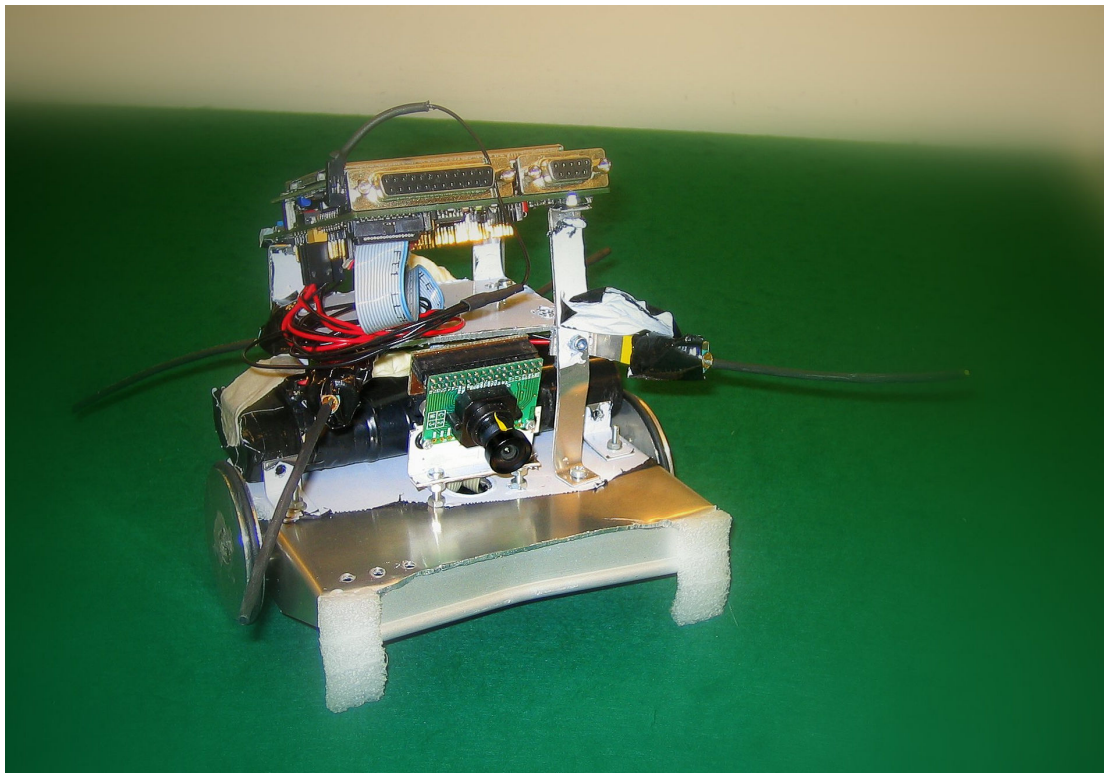


KTH - Royal Institute of technology
2D1426, Robotics and Autonomous Systems

Report of our “Röbi”

Summer 2006



Daniel Bueb kth_robotics@paxi.ch
Han Wu goe@kth.se
Dilhan Balage dilhan@kth.se
Ronney Meier

Abstract

Röbi is a robot which was built in the course 2D1426 of KTH in spring 2006. This report is about the Soft- and Hardware which enabled it to shoot goals on its own and tries to analyze the problems it had while competing against other robots.

Contents

Background.....	4
Hardware.....	4
Design and purpose.....	4
Programming environment and robot communication	5
State Diagram.....	6
Locomotion	7
Navigation.....	7
Image Processing	9
Result	12
Conclusion	12
Reference	13

Background

This is a project report for the course *2D1426, Robotics and Autonomous systems* taken by the authors in the summer 2006 at the *Royal Institute of Technology, KTH*.

The aim of the course is to give theoretical knowledge about fundamental concept of autonomous mobile robots and give hands on experience in building and programming an autonomous mobile robot. The aim of making robot is to fulfill course requirement of the course to make an autonomous mobile robot to play soccer with the one opponent in the field.

The heart of the robot Motorola 68000 family processor and CMOS camera, Motors with encoders, wheels, 3 A4/sized aluminum plates and accessories were given to each group.

This report contains the how we implement the robot in order to fulfill course requirement while preserving constraint given to us.

Hardware

- one Eyebot Controller board M5 (Motorola 68332)
- one color camera of resolution 162*144 with auto-brightness and adjustable frequency.
- two electric motors with built in quadrature encoder.
- two servos
- one 7.5v rechargeable battery
- three aluminum plates

Design and purpose

The field is rectangular, covered by green carpet, it is surrounded by white wall, and 2 goals on opposite side are painted as blue respective yellow. The soccer ball is an red colored golf ball.

Main constraint on the size of the robot is that it must fit within 180mm diameter vertical cylinder all the time including dribbling mechanism. However whiskers can be outside of the cylinder provided that those can not touch the ball.

To be detectable by an opponent, the robot should wear a purple paper slip in at least 4cm height and 15cm in diameter which should be visible from all directions. The robot may not carry any large area confusing-other-robots colors such as red, yellow, blue or green.

The robot should be able to score at least one goal in 2 minutes without a robot opponent on field to qualify for the competition.

Dribble device allowed (A spinning wheel which makes the ball to spin back towards the robot). But the robot performance is good even without a dribbling mechanism. So we did not introduce any dribbling mechanism to robot.

With the conditions above our team made the design in a very early stage: (picture see cover page). Our robot was built using aluminum plates only, since all the other construction parts are red or blue.

The robot was constructed with two differential wheels and on the third contact point to soccer field a rounded head threaded bolt was used. The front of the robot was made slightly “V-shaped” in order to keep the ball in the middle of the robot while it moves with the ball. But at least 75% of the ball was kept outside of the convex shape, as the rule in the competition required.

Servos for tilting the camera were not needed because the camera has a wide angle lens which can cover the whole length of the field.

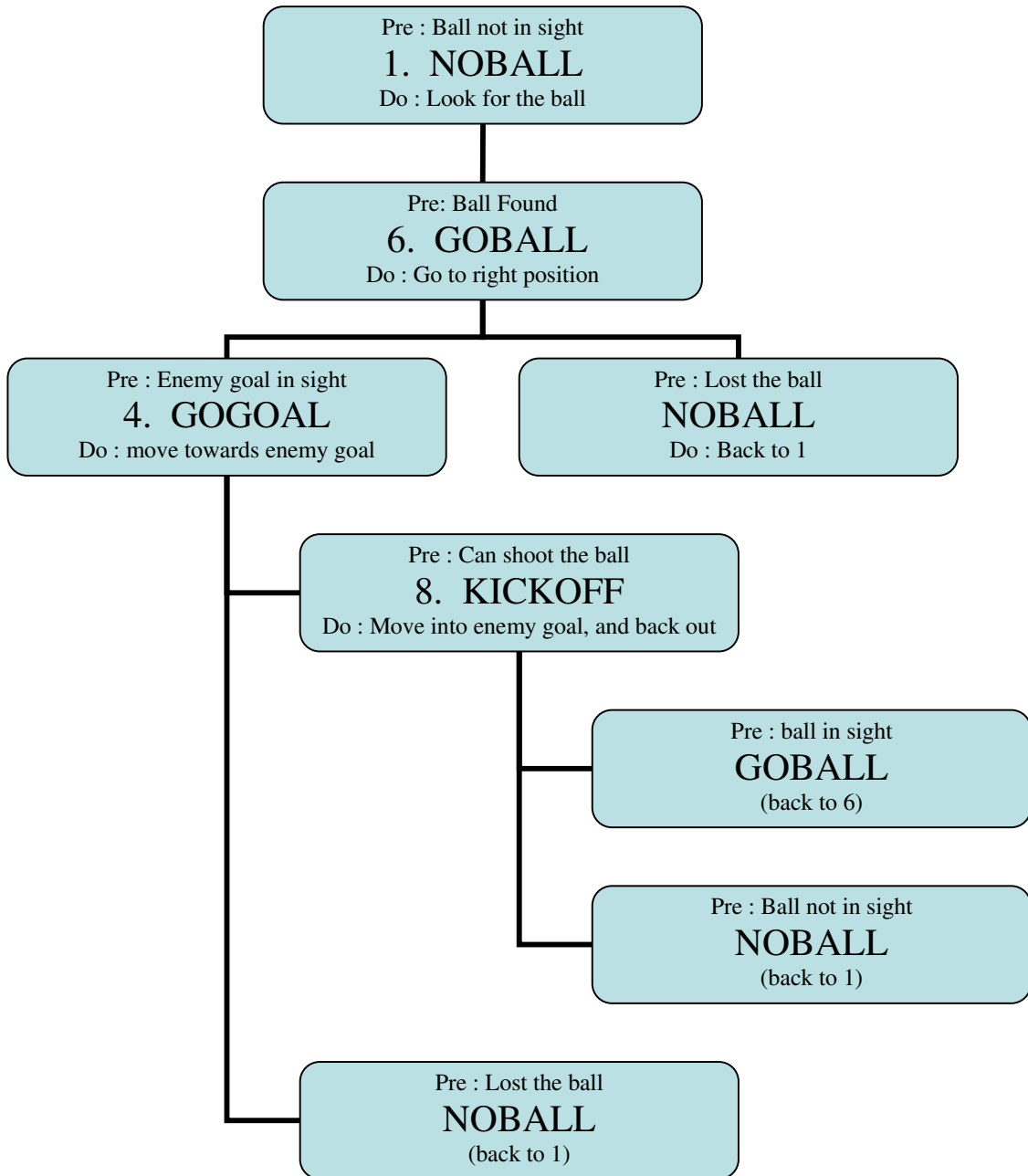
Robot size are made to exact fit into the 18cm circle, thus grants the largest coverage when driving towards the ball.

Programming environment and robot communication

Programming is done in C, linked with [RoBIOS Library functions](#), compiled in Linux with gcc. And the executable file .hex is uploaded to the Eyebot controller board through RS 232 interface. Through the same serial port we can also download data from the Eyebot, which we massively used for image downloading for calculating the proper Hue value.

There is a Radio Communication unit to communicate with Robot. According to rule in the competition, any type of communication with the robot during soccer playing is prohibited.

State Diagram



The program works in states. The state diagram presents how the program flows in each situation.

Sensing

Main sensor of the robot is CMOS camera. There were quadrature encoders and four whiskers. Camera was used to navigate the robot, obstacle avoidance, find ball, to find enemy goal and its own goal, score and safe its own goal. Quadrature encoders are included in the motor/gearbox assembly in order to facilitate dead reckoning positioning and closed loop control system for locomotion. Quadrature encoders are used by V- ω interface to calculate the angle of rotation or distance traveled by the robot. The whiskers are mounted on the robots 45° axes, two facing forward and two facing backward. Whiskers are used to detect obstacles mainly walls of the soccer field and presence of the enemy near by but outside the field of view of camera.

Locomotion

Intended area of the robot motion is soccer field. It is a flat surface, so wheeled locomotion is ideal for flat surfaces. Differential drive wheels were using in the robot. Rounded head threaded bolt was used as a third contact point to the soccer field in order to maintain its stability. V- ω interface is used for drive the differential drive motors.

Navigation

The Navigation was mainly based on the V- ω driver and the camera input. We assumed that the coordinates of the V- ω driver are most of the time more or less correct and used them for calculating the needed paths. Since the robot lost quite fast orientation in space, it checked after each processed image if it sees one of the goals. Then, if there is no enemy in the image, and it is far enough away from the goal it calculates its orientation in space relative to the angular in which the robot sees the goal and the x/y position it thinks it is. This worked quite well (about ± 10 degrees) as long as the x/y position was not completely wrong. Additionally the robot corrected its x/y position when it shot a goal, to being in the goal.

For going behind the ball in the state GOBALL exists 2 different moving algorithms. First a point, which lies on a straight line from the goal to ball behind the ball, is calculated. Then it is checked if it is possible to move on a straight line to this point without touching the ball. If yes, the robot does that. If not, the robot will turn itself into the direction of the ball and move there until a certain distance, and then it will do a circle around the ball, until it is at the previously calculated point. When it arrived at this point it will turn into the direction of the enemy goal, and checks if it has visual contact to the goal and the ball, and if it is able to move the ball on a straight line into the goal. This check is also issued at different other points in the

moving algorithms. As soon this check returns a true, the robot will stop doing whatever it was doing, and will try to make the goal.

One of the problems with this design is that if the enemy robot is in front of its goal, and too near to our robot, our robot will not be able to see the goal, and thus never try to shoot it.

Image Processing

It should be easiest to recognize the ball with the color camera by Hue, Saturation Value identification. Especially when it is not able to turn off the auto-brightness function for the camera.

The task of the image processing part is to explore the captured picture in a short time to save time and thus processor power. In the same time it should be efficient and give back reliable results.

We had different states in our developpement.

The first version was a simple algorithm which explored in the picture every second pixel. A higher resolution is not needed; because the smallest object – the ball – fill several pixels also when having the biggest possible distance.

The object recognition is based on color recognition. So it detected an accumulutiation of the same color and assign to an object.

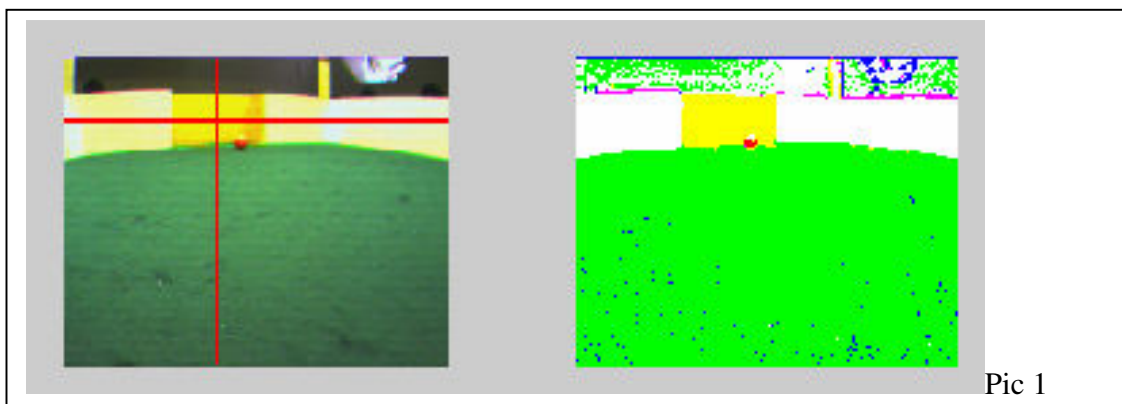
This algorithm worked ok, but was very slow and the color detection caused problem, because it was not adjusted that far. The calibration was done each time we started the robot by setting the white and black point.

And for the color detection we used the standard hue values, which help us a little bit to handle the not that reliable variations from the cam. A problem caused also the auto-calibration of the cam. So the same picture taken at the same position under the nearly same conditions could look not the same at all.

First we tried to improve the algorithm on the robot, doing online correction via the processor keyboard and help of outputs on the display. This was rather annoying, so we decided to download a lot of pictures, declaring all objects and distances in a table.

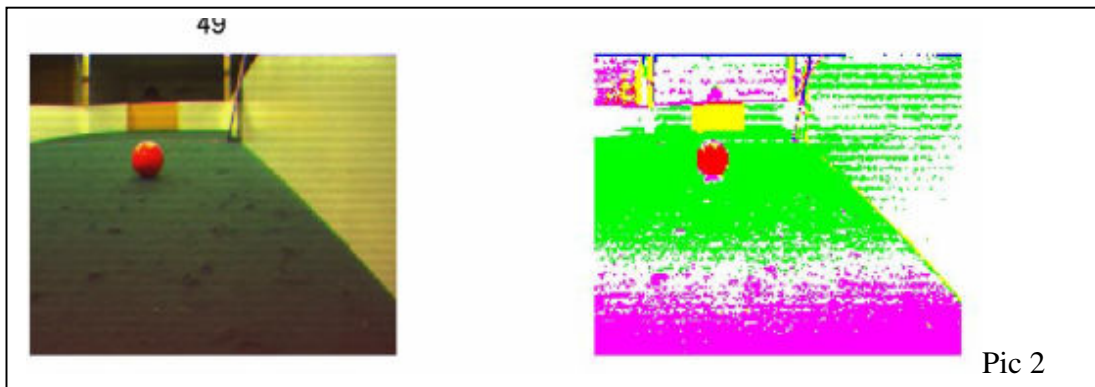
The developing on the second part algorithm was done in Matlab. Of course we did not use the powerful tools of it, because we had to transform back the algorithm. But nevertheless Matlab is a good tool, because it is simple, has a powerful output and is fast and stable.

We like to present and explain some printouts from Matlab:



Picture 1: On the left picture we see the original picture, but with an added red cross, which show where the algorithm detect the center of the goal. This will later on used for calculating the distance to the goal with help of a distance table.

On the right side we see the picture as the robot will detect the colors. We see that this happens with some errors, and the we have to consider that the returned hue values are newer 100% perfect and can contain several faults. (E.g. here blue pixels in the bottom, or an enemy color at the border between the wall and the environment. Things like this have to be handled in the image processing part but not in the hue value calculation part.



Pic 2

Picture 2: Here a picture of a more problematic image.

The worst picture we had was when the whole picture was a kind of reddish. We decided to not throw away the whole image but to search still in the less critical part for a ball. This was probably the one of the two reasons why we had a big problem in the second part of our competition where it detected a ball in the wall. (In reality there was nothing but a white wall)

First we tried to improve the color detection. This was done by stretching the whole hue band and not using a standard one. Moreover similar colors as purple and orange where distinguished better with relation between the color values.

Here a table with the hue values, and what we have after these algorithms.

Color:	blue	green	yellow	orange	purple	white
Average Value:	246	195	128	60	20	6
Variation:	8	41	26	25	--	--

The purple and white are “artificial” hue values, simply because the purple hue value interference with the orange and the white with different colors. So we did a enhanced decision with help of other relations: this will be explained now in the pseudo/ simplified code:

```
// Set values to 1 to prevent from 0 division
// delta = max(r,g,b) - min(r,g,b)
if (6 * delta <= max) hue = 10; // if delta is too small then it is white
```

else

```
if (r == max) hue = 60 + 60*(g-b) / (max-min) /* we used the standard hue
calculation with modified values: We shifted it a little bit higher. We did the same for
green and blue with the adequate numbers.*/
//Moreover we did following corrections:

if ((102<hue)&&(hue<=140)&&(r/b<3.9))
    hue=10;//when relation r/b is quite low, set white instead of yellow
if ((102<hue)&&(hue<=243)&&((r+g+b)>500))
    hue=10;// when the pixel is quite bright, set white instead of green
if ((40<=hue)&&(hue<=80)&&(r/g)<1.2))
    hue=10;//if detected by a low relation of r/g, set white instead of ball,
if ((25<hue)&&(hue<=102)&&(r/b<4))
    hue = 20; // to make a obvious purple instead orange detected by a
    relative low r/b relation
if (hue>250)
    hue=250; // 250 is blue, every hue higher than 250 is limited to the blue
    value because we shifted the whole range to rather high values
if ((102<hue)&&((r+g)/(abs(b-16)+1)) > 50))
    hue = 120; // to fix conflict yellow / red
```

We have to admit here a big drawback: When we developed the second part we decided to do no more the black/ white correction (color-correction). This was obvious a great loss to the precision because the lookup table make loose precision. We did not have this down sampling in Matlab and thus had to fight with some strange behaviors doing processing the image on the robot.

How the image processing part works:

Voilà:

First we have to know that the cam is adjusted like that we can see everything, including the enemy, viewed in relation to the highness. We have placed the cam at a certain highness, simply that it is easier to calculate the distance with help of the distance table related to the number of row for the ball respective for the goals.

We could split the taken image into different parts where we can see certain colors/objects.

So a ball can be only in the lower part of the picture, whereas in really lower, it will be quite huge, thus we do not have to check all pixels, but only every 3rd. This saves us processing power. In the lower-middle/ upper-middle part we have to work more precise and check every 2nd respective every pixel.

With help of the angle (detected by the number of column) and the row we can lookup in a distance table and get a quite accurate distance table.

The upper part of the image is used only for enemy detection.

Finally for detection of the center of the image we used to detect the broadest part: the line where it has most consecutive red pixel.

The detection of the goals: from the principle it is similar one to the ball detection, with the difference, that we just check one line, if, and where (angle \approx column) the goal is. From the middle part of the goal we check up and downward the boarder to the ground or

the unknown “outside”. This is done also in checking if there are within 3 pixel at least 2 consecutive of the same color in the upper or downer way. For the distance to the goal, we observed that the down boarder (number of row) is more reliable for huge distances, whereas the upper border is more reliable for the near distances.

Concluding of this part:

I would say that this part of the project was very interesting and I had learned a lot. Especially that it does not help to know only the theory, but that it is also a big task to apply it on real-world data with time-sensitive needs! Moreover never do last minute changes, when you know that it could causes problem. We did it by deciding to explore the picture from down to bottom, but no more bottoms up. This changement sounds simple, but because of rather tricky relation in the algorithm it was easy to oversee something. This was probably our 2nd big problem in the final soccer game!

Result

Röbi scored 2 goals with no opponent on field. But could not stand against an enemy due to some errors in recognize the goal color.

Conclusion

The project was fun. But we realized too late that we actually need a good plan in forward to achieve a better representation. A good teamwork is very much needed for such sized mission. The programming was messy without a clear defined responsible area for each of members.

Reference

1. M. Bratt. Project lecture notes. For course 2D1426 Robotics and Autonomous Systems, KTH, 2006.
2. H. I. Christensen. Lecture notes. For course 2D1426 Robotics and Autonomous Systems, KTH, 2006.
3. T. Bräunl. <http://robotics.ee.uwa.edu.au/eyebot/>. EyeBot - Online documentation.
4. B. Graf, T. Bräunl, and M. Bratt. Example source code. Supplied by course leader.
5. R. Siegwart, I. R. Nourbakhsh: Introduction to Autonomous Mobile Robots, MIT Press 2004, ISBN 0-262-19502-X
6. Soccer rules from course homepage:
<http://www.nada.kth.se/kurser/kth/2D1426/aktuellt.html>