}
		}
		if((object->nrPixelsFound) > 0){
				object->xScreen /= object->nrPixelsFound;
				object->yScreen /= object->nrPixelsFound;
		}
}

#endif
```

6.1.5 region2.h

```
#ifndef REGION
#define REGION

#include "ourStructs.h"


/*
 * upperbound is "upper" as being close to the upper edge of the image
 * lowerbound is "lower" as being close to the lower edge of the image
 * 0 <= upperbound < lowerbound <= imagerows
 */

void initializeRegion(Region* region){
                        int i=0;
                        for(i=0; i<MAX_NR_REGIONS; i++)
                        {
                          region->sizes[i]=0;
                          region->index[i]=-1;
                        }

                        region->nr_of_regions=0; //actual number of regions found - 1
                        region->max_idx=-1;
                        region->max_size=0;
}

int min_val(image ret, int i, int j, int upperbound, int lowerbound)
{
                        int ny=MAX_NR_REGIONS;
                        if(i<=upperbound)
                        {
                                        if(j==0)
                                        {
                                                        //only the right and bottom pixel are of intrest
                                                        //r[i][j]
                                                        ny=min(MAX_NR_REGIONS, MAX_NR_REGIONS, ret[i+1][j], ret[i][j+1],
ret[i][j]);
                                        }
                                        else if(j==imagecolumns-1)
                                        {
                                                        //we disregard the upper pixel and right pixel
                                                        ny=min(MAX_NR_REGIONS, ret[i][j-1], ret[i+1][j], MAX_NR_REGIONS,
ret[i][j]);
                                        }
                                        else
                                        {
                                                        //j is in between, only leave the upper pixel out
                                                        ny=min(MAX_NR_REGIONS, ret[i][j-1], ret[i+1][j], ret[i][j+1], ret[i][j]);
                                        }
                        }
                        else if(i==lowerbound-1)
                        {
                                        if(j==0)
                                        {
                                                        //only the right and upper pixel are of intrest
                                                        ny=min(ret[i-1][j], MAX_NR_REGIONS, MAX_NR_REGIONS, ret[i][j+1],
ret[i][j]);
                                        }
                                        else if(j==imagecolumns-1)
                                        {
                                                        //we disregard the bottom pixel and right pixel
                                                        ny=min(ret[i-1][j], ret[i][j-1], MAX_NR_REGIONS, MAX_NR_REGIONS,
ret[i][j]);
                                        }
                                        else
                                        {
                                                        //j is in between, only leave the bottom pixel out
```

```
                                        ny=min(ret[i-1][j], ret[i][j-1], MAX_NR_REGIONS, ret[i][j+1], ret[i][j]);
                                }
                        }
                        else
                        {
                                if(j==0)
                                {
                                        //only the right, upper and bottom pixel are of intrest
                                        ny=min(ret[i-1][j], MAX_NR_REGIONS, ret[i+1][j], ret[i][j+1], ret[i][j]);
                                }
                                else if(j==imagecolumns-1)
                                {
                                        //we disregard the right pixel
                                        ny=min(ret[i-1][j], ret[i][j-1], ret[i+1][j], MAX_NR_REGIONS, ret[i][j]);
                                }
                                else
                                {
                                        //j is in between, leave nothing out!
                                        ny=min(ret[i-1][j], ret[i][j-1], ret[i+1][j], ret[i][j+1], ret[i][j]);
                                }
                        } //if(i==0)
                        return ny;
}

void set_coord(image bin, image ret, int* i, int* j, int upperbound, int lowerbound)
{
                        if((*i)>upperbound)
                        {
                                if(bin[(*i)-1][(*j)] && ret[(*i)-1][(*j)]!=ret[(*i)][(*j)])
                                {
                                        (*i)--;
                                        (*j)--;
                                }else if((*j)>0 && bin[(*i)][(*j)-1] && ret[(*i)][(*j)-1]!=ret[(*i)][(*j)])
                                {
                                        (*j)-=2;
                                }
                        }else if((*j)>0)
                        {
                                if(bin[(*i)][(*j)-1] && ret[(*i)][(*j)-1]!=ret[(*i)][(*j)])
                                {
                                        (*j)-=2;
                                }
                        }
}

int findRegion(image bin, Region* region, image ret, int upperbound, int lowerbound)
{
                        //struct reg needs to be initialized!
                        // sizes to be all 0
                        // index all -1
                        // nr_of_regions = 0

                        if(upperbound<0 || upperbound >= lowerbound || lowerbound>imagerows)
                        {
        upperbound = 0;
        lowerbound = imagerows;
    }

                        if(region->nr_of_regions != 0)
                                return -1;
                        int nr = 1; //local variable, to not type so much

                        //need to set each "pixel" in r to MAX_NR_REGIONS
                        int i,j;
                        for(i=0; i<imagerows;i++)
                        {
                                for(j=0; j<imagecolumns;j++)
                                {
                                        ret[i][j]= MAX_NR_REGIONS;
                                }
```

```
}

int changed = 0, neu=0; //to recognise if a value was altered, neu is german for new
BYTE old = MAX_NR_REGIONS; //old value in r, to check for change
BYTE ny = MAX_NR_REGIONS; //the value which comes out of min()

int test=0;

for(i=upperbound; i<lowerbound;i++)
{
		for(j=0;j<imagecolumns;j++)
		{
				changed = 0;
				neu=0;
				old = MAX_NR_REGIONS;
				ny = MAX_NR_REGIONS;

				if(bin[i][j])
				{
						old = ret[i][j];
						ny = min_val(ret, i, j, upperbound, lowerbound);
						if((ny==old) & (old==MAX_NR_REGIONS))
						{
								neu=1;
								changed=1;
						}
						else if(ny!=old)
						{
								neu=0;
								changed=1;
						}
				}//if bin

				if(neu)
				{
						ret[i][j]=nr;
						region->sizes[nr]++;
						region->index[nr]=i*imagecolumns+j;
						test = nr;
						nr++;
				}else if(changed)
				{
						ret[i][j]=ny;
						region->sizes[ny]++;
						if(region->sizes[old]==1)
						{
								region->sizes[old]--;
								region->index[old]=-1;
						}
						else
						{
								region->sizes[old]--;
						}
						set_coord(bin, ret, &i, &j, upperbound, lowerbound);
				}//if neu || changed

		}//for j
}//for i
int max_size=0, max_idx=-1, max_i=0;
for(i=0; i<MAX_NR_REGIONS; i++)
{
		if(region->sizes[i]>max_size)
		{
				max_size = region->sizes[i];
				max_idx = region->index[i];
				max_i=i;
		}
}
region->max_idx=max_idx;
region->max_size = max_size;
```

```
                return max_idx;
}

int cleanupRegion(image ret, Region* region, int upperbound, int lowerbound)
{
    /*
     * go through arrays and get rid of too small regions and
     * format surviving ones into a nice array with no holes in it.
     */

    //first delete small regions.
    //therefore define swaparray a[i]=j -> i becomes j
    int swap[MAX_NR_REGIONS+1]; //need to get a value for swap[r[i][j]] even when r[i][j] is MAX_NR_REGIONS
    int i,j,v=0; //v is value of ret[i][j]

    //get back indexes on biggest region to get its value
    div_t temp;
    temp = div(region->max_idx, imagecolumns);
    i = temp.quot;
    j = temp.rem;
    v = ret[i][j];

    for(i=0;i<MAX_NR_REGIONS+1;i++)
    {
      if(i == v)
        swap[i]=15;
      else
                swap[i]=0;
    }

                //now go through regions with swap array
                //we can work on r because we go straight through and never backwards
                //nor twice over it!
                for(i=upperbound; i<lowerbound;i++)
                {
                        for(j=0; j<imagecolumns;j++)
                        {
                                ret[i][j]=swap[ret[i][j]];
                        }
                }

                return v;
}

#endif
```

6.1.6 getMean.h

```
#ifndef GETMEAN
#define GETMEAN

#include "ourStructs.h"

void getMeanFromLUT(colimage src, Object* object){
		int r,c;
		object->xScreen = 0; object->yScreen = 0; object->nrPixelsFound = 0;

		for(r=0; r<imagerows; r++){
			for(c=0; c<imagecolumns; c++){
				if(lut[(int)maxf((src[r][c][0])/10-1,0)][(int)maxf((src[r][c][1])/10-
1,0)][(int)maxf((src[r][c][2])/10-1,0)] == (object->type)){
					object->xScreen += c;
					object->yScreen += r;
					(object->nrPixelsFound)++;
				}
			}
		}
		if((object->nrPixelsFound) > 0){
			object->xScreen /= object->nrPixelsFound;
			object->yScreen /= object->nrPixelsFound;
		}
}

void getMeanFromLUT2(colimage src, Object* object, int lineMin, int lineMax){
		int r,c;
		object->xScreen = 0; object->yScreen = 0; object->nrPixelsFound = 0;

		for(r=lineMin; r<lineMax; r++){
			for(c=0; c<imagecolumns; c++){
				if(lut[(int)maxf((src[r][c][0])/10-1,0)][(int)maxf((src[r][c][1])/10-
1,0)][(int)maxf((src[r][c][2])/10-1,0)] == (object->type)){
					object->xScreen += c;
					object->yScreen += r;
					(object->nrPixelsFound)++;
				}
			}
		}
		if((object->nrPixelsFound) > 0){
			object->xScreen /= object->nrPixelsFound;
			object->yScreen /= object->nrPixelsFound;
		}
}

void getMean(image src, Object* object){
		int r,c;
		object->xScreen = 0; object->yScreen = 0; object->nrPixelsFound = 0;

		for(r=0; r<imagerows; r++){
			for(c=0; c<imagecolumns; c++){
				if(src[r][c] == 15){
					object->xScreen += c;
					object->yScreen += r;
					(object->nrPixelsFound)++;
				}
			}
		}
		if((object->nrPixelsFound) > 0){
			object->xScreen /= object->nrPixelsFound;
			object->yScreen /= object->nrPixelsFound;
		}
}

void getMean2(image src, Object* object, int lineMin, int lineMax){
		int r,c;
```

```
		object->xScreen = 0; object->yScreen = 0; object->nrPixelsFound = 0;

		for(r=lineMin; r<lineMax; r++){
			for(c=0; c<imagecolumns; c++){
				if(src[r][c] == 15){
					object->xScreen += c;
					object->yScreen += r;
					(object->nrPixelsFound)++;
				}
			}
		}
		if((object->nrPixelsFound) > 0){
			object->xScreen /= object->nrPixelsFound;
			object->yScreen /= object->nrPixelsFound;
		}
}

#endif
```

6.1.7 ourdrive.h

```
#include "eyebot.h"

#ifndef OURDRIVE

#define MOTORSSTALLED 1

int ourDriveWait(VWHandle);
int ourDriveStraight(VWHandle handle, meter delta, meterPerSec v);
int ourDriveTurn(VWHandle handle, radians delta, radPerSec w);

/*
int ourDriveWait(VWHandle handle)
{
  while(!VWDriveDone(handle))
    {
      if(VWStalled(handle)) return MOTORSSTALLED;
      else OSWait(10);
    }
  return 0;
}


int ourDriveStraight(VWHandle handle, meter delta, meterPerSec v)
{
  int retval = 0;
  VWDriveStraight(handle, delta, v);
  return ourDriveWait(handle);
}


int ourDriveTurn(VWHandle handle, radians delta, radPerSec w)
{
  VWDriveTurn(handle, delta, w);
  return ourDriveWait(handle);
}*/

int ourDriveWait(VWHandle handle)
{
  while(!VWDriveDone(handle))
    {
      if(VWStalled(handle)) return MOTORSSTALLED;
      else OSWait(10);
    }
  return 0;
}


int ourDriveStraight(VWHandle vw, meter delta, meterPerSec v)
{
  int stalled=0;
  VWDriveStraight(vw, delta, v);
  /*
  while(VWDriveRemain(vw)>0.0 && !stalled)
  {
                        OSWait(10);
                        stalled = VWStalled(vw);
  }
  return stalled;
  */
  VWDriveWait(vw);
  return 0;
}


int ourDriveTurn(VWHandle vw, radians delta, radPerSec w)
{
```

```
                int stalled=0;
                VWDriveTurn(vw, delta, w);
                /*
                while(VWDriveRemain(vw)>0.0 && !stalled)
                {
                                OSWait(10);
                                stalled = VWStalled(vw);
                }
                return stalled;*/
                VWDriveWait(vw);
  return 0;
}

#endif
```

6.1.8 roller.h

```
#include "eyebot.h"

const BYTE LOW = 0x00;
const BYTE KICKER   = 0x02; // 00000010
const BYTE DRIBBLER = 0x01; // 00000001
const BYTE ROLLER   = 0xFC; // Mask for the roller signals, 00000011

void rollerOff();
void rollerDribbler();
void rollerKicker();
void kickBall(VWHandle);
void kickBallConf(VWHandle, float, float);

/* Test main function
int main()
{
  int key = 0;


  VWHandle vw;
  vw = VWInit(VW_DRIVE,1);  //init v-omega interface
  if(vw == 0) { LCDPutString("VWInit Error!\n"); return 0; }
  VWStartControl(vw,7,0.3,7,0.1);
  VWSetSpeed(vw, 0, 0);

  LCDMenu("DRBL","KICK","OFF","EXIT");
  key = KEYGet();
  while(key != KEY4){
    switch(key){
    case KEY1:
      rollerDribbler();
      break;
    case KEY2:
      OSWait(100);
      kickBall(vw);
      break;
    case KEY3:
      rollerOff();
      break;
    default:
      rollerOff();
      break;
    }
    key = KEYGet();
  }

  rollerOff();
  return 0;
}
*/

// Turn off the roller
void rollerOff()
{
  OSWriteOutLatch(0, ROLLER, LOW);
}
// Start roller for backspin on the ball
void rollerDribbler()
{
  OSWriteOutLatch(0, ROLLER, DRIBBLER);
}
// Start roller for kickspin on the ball
void rollerKicker()
{
  OSWriteOutLatch(0, ROLLER, KICKER);
}
```

```
/*
 * kickBall(), kickBallConf - Attemts to kick the ball forward
 * Needs the VW handle
 * Default attack distance is 10 cm and the speed is 5 m/s.
 * This can be configurated by giving the function 2 float values (1 is 100%).
 */
void kickBall(VWHandle handle)
{
    kickBallConf(handle, 1, 1);
}
void kickBallConf(VWHandle handle, float attackDistProc, float attackPowerProc)
{
  // Get state of roller and turn it off if it's on.
  rollerDribbler();
  OSWait(100);

  // Drive back to get a good distance to the ball.
  ourDriveStraight(handle, 0.10*attackDistProc, 0.5*attackPowerProc);

  // Turn off the annoying roller
  rollerOff();

  // Drive back to get a good distance to the ball.
  ourDriveStraight(handle, -0.10*attackDistProc, 0.5*attackPowerProc);

  // Start the roller for kicking
  rollerKicker();

  // Attack the ball and go past the ballpoition by 10 cm
  ourDriveStraight(handle, 0.10*attackDistProc+0.20, 5*attackPowerProc);

  // Turn off the annoying roller
  rollerOff();

  // Drive back to the start position.
  ourDriveStraight(handle, -0.20, 0.5*attackPowerProc);
}
```

6.2 Matlab

6.2.1 Learning.m

```
clear

ennemiPoints = [255 255 255]';
files = dir('oponent');
names = {files.name};
names = names(3:end);
for i = 1:length(names)
    image = double(imread(['oponent' filesep cell2mat(names(i))],'ppm'));
    ennemiPoints = learn(ennemiPoints, image);
end
ennemiPoints = ennemiPoints(:,2:end);
disp('Learning Ennemi : done');


goalBleuPoints = [255 255 255]';
files = dir('goal blue');
names = {files.name};
names = names(3:end);
for i = 1:length(names)
    image = double(imread(['goal blue' filesep cell2mat(names(i))],'ppm'));
    goalBleuPoints = learn(goalBleuPoints, image);
end
goalBleuPoints = goalBleuPoints(:,2:end);
disp('Learning GoalBleu : done');

TapisPoints = [255 255 255]';
files = dir('grass');
names = {files.name};
names = names(3:end);
for i = 1:length(names)
    image = double(imread(['grass' filesep cell2mat(names(i))],'ppm'));
    TapisPoints = learn(TapisPoints, image);
end
TapisPoints = TapisPoints(:,2:end);
disp('Learning Tapis : done');

BallePoints = [255 255 255]';
files = dir('ball');
names = {files.name};
names = names(3:end);
for i = 1:length(names)
    image = double(imread(['ball' filesep cell2mat(names(i))],'ppm'));
    BallePoints = learn(BallePoints, image);
end
BallePoints = BallePoints(:,2:end);
disp('Learning Balle : done');

GoalJaunePoints = [255 255 255]';
files = dir('goal yellow');
names = {files.name};
names = names(3:end);
for i = 1:length(names)
    image = double(imread(['goal yellow' filesep cell2mat(names(i))],'ppm'));
GoalJaunePoints = learn(GoalJaunePoints, image);
end
GoalJaunePoints = GoalJaunePoints(:,2:end);
disp('Learning GoalJaune : done');

MurPoints = [255 255 255]';
files = dir('wall');
names = {files.name};
names = names(3:end);
```

```
for i = 1:length(names)
    image = double(imread(['wall' filesep cell2mat(names(i))],'ppm'));
MurPoints = learn(MurPoints, image);
end
MurPoints = MurPoints(:,2:end);
disp('Learning Mur : done');



ModelEnnemi = adaptGauss2([0,88,256],[0,256],[0,40,80,256],ennemiPoints);
disp('Creating Model Ennemi : done');
ModelGoalJaune = adaptGauss2([50,235],[50,246],[14,19],GoalJaunePoints);
disp('Creating Model GoalJaune : done');
ModelGoalBleu = adaptGauss2([0,256],[0,256],[0,108,256],goalBleuPoints);
disp('Creating Model GoalBleu : done');
ModelMur = adaptGauss2([0,237,256],[0,238,256],[0,256], MurPoints);
disp('Creating Model Mur : done');
ModelBalle = adaptGauss2([235,246],[0,216,256],[10,50], BallePoints);
disp('Creating Model Balle : done');
ModelTapis = adaptGauss2([0,60,125,256],[0,86,112,142,256],[0,256], TapisPoints);
disp('Creating Model Tapis : done');



Liste(1).Model = ModelBalle;
Liste(2).Model = ModelEnnemi;
Liste(3).Model = ModelGoalJaune;
Liste(4).Model = ModelGoalBleu;
Liste(5).Model = ModelMur;
Liste(6).Model = ModelTapis;
Liste(1).nbrModel = 6
Liste(1).long = length(BallePoints);
Liste(2).long = length(ennemiPoints);
Liste(3).long = length(GoalJaunePoints);
Liste(4).long = length(goalBleuPoints);
Liste(5).long = length(MurPoints);
Liste(6).long = length(TapisPoints);

%Personnal changements :
%
% Liste(1).Model.long = 3e14;
% Liste(2).Model(4).long = 0;
% Liste(2).Model(2).long = 0;
% Liste(2).Model(3).long = 0;
% Liste(2).Model(1).long = 0;
% Liste(3).Model.long = 5e5;
Liste(5).Model(1).long = 1;
```

6.2.2 Learn.m

```
function colorPoints = learn(previousPoints, image)

[x,y,dumb] = size(image);
colorPoints = previousPoints;

for i = 1:x
   if mod(i,50) == 0
      disp(sprintf('%d out of %d',i,x))
   end
   ligne = [];
   for j = 1:y

      col = squeeze(image(i,j,:));
      if (col ~= [255 255 255]')
%          points = size(colorPoints,2);
%          test = sum(colorPoints == col * ones(1,points));
%           if (max(test)<3)
%              colorPoints = [colorPoints col];
%           end
         ligne = [ligne col];
      end

   end
   colorPoints = [colorPoints ligne];
end
```

6.2.3 adaptGauss2.m

```
function Model = adaptGauss2(nr,ng,nb,data)
%
% % Placer les gaussiennes aléatoirement.
%
% % Selection des points aléatoirement au sein des possibilités, un nombre
% % plus ou moins important dépendant du nombre de gaussiennes désirées.
%
% % Attribution des points aux différentes gaussiennes,
% % Calcul des centres de gravité et matrices de covariance,
% % tant que les choses évoluent.
%
% % Attribution des points aux différentes gaussiennes.
% % Suppression des gaussiennes "mortes".

%NOOOON
% ON VA FAIRE CA :

% Selection des points aléatoirement au sein des possibilités, un nombre
% plus ou moins important dépendant du nombre de gaussiennes désirées.
% Tu lui donnes la liste des minimums locaux pour les trois couleurs RGB
% Il améliore potentiellement les valeurs.
% Il sépare les points selon les différentes minimums.. et il construit les
% gaussiennes (probabilités et matrice de covariance).

Npoints = size(data,2);
limitRatio = .13;
index = randperm(size(data,2));
training = data(:,index(1:Npoints));
nbr = 1;

for i = 2:length(nr)
   for j = 2:length(ng)
      for k = 2:length(nb)
         I = find (  training(1,:) >= ones(1,Npoints)*nr(i-1) & ...
                     training(1,:) < ones(1,Npoints)*nr(i) & ...
                     training(2,:) >= ones(1,Npoints)*ng(j-1) & ...
                     training(2,:) < ones(1,Npoints)*ng(j) & ...
                     training(3,:) >= ones(1,Npoints)*nb(k-1) &...
                     training(3,:) < ones(1,Npoints)*nb(k) );
%           disp(sprintf('Size : %d, ratio of
%           %d.',size(I,2),size(I,2)/Npoints))
         if size(I,2)> limitRatio * Npoints;%~isempty(I)%
%                           disp('Accepted')
            Model(nbr).mean = mean(training(:,I),2);
            Model(nbr).cov = cov(training(:,I)')+eye(3);
            Model(nbr).limitR = nr(i-1:i);
            Model(nbr).limitG = ng(j-1:j);
            Model(nbr).limitB = nb(k-1:k);
            Model(nbr).ratio = size(I,2)/Npoints;
            Model(nbr).long = size(I,2);
%              Model(1).tot = Model(1).tot + size(I,2)/Npoints;
            nbr = nbr + 1;
         else
%              disp  ('Refused !')

         end
      end
   end
end

% [Model.ratio] = [Model.ratio] ./ Model(1).tot
```

6.2.4 computeLUT.m

```
% générer la LUT
disp('Building small Look-Up table');
resol = 25;
ref = zeros(resol,resol,resol);
totalPoints = sum([Liste.long]);

for i = 1:resol
   for j = 1:resol
      for k = 1:resol

         if mod(i* resol^2+j*resol+k,100) == 0
            disp(sprintf('R : %d, G : %d, B : %d.',i,j,k))
         end
%
         x = floor(255/resol)*[i;j;k];
         proba = 0;

         for l = 1:Liste(1).nbrModel

            proba(l) = 0;

            for m = 1:size(Liste(l).Model,2)

               ratio = Liste(l).Model(m).long / totalPoints;
%                 if l == 3
%                    disp(ratio)
%                 end
               sigma = Liste(l).Model(m).cov;
               mu = Liste(l).Model(m).mean;

               proba(l) = proba(l) + ...
                        ratio / ((2*pi)^(3/2)* sqrt(det(sigma))) * ...
                        exp(-1/2 * (x-mu)'*inv(sigma)*(x-mu)) ;
%                         if l ==3
%                            disp(proba)
%                         end
            end
         end

         [dumb value] = max(proba);
%          if dumb > 1e-13
            ref(i,j,k) = value;
            val(i,j,k) = dumb;
%          else
%             ref(i,j,k) = 7;
%             val(i,j,k) = dumb;
%          end

      end
   end
end
```

6.2.5 TransferMatrixToFile.m

```
fid = fopen('lut.h','w+');

count = fwrite(fid,sprintf('int lut[25][25][25] = {'),'uchar');
for i = 1: size(ref,1)
   i
   if i==1
      count = fwrite(fid,sprintf('\n  {'),'uchar');
   else
      count = fwrite(fid,sprintf(',\n  {'),'uchar');
   end

   for j = 1 : size(ref,2)

      if j == 1
         count = fwrite(fid,sprintf('\n    {'),'uchar');
      else
         count = fwrite(fid,sprintf(',\n    {'),'uchar');
      end

      for k  = 1 : size(ref,3)

         if k~=1
            count = fwrite(fid,',','uchar');
         end
         count = fprintf(fid,'%d',ref(i,j,k));

      end
      count = fwrite(fid,sprintf('}'),'uchar');

   end
   count = fwrite(fid,sprintf('\n  }'),'uchar');

end
count = fwrite(fid,sprintf('\n};\n'),'uchar');

fclose(fid);
```

6.2.6 traiterImageGaussMixture.m

```
test = [double(imread('pic01(1)','ppm')) double(imread('pic02(1)','ppm'));...
double(imread('pic03(1)','ppm')) double(imread('pic04(1)','ppm'))];% double(imread('pic05','ppm')); ...
% double(imread('pic06','ppm')) double(imread('pic07','ppm')) double(imread('pic09','ppm'))];
%
% test = [double(imread('pic55','ppm')) double(imread('blur55','ppm'))]% double(imread('pic47','ppm')) double(imread('pic02','ppm'));...
% double(imread('pic03','ppm')) double(imread('pic04','ppm')) double(imread('pic05','ppm')); ...
% double(imread('pic06','ppm')) double(imread('pic07','ppm')) double(imread('pic09','ppm'))];

% test = [double(imread('pic001','ppm')) double(imread('test','ppm'))];

[x y dumb] = size(test);

result = zeros(x,y);


figure

subplot(1,2,1)

imagesc(test/255);

%

% for i = 1 : 25

% for j = 1:25

% for k = 1:25

% ref2(i,j,k) = ref(i*10,j*10,k*10);

% end

% end

% end




for i=1:x

%     i

    for j =1:y


%         result(i,j) = ref(max(test(i,j,1),1),max(test(i,j,2),1),max(test(i,j,3),1));

        X = max(floor(test(i,j,1)/10),1);

        Y = max(floor(test(i,j,2)/10),1);

        Z = max(floor(test(i,j,3)/10),1);

        result(i,j) = ref(X,Y,Z);

    end

end


map =[0.9882    0.4235    0.0118
```

```
    0.9892    0.3882    0.0526
    0.9902    0.3529    0.0935
    0.9912    0.3176    0.1343
    0.9922    0.2824    0.1752
    0.9931    0.2471    0.2160
    0.9941    0.2118    0.2569
    0.9951    0.1765    0.2977
    0.9961    0.1412    0.3386
    0.9971    0.1059    0.3794
    0.9980    0.0706    0.4203
    0.9990    0.0353    0.4611
    1.0000         0    0.5020
    0.9994    0.0552    0.5036
    0.9989    0.1104    0.5053
    0.9983    0.1655    0.5070
    0.9978    0.2207    0.5087
    0.9972    0.2759    0.5104
    0.9966    0.3311    0.5120
    0.9961    0.3863    0.5137
    0.9955    0.4415    0.5154
    0.9950    0.4966    0.5171
    0.9944    0.5518    0.5188
    0.9938    0.6070    0.5204
    0.9933    0.6622    0.5221
    0.9927    0.7174    0.5238
    0.9922    0.7725    0.5255
    0.9158    0.7131    0.5237
    0.8395    0.6537    0.5219
    0.7632    0.5943    0.5201
    0.6869    0.5348    0.5183
    0.6106    0.4754    0.5164
    0.5342    0.4160    0.5146
    0.4579    0.3566    0.5128
    0.3816    0.2971    0.5110
    0.3053    0.2377    0.5092
    0.2290    0.1783    0.5074
    0.1526    0.1189    0.5056
    0.0763    0.0594    0.5038
         0         0    0.5020
    0.0833    0.0833    0.5020
    0.1667    0.1667    0.5020
    0.2500    0.2500    0.5020
    0.3333    0.3333    0.5020
    0.4167    0.4167    0.5020
    0.5000    0.5000    0.5020
    0.5833    0.5833    0.5020
    0.6667    0.6667    0.5020
    0.7500    0.7500    0.5020
    0.8333    0.8333    0.5020
    0.9167    0.9167    0.5020
    1.0000    1.0000    0.5020
    0.9167    0.9376    0.4602
    0.8333    0.8752    0.4183
    0.7500    0.8127    0.3765
    0.6667    0.7503    0.3347
    0.5833    0.6879    0.2928
    0.5000    0.6255    0.2510
    0.4167    0.5631    0.2092
    0.3333    0.5007    0.1673
    0.2500    0.4382    0.1255
    0.1667    0.3758    0.0837
    0.0833    0.3134    0.0418
         0    0.2510         0
];


      subplot(1,2,2)

%       imagesc(result);
```

```
%       drawnow

%       colormap(map);

%       figure
            imagesc(result);
      drawnow

      colormap(map);

%       figure
% for i=2:x-1
% %     i
%     for j =2:y-1
%       d = result(i-1:i+1,j-1:j+1);
%       [dumb filt(i,j)] = max(hist(d(:),1:6));
%     end
% end
% imagesc(filt);
%       drawnow
%
%       colormap(map);
```

6.2.7 ppmToCArray.m

```
fid = fopen('cArray.txt','w+');
pic = double(imread('pic03(1)','ppm'))

count = fwrite(fid,sprintf('unsigned char pic[62][82][3] = {'),'uchar');

for i = 1: size(pic,1)
   i
   if i==1
      count = fwrite(fid,sprintf('\n  {'),'uchar');
   else
      count = fwrite(fid,sprintf(',\n  {'),'uchar');
   end

   for j = 1 : size(pic,2)

      if j == 1
         count = fwrite(fid,sprintf('\n    {'),'uchar');
      else
         count = fwrite(fid,sprintf(',\n    {'),'uchar');
      end

      for k  = 1 : size(pic,3)

         if k~=1
            count = fwrite(fid,',','uchar');
         end
         count = fprintf(fid,'%d',pic(i,j,k));

      end
      count = fwrite(fid,sprintf('}'),'uchar');

   end
   count = fwrite(fid,sprintf('\n  }'),'uchar');

end
count = fwrite(fid,sprintf('\n};\n'),'uchar');

fclose(fid);
```