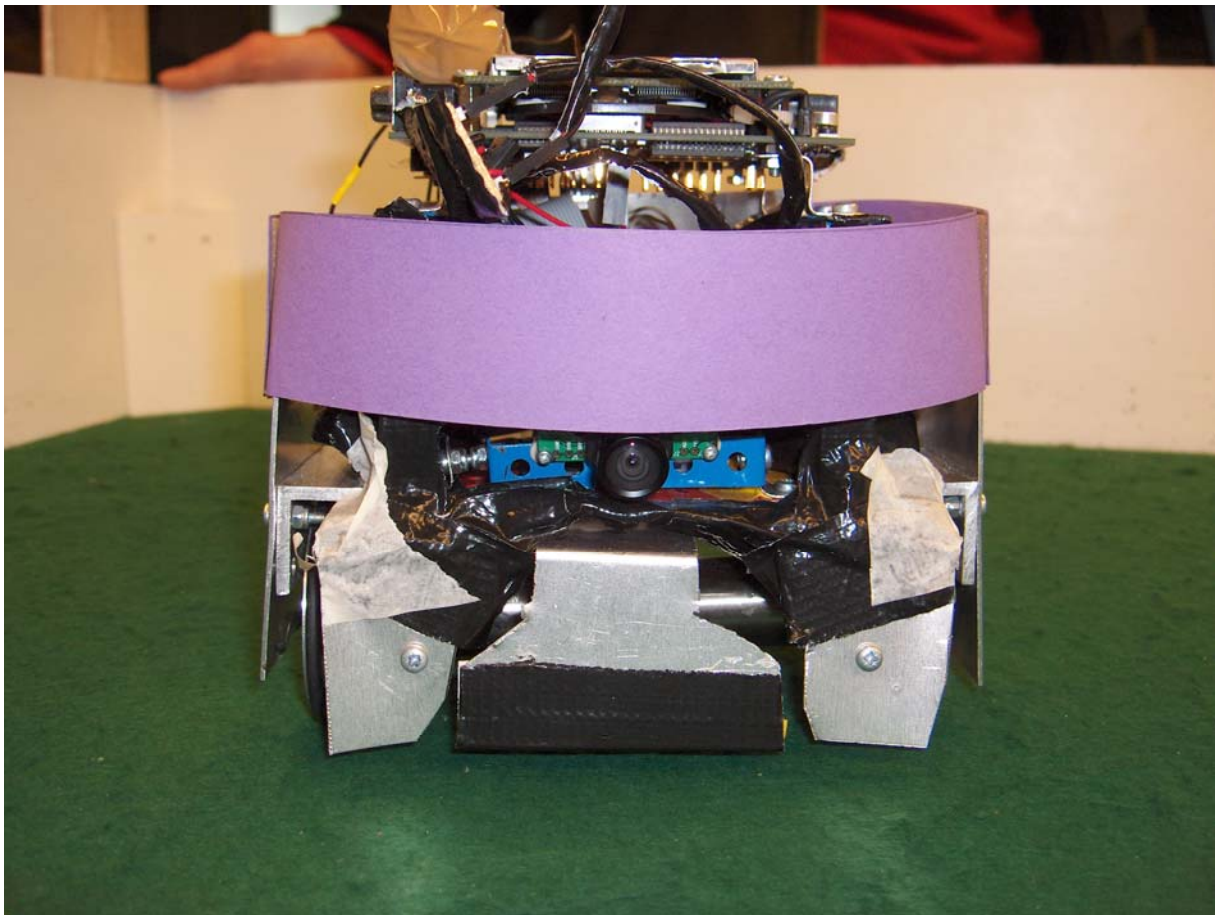


KTH - Royal Institute of Technology
NADA
2D1426 Robotics and Autonomous Systems
Spring 2006 for Mattias Bratt

Chuck

Project report



Andreas Berglund - anbe02@nada.kth.se
Stefan Dagnell - stda02@nada.kth.se
Johan Risén - jori02@nada.kth.se
Ragnar Rova - raro02@nada.kth.se

| | |
|--|----|
| Introduction | 3 |
| Soccer | 3 |
| Team members | 3 |
| Robot | 4 |
| Parts list | 4 |
| Controller board | 4 |
| Construction | 5 |
| Vision | 6 |
| SVM | 6 |
| Image processing | 7 |
| Camera | 7 |
| Movement | 8 |
| Path planning | 8 |
| Localization | 9 |
| Sensors | 9 |
| Shooting mechanism | 9 |
| Electrical chart | 10 |
| Behavior & strategy | 10 |
| Software – programming | 11 |
| Results | 11 |
| Conclusion | 12 |
| Sources of information | 12 |
| Appendix A (flow chart) | 13 |
| Appendix B (simplified call graph) | 14 |
| Appendix C (code) | 14 |
| Findobject.h | 14 |
| Findobject.c | 15 |
| Testfindangles.c | 20 |
| Movement_new.h | 21 |
| Movement_new.c | 22 |
| Findangles_new.h | 26 |
| Findangles_new.c | 27 |
| Changed parts of helper.c | 30 |
| Classlut.c | 30 |
| Distlut.c | 31 |

Introduction

This is the report from the project made in the course 2D1426, “Robotics and autonomous systems” at the Royal Institute of Technology, Stockholm, 2006. The project goal is to build a mobile robot with features that enables the robot to play soccer. A brief introduction about the soccer competition and the soccer rules is explained below.

Soccer

The rules for the soccer tournament can be found at <http://www.nada.kth.se/kurser/kth/2D1426/slides2006/proj2.pdf>, but here is a short introduction to the rules. The soccer tournament is played on a field, 120x240 centimeters with a green carpet. The field has two goals, one blue, and one yellow goal. The different colors are to be able to detect your own goal as well as your opponents. The field also has a white centerline. The most important rules are:

1. The robot must fit in a 180 mm circle.
2. Dangerous play is not allowed.
3. All robots must carry a colour marker in the form of a 4 cm high purple paper cylinder visible from all directions.
4. All forms of communication with the robot are forbidden.
5. Holding the ball is not allowed
6. A robot may be taken off the field for a minimum of 30 seconds for repair.
7. A match is three, four or five minutes long.



Figure 1 - Soccer field

Team members

The members in the team that built the robot, and authors of this report are: Andreas Berglund, Stefan Dagnell, Johan Risén and Ragnar Rova. All team members study computer science at KTH.



Figure 2 - The glorious team

Robot

As a start every group got a set of parts to build the robot. The most significant parts are the controller board, the motors and wheels, a battery pack and a charger, and a servo. However, we never used the servo. The group also got aluminum plates, tools and other parts like screws and nuts. During the construction of the robot a chuck key was missing to an electrical drill, hence the name of the robot became Chuck.

Parts list

- Lots of Meccano parts
- Aluminium plates
- 2 motors (Faulhaber/minimotor 2224006SR)
- 1 battery (7.2 V)
- 2 wheels
- 1 camera (color camera, 176x144 pixels)
- 1 microcontroller (see below)
- 1 Shooting device (from an electrical put cup, see section shooting mechanism)
- 1 BD139 NPN transistor (for the shooting device)
- 1 47 Ω resistor (for the shooting device)
- 2 1.5V battery (for the shooting device)

Controller board

The controller board is a 35MHz, 32-bit Motorola Microcontroller called Eyebot with an operating system called Robios. The microcontroller has 2MB RAM, digital camera interface and digital and analog I/O pins. The programming language is C.

Construction

The “hardware” of the robot, which is the mechanical parts and the main platform was built without a detailed plan.

We decided right away to use a 3 wheel system with differential drive. At first the rear wheel was a castor wheel but we realized that the degree of freedom with that kind of wheel would confuse the trajectory controller and switched to a bolt with a smooth head turned downwards.

Being a team altogether consisting of computer science students we did not quite relish the fact that the robot somehow had to be built. We are not what one can call very “practical”. This had some impact on our choice of building material, we quickly seized some good meccano parts. Using meccano for the main platform instead of aluminum was in hindsight a great choice although at times debated. Our complete lack of planning in our building made meccano ideal, moving things around on the platform is much easier since you do not have to drill new holes each time.

To save time on construction we decided to construct the final robot-platform right from the start. We do not regret this intention however, we had to almost completely rebuild the robot a few times. The first time was when we decided to use a mechanism for shooting, we had to make room for it. The second time was when we had already built the robot and realized it did not comply to the rules regarding size. We also had a few minor setbacks, like fastening the shooting mechanism (tedious work) only to realize we put it backwards or that there was not any space for the camera. We were actually very close to abandoning the idea of the shooting-mechanism altogether after spending several days trying to get it in place.

The use of brute force and hole making for parts like the shooting-mechanism rendered our platform increasingly ragged. In the end, the main platform was barely holding together and was very skewed. Other problems was that the shooting-mechanism was nowhere near the middle of the robot and it was not directed straight forward. Our wheel axis was not straight in relation to the platform, on top of this the robot became rather knock-kneed.

Despite all these problems we soon realized that the robot served our purposes just fine. A skewed and instable platform posed very little problem in the end and the use of brute force made our shooting-mechanism shoot straight.

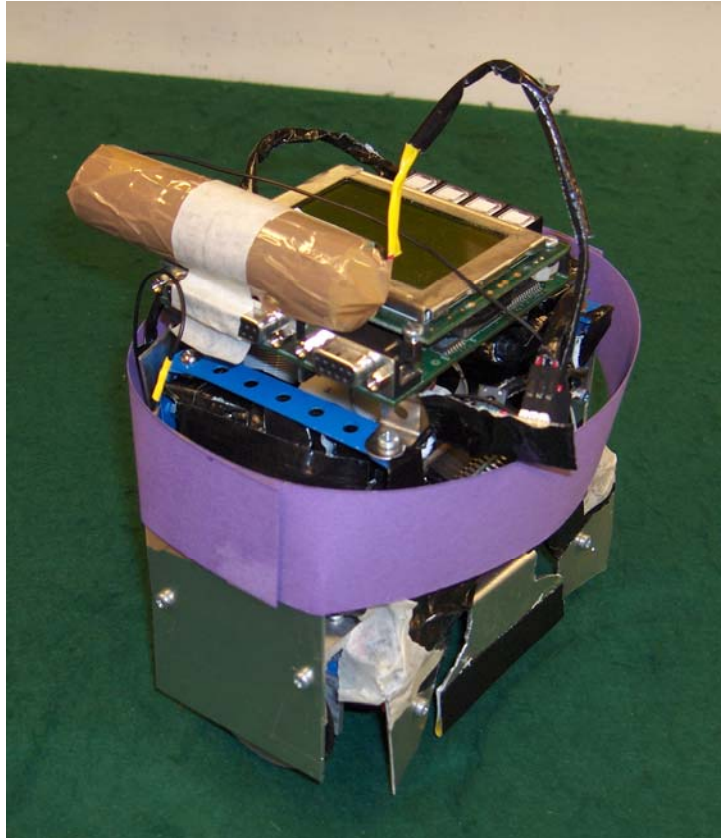


Figure 3 - Chuck profile shot

Vision

The first attempt only used the hue values¹ and an interval for each class. Due to noise in the images and the odd behavior of the camera in different light conditions it was very hard find intervals that were good enough. Either they overlapped or they were too narrow. Because of these difficulties we used the RGB values instead. Since the parts of the field had very different colors we thought that this would give a much better result. Using RGB intervals proved to be very hard because of them not being light invariant.

SVM

To classify the pixels we used a lookup table from RGB value to class (blue goal, yellow goal, ball, field, wall, and opponent). To create this table and get a good generalization we created an SVM (support vector machine²) with the r, b and g-values as input and class as output. To create enough samples, lots of pictures with different illumination was taken. After this, we opened the images in a custom made C# application where we could assign a class to the different pixels by clicking on them. These samples were then exported to a format that we imported to Matlab. We trained the net with approximately 6000 samples. When the net was created we used every possible RGB combination as input to the net and stored the output in a lookup table.

¹ <http://en.wikipedia.org/wiki/Hue>

² http://en.wikipedia.org/wiki/Support_vector_machine

Image processing

To locate for example the ball in the image we created a bitmap with the same size as the image from the camera. We then stored a 1 at every position where the pixel got classified as the current class. To get rid of noise we only used the pixels that were surrounded (4 connectivity) by pixels belonging to the same class. Because of this the robot was unable to see the ball at a distance greater than about half the field length, but it got rid of almost all the noise. We also used a threshold, based on the number pixels belonging to the class, for the goals and ball to handle noise even better.

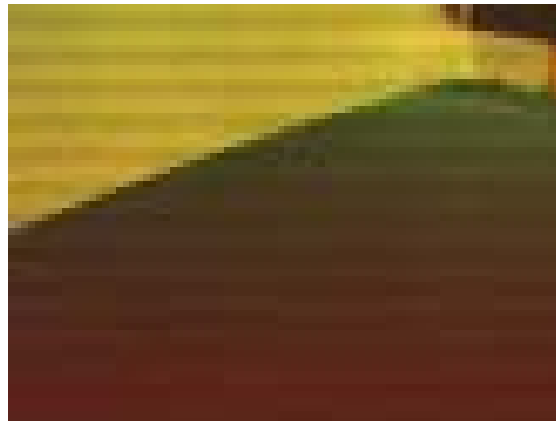
To get the position of the ball in the image we used the mean of the x and y position of the pixels belonging to the class. To get the angle to the ball/goal we divided the FOV of the camera by the x resolution and multiplied it with the x position in the image. A big improvement would have been to get the angle to left and right edge of the goal instead of just an angle. Because we did not do this we got different angles to the goal depending on which part of the goal the camera was pointing at. To get the distance to the ball we created a distance lookup table that mapped y position in the image to decimeters.

We also implemented an algorithm to ignore the pixels above the wall. The pixels above the wall had a tendency to be the same color as the ball or one of the goals. This could also have been solved by lowering the camera.

The class lookup table had a resolution of 32 for each color channel. This was enough to get good results and used little memory. The total size of the table was $32^3 = 32768$ bytes. By storing multiple classes in each byte we could have achieved a smaller table at the expense of a more complex lookup. Because of the good results this was not implemented in the final system.

Camera

Early in the project we discovered that we got strange images from the camera quite often. The most annoying ones were very red near the bottom of the image. This made the robot see lots of balls when there were non to be found. The odd thing was that we always got red images in certain positions and angles on the field. Because these images did not get any better if we just stood there, we drove a few centimetres backwards and rotated a bit when we this happened. We identified these images by looking at the bottom left and right pixels. If both were classified as balls, which is impossible due to the size of the ball, we knew that we had a red image.



In the beginning we also got lots of blueish, broken images. This was solved by lowering the FPS of the camera.

The first vision system only used the sub sampled (82x62 pixels) images. This was very fast, but the resolution was too bad to see the difference between the ball pixels and the noise. Because of this we later used the full (176x144 pixels) images which really improved the results. The disadvantage with this solution is that the image processing became quite slow.

Movement

For trajectory control we used the included VWDrive interface. At one of the project lectures it was mentioned that we might waste a lot of processing power with this design choice but after investigating the complexity of making our own routines we quickly settled for the simple solution in this area. This decision paid off greatly since we watched other robots struggle with trajectory control problems for a very long time.

Also we never used a program design where different behaviors were active simultaneously (Sensing and trajectory at the same time). We tried driving and vision simultaneously by just issuing a VWDrive command and not waiting for its completion, but the images we then grabbed from the camera were sometimes garbled since the VWDrive code was stealing time from the camera buffer update. This was of course also our biggest drawback.

Since the robots mechanical platform constantly was being rebuilt it was hard to test the driving routines in the beginning. After changing the wheel distance in the HDT we wrote a small VWDrive-parameter testing application which was set up to drive in a square and let us alter the parameters without recompilation. We used parameters from a robot from a previous year as a starting point. In the beginning before the shooting mechanism was mounted and the wheel configuration was pretty straightforward it was no problem to get it calibrated. Later on we got into problems because the mechanical parameters had changed, the robot had gotten much heavier which made it refuse to turn correctly so we increased the P value for the W (angle) control. Once we had the problem that the turn command would always turn 5% less than its parameter. Since we felt we had tried everything, the wheel distance was exactly measured, all VW-parameters had been tried, we used the ugly kludge to add 5% to all internally calculated angles to make it turn correctly. Since this just felt plain wrong we retried all settings and changed the wheel distance just slightly (but not to the measured value) and everything was working again.

Path planning

For path planning we used extremely simple angle calculations, not even trigonometry was needed. The two important measures was the angle between when the robot had seen the ball and the angle at which it had seen the goal. Based on the ball angle we drove towards the ball and used the distance lookup table to determine if we are close. Then we started scanning for the goal (if we saw the goal while scanning for the ball we turned towards it as a starting position for the scan). Based on the angle between goal and ball we calculated a circular curve which would position the robot suitable for scoring. If the angle between the goal and ball was small enough (approx 4 degrees) we decided to shoot.

This behavior of finding ball, driving close to the ball, finding goal, reposition for shoot would be iterated over and over again until we had a good scoring position. This made the robot pretty picky about its position relative to ball and goal and it would often make ineffective small adjustments due to that driving and sensing was inaccurate and that the driving instructions planned also were inexact. But eventually after much iteration the robot would score. This iterative pattern of repositioning was pretty slow but made the robot very robust. It could score from any position where to ball was not too close to the wall given enough time.

Localization

The main strategy that the group tried to implement during the whole project was to keep the behavior as simple (not necessarily easy) as possible. The group decided not to implement an absolute positioning model. The robot has no opinion of where it is and can as such never get lost.

The fact that it never gets lost made robot kidnapping and hitting obstacles a non issue and this was a huge advantage. We would often see other robots getting lost and having to run some code to relocalize and each and every time we cherished the moment we decided against it.

As one can imagine we discovered some disadvantages of this approach. As the robot does not save information from earlier scans as it change position it has to scan for the ball and goal more often then a robot which already have an opinion where everything is. To speed up the robot we used several tricks regarding where to start scanning to speed the process.

In conclusion, this approach made our robot slow due to excessive scanning but reliable.

Sensors

We designed the robot with the camera and wheel sensors in mind, planning to add additional sensors as we felt the need for it in the programming and testing phase. In the end we did not feel the need for any additional sensors like bumpers for close range obstacle detection.

While considering using bumpers for obstacle detection we first tried to use the sensors we already had relevant for the task, like stalling detection in the VW drive. After some considerable experiments these proved unreliable and we had little, however some use of it in the final code. Stalling occurs when the wheels can not spin due to the robot being stuck. However if the robot is not heavy enough the wheels may spin despite the robot being stuck so it was hard to use this method of obstacle detection. Cases like partially stuck and stuck in turning was also impossible to detect using stalling data only.

Ignoring obstacle detection was not a hard choice, we had enough work with programming a good behavior for the robot in more commonly occurring cases. Since we had decided against using localization, hitting obstacles did not affect any such part of our program.

The main problem we did have while excluding obstacle avoidance was when the ball was close to the wall. Since we do not detect walls the robot is helpless when the ball is too close to a wall, but this seldom happened.

Shooting mechanism

At an early stage we chose between a roller or shooter for ball-handling, having both would not only redundant but hopelessly difficult to implement. Seeing how nobody had or planned to make an effective shooter we quickly decided to have one.

The ball in the soccer competition is a golf ball and that is how the idea of the shooting device came to life. The device was taken from a golf putting accessory (see picture). If the golfer hits the hole the device automatically shoots the ball back to the golfer.

We decided to have a shooter mostly due to the fun-factor of it. There were some reservations for the actual usability of the shooter at first and we had the option of having two different behaviors in mind, one including shooting and the other not.

The shooter however proved to be very efficient. Chuck are able to shoot at least five meters, or more important, the total length of the soccer field. The precision of the shooter was also very satisfactory, even at an early stage it more often then not hit goal due to the large goal. Even though the shooter was skewed it did not take much effort to position the robot to hit the middle of the goal each and every time (given correct sensor data). In the end this made us completely rely on the shooter for scoring. Since we chose not to use a roller for handling the ball, shooting it proved to be a safer and faster way then driving with it to score.



Figure 4 - Golf putting trainer

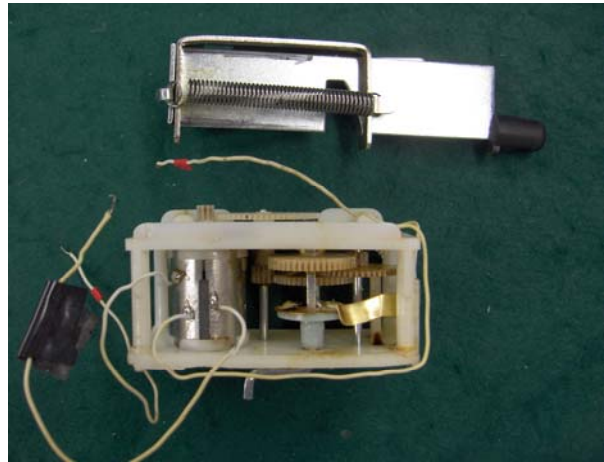


Figure 5 - Putting trainer shootback engine

Electrical chart

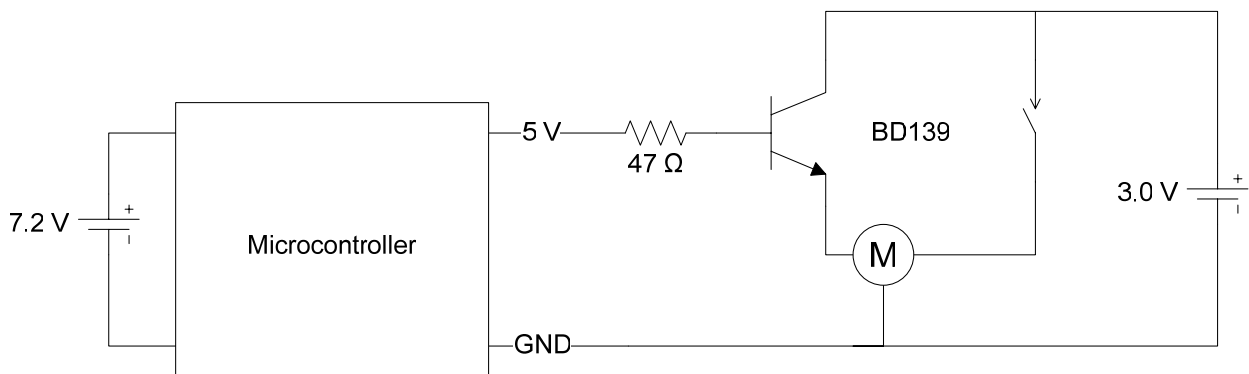


Figure 6 - Shooter activation circuit

Behavior & strategy

The main strategy that the group tried to implement during the whole project was to keep the behavior as simple (not necessarily easy) as possible. The main strategy is

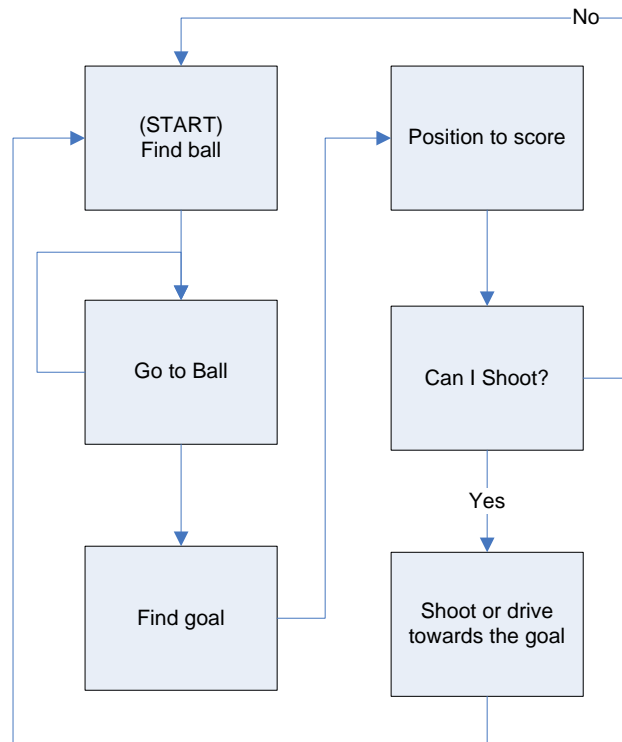


Figure 7 - Basic program flow

More specified flow-chart can be found in appendix A.

The first step is to find the ball. If the robot turns 360 degrees without finding the ball something has to be done. Two reasons why the robot does not find the ball are that the opponent covers the ball, or the ball is too far away to be seen with the camera. So, if the robot does not find the ball, it will drive a predefined length towards the goal farthest away, and then do a new search. When it finds the ball it will drive towards the ball, and do that until the position is close to the ball. Then the search for the opponent goal commences. When it finds the goal the robot makes a curve to position itself for a possible shot. After that a control of the angle between the goal and the ball is made. If it is small, otherwise the robot shoots, it makes a new search for the ball and reposition. After a shot, a new search for the ball starts and it all repeats itself.

Software – programming

We used subversion for source code revision control and gnu make and gcc m68k cross compiler for building. The work was roughly divided into vision code and trajectory code between different people.

Results

Chuck made the competition, by making two goals within the time limit of two minutes. In the group play Chuck played two matches, one win and one draw (the robot S.O.B). Chuck did not make it to the final group play due to a higher goal rate from S.O.B. within the group. All team members are satisfied with the performance of the robot.

Conclusion

To characterize the robot – which was obvious from the tournament results – we built a slow but accurate and robust one. We would not fare well against a quicker adversary due to our slow sense-plan-act cycle but could in return the robot would never get lost.

Our vision code was probably the best and most thoroughly tested part of our program and in the end misclassifications were extremely rare. Spending a lot of time and effort on the vision code was in our case a necessity, since faulty vision interpretations would make our slow robot unbearably slow. Seeing how other robots were distracted by fake ball readings in their sensor interpretations during the competition we are very satisfied we spent so much time on it.

For the mapping, localization and behavior of the robot we early on decided on a simple design, making it possible to spend much time on our vision and planning code. We used no map to speak of and only localization relative to the ball and goal. The only sensors we used were the camera and quadrature encoders of the motors. We also used the ready-made VW-drive for trajectory control.

Doing this we could focus on optimizing our code and testing it thoroughly. We spent much time tweaking minor parameters to achieve the best possible performance our design would permit. In hindsight we are satisfied with this approach, we did not have to give up any code or ideas and our time was used effectively.

To win the competition however, a more complex design would probably be necessary. As the goal of the project was not as much to win the competition as to have a working robot we were satisfied.

Choosing a mechanism for ball-handling we considered a roller as well as a shooting mechanism. Seeing how nobody would or had used a shooter we quickly decided to use one. The shooter was probably not as good as a roller for ball-handling however somewhat faster in scoring, our choice however was based purely on the entertainment value of it. The optimal design would probably be to combine a roller to hold the ball and a shooter. This would allow the robot to keep the ball under control when searching for the goal. This is a quite challenging design problem to solve though.

Excluding a roller and localization with world-coordinates in our design were not only positive. The tedious iterative repositioning to get a good shot as well as our inability to handle a ball close to the wall were weaknesses we could not deal with.

Sources of information

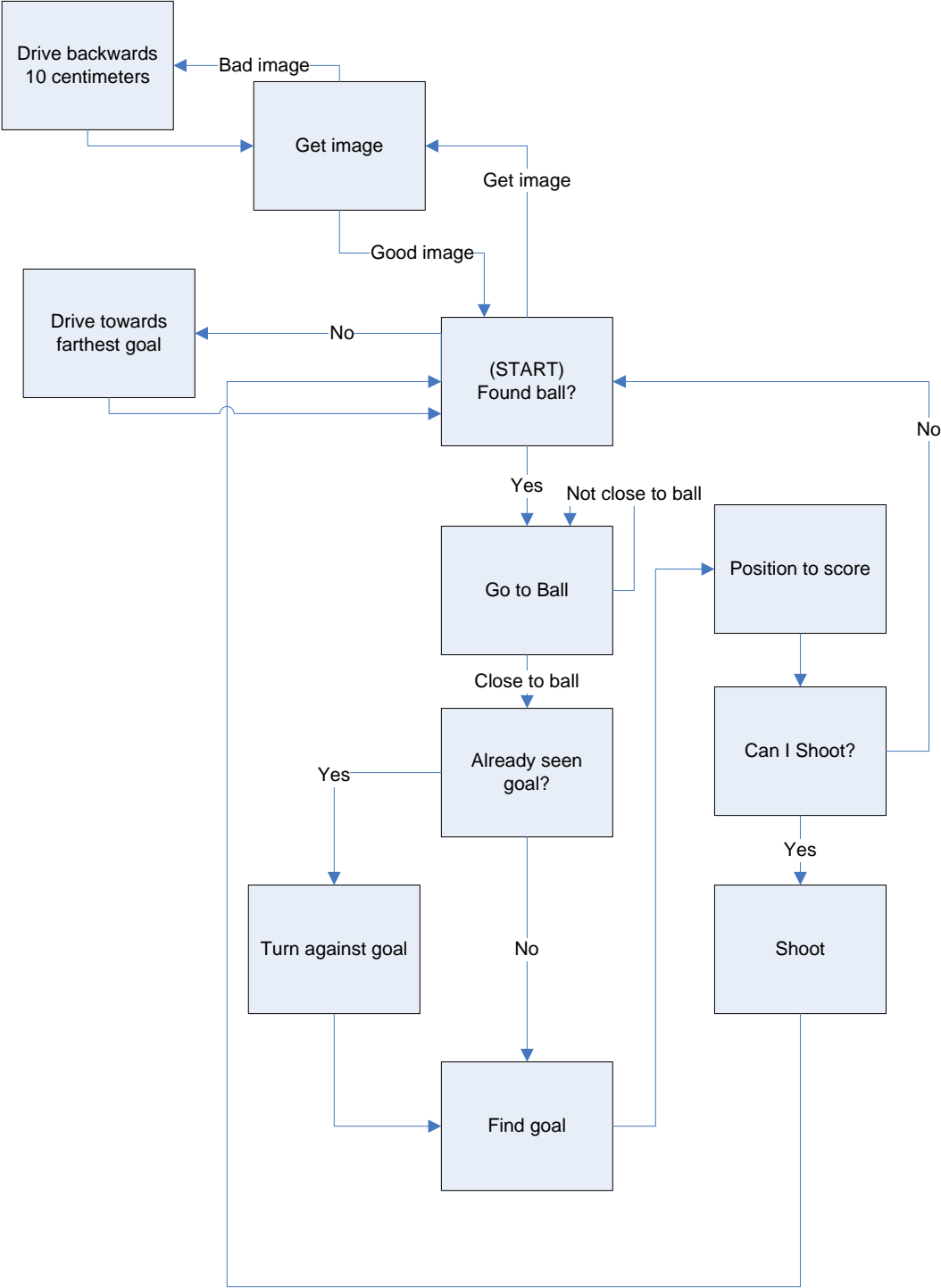
The following main sources of information were used during the project .

[1] R. Siegwart, I. R. Nourbakhsh: Introduction to Autonomous Mobile Robots, MIT Press 2004, ISBN 0-262-19502-X

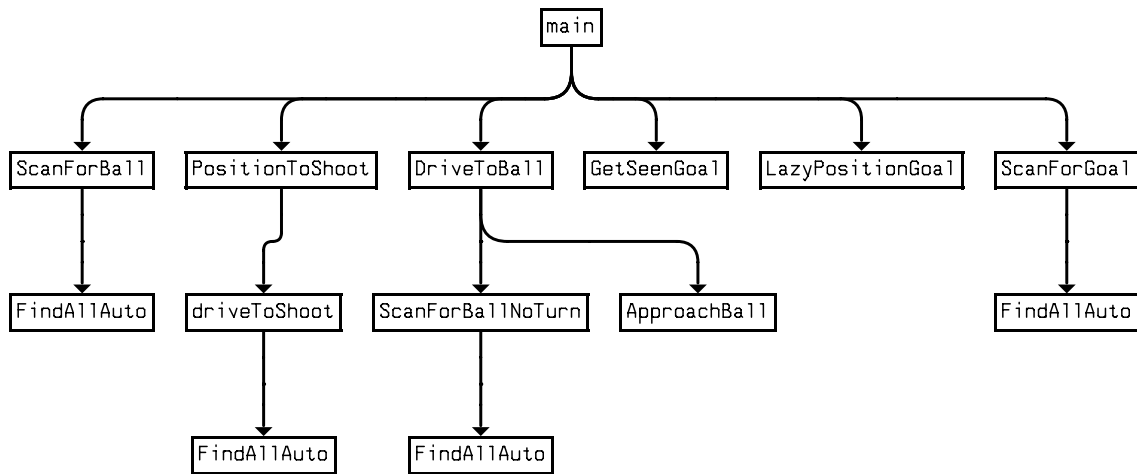
[2] Bräunl, Embedded Robotics, Springer Verlag

[3] Eyebot Online documentation - <http://robotics.ee.uwa.edu.au/eyebot/> [Fetched May 2006]

Appendix A (flow chart)



Appendix B (simplified call graph)



Appendix C (code)

Findobject.h

```
#ifndef __HEADER_CHUCK_FINDOBJECT
#define __HEADER_CHUCK_FINDOBJECT

#include <types.h>
#include <string.h>
#include <chuck/huedefs.h>
#include <chuck/classlut.h>

#define in_interval(x,low,high) ((x) >= (low) && (x) <= (high))

#ifdef IMAGE_TYPE_BIG
#define IMAGE_COLUMNS 176
#define IMAGE_ROWS 144
typedef BYTE _bigcolimage[IMAGE_ROWS][IMAGE_COLUMNS][3];
typedef BYTE _bigimage[IMAGE_ROWS][IMAGE_COLUMNS];
#define ColorImage _bigcolimage
#define GrayImage _bigimage
#define CUT_LOW 24
#else
#define IMAGE_COLUMNS imagecolumns
#define IMAGE_ROWS imagerows
#define ColorImage colimage
#define GrayImage image
#define CUT_LOW 12
#endif

#define GOAL_MAX_ROW (IMAGE_ROWS/2)

typedef struct _point {
    short x;
    short y;
    short pixels;
} Point;

#define FIND_PARAM_CLEANUP_GOAL 0x1
#define FIND_PARAM_GOAL_THRESH_15 0x2
#define FIND_PARAM_GOAL_THRESH_10 0x4
#define FIND_PARAM_CLEANUP_BALL 0x8
#define FIND_PARAM_CUT_ABOVE_WALL 0x10
#define FIND_PARAM_BALL_THRESH 0x20
```

```

#define FIND_PARAM_GOAL_THRESH      0x40

int GetColorImage(ColorImage *img);

void FindObject(image img, int hue_low, int hue_high, Point* posp);
goallpos, Point* goal2pos);
int FindAll(ColorImage colimg, GrayImage hueimg, Point* ballpos, Point* goallpos, Point*
goal2pos, short params);
int FindAllAuto(Point* ballpos, Point* goallpos, Point* goal2pos);
short params);
void CleanupClassImage(GrayImage *img, GrayImage *outimg, unsigned short threshold);
void CutPixelsAboveWall(ColorImage *colimg);

#endif

```

Findobject.c

```

#include <chuck/findobject.h>

int FindAllAuto(Point* ballpos, Point* goallpos, Point* goal2pos) {
    ColorImage img;
    GrayImage hueimg;
    GetColorImage(&img);
    return FindAll(img, hueimg, ballpos, goallpos, goal2pos, FIND_PARAM_CLEANUP_GOAL |
FIND_PARAM_GOAL_THRESH | FIND_PARAM_CLEANUP_BALL | FIND_PARAM_CUT_ABOVE_WALL |
FIND_PARAM_BALL_THRESH);
}

int GetColorImage(ColorImage *img) {
#ifdef IMAGE_TYPE_BIG
    CAMGetFrameRGB((BYTE *)img);
#else
    get_subsampled_colimage(img);
#endif
    return 0;
}

void FindObject(image img, int hue_low, int hue_high, Point* posp) {
    short rowmax = 0,
        colmax = 0,
        maxrow = -1,
        maxcol = -1;

    short rowsum = 0;
    short colcount[imagecolumns];
    memset(&colcount, 0, sizeof(short)*imagecolumns);

    short row, col;
    for(row = 0; row < imagerows; row++) {
        rowsum = 0;

        for(col = 0; col < imagecolumns; col++) {
            if(in_interval(img[row][col], hue_low, hue_high)) {
                /* make sure we have a neighboring pixel too */
                if((col > 0 && in_interval(img[row][col-1], hue_low, hue_high)) ||
                    (col < imagecolumns-1 && in_interval(img[row][col+1], hue_low, hue_high)))
                {
                    rowsum++;
                }
                if((row > 0 && in_interval(img[row-1][col], hue_low, hue_high)) ||
                    (row < imagerows-1 && in_interval(img[row+1][col], hue_low, hue_high)))
                {
                    colcount[col]++;
                }
            }
        }

        /* new max? */
        if(rowsum > rowmax) {
            rowmax = rowsum;
            maxrow = row;
        }
    }

    /* find max x */
    for(col = 0; col < imagecolumns; col++) {

```

```

        if(colcount[col] > colmax) {
            colmax = colcount[col];
            maxcol = col;
        }
    }

    posp->x = maxcol;
    posp->y = maxrow;

    return;
}

int FindAll(ColorImage colimg, GrayImage hueimg, Point* ballpos, Point* goallpos, Point*
goal2pos, short params) {
    // init positions
    goallpos->pixels = 0;
    goal2pos->pixels = 0;
    ballpos->pixels = 0;
    goallpos->y = -1;
    goal2pos->y = -1;
    ballpos->y = -1;
    goallpos->x = -1;
    goal2pos->x = -1;
    ballpos->x = -1;

    short row, col;

    // try to find out if we have a weird image
    unsigned char classl1 = rgb2class(colimg[IMAGE_ROWS-1][0][0],colimg[IMAGE_ROWS-
1][0][1],colimg[IMAGE_ROWS-1][0][2]);
    unsigned char classl2 = rgb2class(colimg[IMAGE_ROWS-1][1][0],colimg[IMAGE_ROWS-
1][1][1],colimg[IMAGE_ROWS-1][1][2]);
    unsigned char classl3 = rgb2class(colimg[IMAGE_ROWS-1][2][0],colimg[IMAGE_ROWS-
1][2][1],colimg[IMAGE_ROWS-1][2][2]);
    unsigned char classr1 = rgb2class(colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-
1][0],colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-1][1],colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-
1][2]);
    unsigned char classr2 = rgb2class(colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-
2][0],colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-2][1],colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-2][2]);
    unsigned char classr3 = rgb2class(colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-
3][0],colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-3][1],colimg[IMAGE_ROWS-1][IMAGE_COLUMNS-3][2]);

    unsigned short lcount = (classl1 == CLASS_BALL ? 1 : 0)+(classl2 == CLASS_BALL ? 1 :
0)+(classl3 == CLASS_BALL ? 1 : 0);
    unsigned short rcount = (classr1 == CLASS_BALL ? 1 : 0)+(classr2 == CLASS_BALL ? 1 :
0)+(classr3 == CLASS_BALL ? 1 : 0);

    if(lcount >= 2 && rcount >= 2)
    {
        return -1;
    }

    //short colcount[3];
    ColorImage colimgfinal;
    for(col = 0; col < IMAGE_COLUMNS; col++) {
        for(row = 0; row < IMAGE_ROWS; row++) {
            colimgfinal[row][col][0] = colimg[row][col][0];
            colimgfinal[row][col][1] = colimg[row][col][1];
            colimgfinal[row][col][2] = colimg[row][col][2];
        }
    }

    // set pixels above the wall to black
    if((params & FIND_PARAM_CUT_ABOVE_WALL) != 0) {
        CutPixelsAboveWall(&colimgfinal);
    }

    // images where pixels belonging to goall or goal2 = 255 or 0 else
    GrayImage goallimg, goal2img, ballimg;

    for(col = 0; col < IMAGE_COLUMNS; col++) {
        for(row = 0; row < IMAGE_ROWS; row++) {
            // set pixel to "false"
            goallimg[row][col] = 0;
            goal2img[row][col] = 0;
            ballimg[row][col] = 0;
        }
    }
}

```

```

        if(colimgfinal[row][col][0] == 0 &&
           colimgfinal[row][col][1] == 0 &&
           colimgfinal[row][col][2] == 0)
        {
            continue;
        }

        unsigned char class =
rgb2class(colimgfinal[row][col][0],colimgfinal[row][col][1],colimgfinal[row][col][2]);

        if(row < GOAL_MAX_ROW && class == CLASS_GOAL1) {
            goallimg[row][col] = 255;
        } else if(row < GOAL_MAX_ROW && class == CLASS_GOAL2) {
            goal2img[row][col] = 255;
        } else if(class == CLASS_BALL) {
            ballimg[row][col] = 255;
        }
    }
}

// the final images
GrayImage goallimgfinal, goal2imgfinal, ballimgfinal;

if((params & FIND_PARAM_CLEANUP_GOAL) != 0) {
    CleanupClassImage(&goallimg, &goallimgfinal, 4);
    CleanupClassImage(&goal2img, &goal2imgfinal, 4);
} else {
    for(col = 0; col < IMAGE_COLUMNS; col++) {
        for(row = 0; row < IMAGE_ROWS; row++) {
            goallimgfinal[row][col] = goallimg[row][col];
            goal2imgfinal[row][col] = goal2img[row][col];
        }
    }
}

if((params & FIND_PARAM_CLEANUP_BALL) != 0) {
#ifdef IMAGE_TYPE_BIG
    CleanupClassImage(&ballimg, &ballimgfinal, 4);
#else
    CleanupClassImage(&ballimg, &ballimgfinal, 2);
#endif
} else {
    for(col = 0; col < IMAGE_COLUMNS; col++) {
        for(row = 0; row < IMAGE_ROWS; row++) {
            ballimgfinal[row][col] = ballimg[row][col];
        }
    }
}

// count number of matching pixels in each column and row
short goallcolcount[IMAGE_COLUMNS],
goal2colcount[IMAGE_COLUMNS],
ballcolcount[IMAGE_COLUMNS];

short ballrowcount[IMAGE_ROWS],
goallrowcount[IMAGE_ROWS],
goal2rowcount[IMAGE_ROWS];

for(col = 0; col < IMAGE_COLUMNS; col++) {
    goallcolcount[col] = 0;
    goal2colcount[col] = 0;
    ballcolcount[col] = 0;

    for(row = 0; row < IMAGE_ROWS; row++) {
        if(col == 0) {
            ballrowcount[row] = 0;
            goallrowcount[row] = 0;
            goal2rowcount[row] = 0;
        }

        if(goallimgfinal[row][col] > 127) {
            goallcolcount[col] += 1;
            goallrowcount[row] += 1;
        }
        if(goal2imgfinal[row][col] > 127) {
            goal2colcount[col] += 1;
            goal2rowcount[row] += 1;
        }
    }
}

```

```

    }
    if(ballimgfinal[row][col] > 127) {
        ballcolcount[col] += 1;
        ballrowcount[row] += 1;
    }
}
}

// get middle of the pixels
unsigned int goallsum = 0,
goal2sum = 0,
ballsum = 0;

unsigned int ballcolsum = 0,
goallcolsum = 0,
goal2colsum = 0,
ballrowsum = 0,
goallrowsum = 0,
goal2rowsum = 0;

for(col = 0; col < IMAGE_COLUMNS; col++) {
    goallsum += goallcolcount[col];
    goal2sum += goal2colcount[col];
    ballsum += ballcolcount[col];

    goallcolsum += col*goallcolcount[col];
    goal2colsum += col*goal2colcount[col];
    ballcolsum += col*ballcolcount[col];
}

for(row = 0; row < IMAGE_ROWS; row++) {
    goallrowsum += row*goallrowcount[row];
    goal2rowsum += row*goal2rowcount[row];
    ballrowsum += row*ballrowcount[row];
}

// store number of matching pixels
goal1pos->pixels = goallsum;
goal2pos->pixels = goal2sum;
ballpos->pixels = ballsum;

// get middle of pixels
unsigned short goalthresh = 0;
if((params & FIND_PARAM_GOAL_THRESH) != 0) {
#ifdef IMAGE_TYPE_BIG
    goalthresh = 200;
#else
    goalthresh = 30;
#endif
}

if(goallsum > goalthresh) {
    goal1pos->x = goallcolsum/goallsum;
    goal1pos->y = goallrowsum/goallsum;
}
if(goal2sum > goalthresh) {
    goal2pos->x = goal2colsum/goal2sum;
    goal2pos->y = goal2rowsum/goal2sum;
}

#ifdef IMAGE_TYPE_BIG
    if( ( (params & FIND_PARAM_BALL_THRESH) != 0 && ballsum >= 1 ) ||
        ( (params & FIND_PARAM_BALL_THRESH) == 0 && ballsum > 0 ) )
#else
    if(ballsum > 0)
#endif
    {
        ballpos->x = ballcolsum/ballsum;
        ballpos->y = ballrowsum/ballsum;
    }

// find lower part of goal (=y)
if(goallsum > goalthresh) {
    for(row = IMAGE_ROWS; row > 0; row--) {
        if(goallrowcount[row] > 4) {
            goal1pos->y = row;
            break;
        }
    }
}

```

```

    }
}
}
if(goal2sum > goalthresh) {
    for(row = IMAGE_ROWS; row > 0; row--) {
        if(goal2rowcount[row] > 4) {
            goal2pos->y = row;
            break;
        }
    }
}
}

return 0;
}

/*
 * Sets every pixel above the wall to #000000
 */
void CutPixelsAboveWall(ColorImage *colimg) {
    short row,col;
    short y_lower[IMAGE_COLUMNS];

    // find the max y for a pixel classified as wall
    for(col = 0; col < IMAGE_COLUMNS; col++) {
        // find y position to cut at
        y_lower[col] = -1;
        char last_class = -1;

        // since the camera is looking as it is, the junk
        // won't be above CUT_LOW atm. row < IMAGE_ROWS can give weird results
        for(row = CUT_LOW; row > -1; row--) {
            unsigned char class =
rgb2class((*colimg)[row][col][0],(*colimg)[row][col][1],(*colimg)[row][col][2]);

            if(row == CUT_LOW && !(class == CLASS_WALL || class == CLASS_GOAL1 || class ==
CLASS_GOAL2)) {
                //last_class = class;
                break;
            } else if(row != CUT_LOW && last_class > -1 && class != last_class) {
                y_lower[col] = row;
                break;
            }

            if(row == CUT_LOW) {
                last_class = class;
            }
        }

        // if we didn't find any pixel in this column that
        // is a wall pixel, use the last one
        if(y_lower[col] < 0 && (col > 0 && y_lower[col-1] > -1)) {
            y_lower[col] = y_lower[col-1];
        } else if(y_lower[col] > -1 && (col > 0 && y_lower[col-1] > -1)) {
            // take mean value of the new y and last
            y_lower[col] = (y_lower[col]+y_lower[col-1])/2;
        }

        // anything to cut?
        if(y_lower[col] > -1) {
            for(row = 0; row <= y_lower[col]+4; row++) {
                (*colimg)[row][col][0] = 0;
                (*colimg)[row][col][1] = 0;
                (*colimg)[row][col][2] = 0;
            }
        }
    }
}

if(y_lower[0] < 0) {
    short colstart = 0;
    short cut_y = -1;

    for(colstart = 0; colstart < IMAGE_COLUMNS; colstart++) {
        if(y_lower[colstart] > -1) {
            cut_y = y_lower[colstart];
            break;
        }
    }
}

```

```

    }

    if(cut_y > -1) {
        for(row = 0; row <= cut_y+4; row++) {
            for(col = colstart-1; col >= 0; col--) {
                (*colimg)[row][col][0] = 0;
                (*colimg)[row][col][1] = 0;
                (*colimg)[row][col][2] = 0;
            }
        }
    }
}

/*
removes noise from an image where pixels belong to a class when they
have a value of 255 and 0 if they don't.
*/
void CleanupClassImage(GrayImage *img, GrayImage *outimg, unsigned short threshold) {
    short row, col;

    for(row = 0; row < IMAGE_ROWS; row++) {
        for(col = 0; col < IMAGE_COLUMNS; col++) {
            if(col == 0 || row == 0) {
                (*outimg)[row][col] = 0;
            } else {
                if((*img)[row][col-1] + (*img)[row][col+1] + (*img)[row-1][col] +
(*img)[row+1][col] + (*img)[row][col] >= threshold*255 ) {
                    (*outimg)[row][col] = 255;
                } else {
                    (*outimg)[row][col] = 0;
                }
            }
        }
    }
}
}

```

Testfindangles.c

```

#include <eyebot.h>
#include <chuck/imageproc.h>
#include <chuck/findobject.h>
#include <chuck/findangles_new.h>
#include <chuck/movement_new.h>
#include <chuck/helper.h>

int main(int argc, char** argv) {
    OSInitRS232( SER115200, NONE, SERIAL1 );
    /* start initialisations */
    LCDMode(SCROLLING|NOCURSOR); /* init lcd */
    LCDClear();
    Init_Cam(); /* init camera */
    adjustCam(); /* Set Camera FPS */
    LCDClear();

    float boc[] = {2.0, 0.5, 8.0, 0.12}; // Important tuned VW params

    VWHandle vw;
    vw = VWInit(VW_DRIVE,1);
    VWStartControl(vw,boc[0],boc[1],boc[2],boc[3]);

    // Set 0 position
    VWSetPosition(vw,0,0,0);

    LCDClear();

    ScanResult ball_result;
    ScanResult goal_result;
    ScanResult g1_result;
    ScanResult g2_result;

    ball_result.degs_turned = -17;
    ball_result.degs_in_image = -17;
    goal_result.degs_turned = -17;
    goal_result.degs_in_image = -17;
}

```

```

int opponent_goal = BLUE_GOAL;

while(1) {
    LCDClear();
    LCDPrintf("Opponent Goal: \n\n");
    LCDMenu("Blue", "", "Yell", "");
    int key = KEYGet();

    if(key == KEY1) {
        opponent_goal = BLUE_GOAL;
        LCDClear();
        break;
    } else if(key == KEY3) {
        opponent_goal = YELLOW_GOAL;
        LCDClear();
        break;
    }
}

while(1) {
    LCDClear();
    LCDPrintf("Drive forward?: \n\n");
    LCDMenu("yes", "no", "", "");
    int key = KEYGet();

    if(key == KEY1) {
        // K r fram 25 cm f r att undvika r da bilder n ra sargen.
        // Sv nger 15 grader f r att eventuellt slippa snurra 1 helt varv f r att
        // hitta bollen.
        DriveForward(0.35, &vw, 0.3);
        makeTurnNoStall(-15, &vw, 0.3);
        LCDClear();
        break;
    } else if(key == KEY2) {
        LCDClear();
        break;
    }
}

// int opponent_goal = BLUE_GOAL;
LCDPrintf("OppGoal %s\n", ((opponent_goal == BLUE_GOAL) ? "blue" : "yellow"));

int lazyTurnDegs;

while(1)
{
    LCDPrintf("Main loop\n");

    lazyTurnDegs = -1;

    if(ScanForBall(&vw, &ball_result, &g1_result, &g2_result) == 0)
    {
        if(DriveToBall(&vw, &ball_result) > -1) {

            if(GetSeenGoal(&vw, &g1_result, &g2_result, opponent_goal, &goal_result) == 1)
                lazyTurnDegs = LazyPositionGoal(&vw, &ball_result, &goal_result);
            ScanForGoal(&vw, &goal_result, opponent_goal);
            if(lazyTurnDegs != -1)
                goal_result.degs_turned += lazyTurnDegs;
            PositionToShoot(&vw, &goal_result);
        }
    }
}

VWStopControl(vw);
VWRelease(vw);
return 0;
}

```

Movement_new.h

```

#ifndef __HEADER_CHUCK_MOVEMENT_NEW
#define __HEADER_CHUCK_MOVEMENT_NEW

```

```

#include <math.h>
#include <chuck/types.h>
#include <chuck/distlut.h>

int DriveToBall(VWHandle* vw, ScanResult* ball_result);

int PositionToShoot(VWHandle* vw, ScanResult* goalresult);
void DriveForward(float distance,VWHandle* vw,float speed);
void DriveForwardNoWait(float distance,VWHandle* vw,float speed);
void ApproachBall(VWHandle* vw, float drive_distance);
int makeTurnNoStall(int degrees, VWHandle* vw,float speed);
int GetSeenGoal(VWHandle* vw,ScanResult* g1_result, ScanResult* g2_result,int findgoal,
ScanResult* goalresult);
int LazyPositionGoal(VWHandle* vw,ScanResult* ballresult, ScanResult* goalresult);

#endif

```

Movement_new.c

```

#include <chuck/movement_new.h>
#include <chuck/findangles_new.h>
#include <chuck/trigtabs.h>
#include <chuck/helper.h>

#define FAR_DISTANCE (0.97)
#define GOAL_FAR_DISTANCE (0.5)
#define DRIVE_BACKWARD_DIST (0.10)

#define STRANGE_IMAGE_TURN (15)
#define TURN_SPEED (0.7)

#define LOW_TURN_SPEED (0.7)
#define CURVE_SPEED (0.3)
#define FORWARD_SPEED (0.3)
#define FORWARD_SHOOT_SPEED (1.0)

int fix_turn_ang(int deg)
{
    int rdeg = 0;
    float scale = 0.9f;

    if(deg < 60)
        rdeg = deg + 0.1*(deg-60);
    else
        rdeg = deg;

    return (int)(scale*rdeg);
}

int normalizeDeg(int degrees)
{
    while(degrees > 180 || degrees < -180)
    {
        if(degrees == 360 || degrees == -360)
            degrees = 0;
        else if(degrees > 180)
            degrees -= 360;
        else if(degrees < -180)
            degrees += 360;
    }
    return degrees;
}

int normalizeDegEasy(int degrees)
{
    if(degrees > 360)
        degrees = degrees-360;
    return degrees;
}

int makeTurnNoStall(int degrees, VWHandle* vw,float speed)
{
    degrees = normalizeDeg(degrees);

```

```

float rads = (M_PI/180.0)*((float)degrees);
VWDriveTurn(*vw, rads, speed);
VWDriveWait(*vw);
return 0;
}

int sign(int val)
{
    if(val < 0 )
        return -1;
    else
        return 1;
}

void DriveForward(float distance,VWHandle* vw,float speed)
{
    if(fabs(distance) < 0.02)
        distance = fsign(distance)*0.02;

    VWDriveStraight(*vw, distance, speed);
    if(VWDriveWaitAndCheckTilt(*vw) == -1)
    {
        LCDPrintf("DF, STALLED\n");
        // Stalled, drive 2 dm in reverse direction of last drive
        VWDriveStraight(*vw, 0.2*-1*sign(distance), speed);
        VWDriveWaitAndCheckTilt(*vw);
    }
}

void DriveForwardNoWait(float distance,VWHandle* vw,float speed)
{
    VWDriveStraight(*vw, distance, speed);
}

int DriveToBall(VWHandle* vw, ScanResult* ball_result)
{
    LCDPrintf("DriveToBall\n");

    float distance;
    ScanResult scan_result;
    int lostballcount = 0;

    makeTurnNoStall(ball_result->degs_in_image,vw,TURN_SPEED);
    distance = calcDist(ball_result->pos.y);
    LCDPrintf("IY: %d\n",ball_result->pos.y);

    if(ball_result->pos.y >= IMAGE_ROWS-8) {
        DriveForward(-0.02, vw, FORWARD_SPEED);
        return -2;
    }

    // We are far from the ball
    while(distance > FAR_DISTANCE)
    {
        int res = ScanForBallNoTurn(&scan_result);

        // Image is too strange to detect anything, change position
        if(res == -1)
        {
            LCDPrintf("STRANGE IMAGE\n");
            makeTurnNoStall(-STRANGE_IMAGE_TURN, vw, TURN_SPEED);
            DriveForward(-0.03, vw, FORWARD_SPEED);
        }
        // We just lost sight of the ball, maybe too close?
        if(res == -2)
        {
            if(lostballcount > 0)
            {
                LCDPrintf("LOST BALL TWICE\n");
                return -1;
            }
            LCDPrintf("LOST BALL\n");
            DriveForward(-0.02, vw, FORWARD_SPEED);
            lostballcount++;
        }
        // too close to ball?

```

```

    if(res == 0) {
        // We found the ball
        lostballcount = 0;
        distance = calcDist(scan_result.pos.y);
        if(scan_result.degs_in_image > 4 || scan_result.degs_in_image < -4)
        {
            makeTurnNoStall(scan_result.degs_in_image,vw,LOW_TURN_SPEED);
        }
        // We are in a nice position, abort
        if(distance <= FAR_DISTANCE)
        {
            //LCDPrintf("NICE POS\n");
            return 0;
        }
        ApproachBall(vw,distance);
    }
}
return 0;
}

void ApproachBall(VWHandle* vw, float drive_distance)
{
    //LCDPrintf("APPROACH %.2f\n",drive_distance);
    //AUBeep();
    float m_dist = 0.1 * drive_distance;
    LCDPrintf("SEEN BALL %.2f\n",m_dist);
    if(m_dist > 0.5)
    {
        LCDPrintf("drive1 %.2f\n",m_dist-0.4);
        DriveForward(m_dist-0.4,vw,FORWARD_SPEED);
    }
    else if (m_dist > 0.3)
    {
        LCDPrintf("drive2 %.2f\n",m_dist-0.25);
        DriveForward(m_dist-0.25,vw,FORWARD_SPEED);
    }
    else
    {
        float newdist = m_dist-(FAR_DISTANCE+0.4)*0.1;
        if(newdist < 0) {
            newdist += 0.04;
        }
        LCDPrintf("drive3 %.2f\n",newdist);
        DriveForward(newdist,vw,FORWARD_SPEED);
    }
}

void driveToShoot(VWHandle* vw, ScanResult* goalresult)
{
    Point ballpos,goallpos,goal2pos;
    LCDPrintf("CAN I SHOOT?\n");
    // Turn towards the goal
    makeTurnNoStall(goalresult->degs_in_image,vw,TURN_SPEED);
    FindAllAuto(&ballpos,&goallpos,&goal2pos);
    float distball = calcDist(ballpos.y);
    LCDPrintf("balldist=%.2f\n", distball);
    if(distball < 0.4) {
        BeginShoot();
        DriveForwardNoWait(0.15,vw,FORWARD_SHOOT_SPEED);
    }
    else
    {
        DriveForwardNoWait(0.15,vw,FORWARD_SHOOT_SPEED);
        OSWait(50);
        BeginShoot();
    }
    EndShoot();
    VWDriveWaitAndCheckTilt(*vw);
}

int PositionToShoot(VWHandle* vw, ScanResult* goalresult)
{
    LCDPrintf("PositionToShoot\n");
    //LCDPrintf("GT = %d\n",goalresult->degs_turned);
    //LCDPrintf("GI = %d\n",goalresult->degs_in_image);
    int ballgoalangle = goalresult->degs_turned+goalresult->degs_in_image;
}

```

```

float goal_distance = calcDist(goalresult->pos.y);

int goal_degree_offset;

// If we are far away from the goal, we are picky about when it's ok to shoot
if(goal_distance > GOAL_FAR_DISTANCE)
    goal_degree_offset = 10;
else
    goal_degree_offset = 20;

ballgoalangle = normalizeDegEasy(ballgoalangle);

//LCDPrintf("DIST = %.2f\n",goal_distance);
//LCDPrintf("BGA = %d\n",ballgoalangle);
//AUBeep();

// We are not centered on the goal, so we have to reposition ourselves
if( (ballgoalangle > goal_degree_offset) && (ballgoalangle < 360-goal_degree_offset) )
{
    // Turn back so that we are 90 deg off the ball line

    int backattheball = -goalresult->degs_turned;

    if(ballgoalangle < 180)
        {
            //LCDPrintf("BACKTURN :%d\n",backattheball-90);
            makeTurnNoStall(backattheball-90,vw,TURN_SPEED);
        }
    else
        {
            //LCDPrintf("BACKTURN :%d\n",backattheball+90);
            makeTurnNoStall(backattheball+90,vw,TURN_SPEED);
        }

    // ballgoalangle always -180..180
    if(ballgoalangle < -180)
        ballgoalangle+=360;
    else if(ballgoalangle >= 180)
        ballgoalangle-=360;

    //LCDPrintf("BGA2 = %d\n",ballgoalangle);
    //AUBeep();

    // Fix a circle which will reposition us
    int rotdegs = fix_turn_ang(ballgoalangle);

    // POSTIVE curverads is CLOCKWISE !!

    float curverads = (M_PI/180.0)*((float)rotdegs);
    float curvelen = fabs(1.0f*((float)rotdegs)/360.0f);

    LCDPrintf("CRADS = %.2f\n",curverads);
    LCDPrintf("CLEN = %.2f\n",curvelen);
    //KEYWait(KEY4);

    if(curvelen < 0.015)
        {
            LCDPrintf("LOW CURVE\n");

            if(ballgoalangle < 180 )
                makeTurnNoStall(backattheball+90,vw,TURN_SPEED);
            else
                makeTurnNoStall(backattheball-90,vw,TURN_SPEED);

            driveToShoot(vw,goalresult);
            return 0;
        }

    VWDriveCurve(*vw, curvelen, curverads, CURVE_SPEED);

    VWDriveWaitAndCheckTilt(*vw);

    if(ballgoalangle < 0)
        makeTurnNoStall(-70,vw,TURN_SPEED);
    else
        makeTurnNoStall(70,vw,TURN_SPEED);
}

```

```

    }
    else
        driveToShoot(vw,goalresult);
    return 0;
}

int GetSeenGoal(VWHandle* vw,ScanResult* g1_result, ScanResult* g2_result,int findgoal,
ScanResult* goalresult){
    ScanResult* oppgoal;
    if(findgoal == BLUE_GOAL)
        oppgoal = g1_result;
    else
        oppgoal = g2_result;
    if(oppgoal->pos.y != -1)
    {
        goalresult->degs_turned = oppgoal->degs_turned;
        goalresult->degs_in_image = oppgoal->degs_in_image;
        goalresult->pos.x = oppgoal->pos.x;
        goalresult->pos.y = oppgoal->pos.y;
        goalresult->pos.pixels = oppgoal->pos.pixels;
        return 1;
    }
    else
    {
        LCDPrintf("No help\n");
        return 0;
    }
}

int LazyPositionGoal(VWHandle* vw,ScanResult* ballresult, ScanResult* goalresult)
{
    int goalangle = -(ballresult->degs_turned+ballresult->degs_in_image)+(goalresult->
>degs_turned+goalresult->degs_in_image);
    LCDPrintf("LAZY = %d\n",goalangle);
    /*LCDPrintf("BT = %d\n",ballresult->degs_turned);
    LCDPrintf("BI = %d\n",ballresult->degs_in_image);
    LCDPrintf("GT = %d\n",goalresult->degs_turned);
    LCDPrintf("GI = %d\n",goalresult->degs_in_image);*/
    //AUBeep();
    //KEYWait(KEY4);
    makeTurnNoStall(goalangle,vw,TURN_SPEED);
    if(goalangle < 0)
        return goalangle+360;
    return goalangle;
}

```

Findangles_new.h

```

#ifndef __HEADER_CHUCK_FINDANGLES_NEW
#define __HEADER_CHUCK_FINDANGLES_NEW

#include <chuck/debug.h>
#include <types.h>
#include <chuck/findobject.h>
#include <chuck/movement_new.h>
#include <chuck/imageproc.h>
#include <chuck/types.h>
#include <chuck/distlut.h>
#include <eyebot.h>

#define FIND_BALL 1
#define FIND_GOAL 0
#define TURN 1
#define NOTURN 0
#define GOAL1 1
#define GOAL2 2

#define BLUE_GOAL GOAL1
#define YELLOW_GOAL GOAL2

void ScanForBallOrGoal(VWHandle* vw, int* ballp, int* goalp, float* distance , int findball,
int turn, int findgoal, int* turnb, int* turng, int* ballingangle, int* goalimgangle);
int ScanForBall(VWHandle* vw, ScanResult* ball_result, ScanResult* goal1_result, ScanResult*
goal2_result);
void ScanForGoal(VWHandle* vw, ScanResult* find_result, int findgoal);
int ScanForBallNoTurn(ScanResult* find_result);

```

```
int PosInImageToDegs(Point* pos);
```

```
#endif
```

Findangles_new.c

```
#include <chuck/findangles_new.h>
```

```
#include <chuck/helper.h>
```

```
#include <math.h>
```

```
#define TURN_ANGLE (45)
```

```
#define EXTRA_TURN_ANGLE (15)
```

```
#define X_TO_DEGREES (60.0f/IMAGE_COLUMNS) // 60 degs FOV
```

```
#define PICTURE_WAIT (0)
```

```
#define TOWARDS_GOAL_SPEED (0.7)
```

```
#define TOWARDS_GOAL_DISTANCE (0.5)
```

```
#define SCAN_TURN_SPEED 1.0
```

```
void ScanForGoal(VWHandle* vw, ScanResult* find_result, int findgoal){  
    LCDPrintf("ScanForGoal %s\n",((findgoal == BLUE_GOAL) ? "blue" : "yellow"));  
    int degs_turned = 0;
```

```
    Point ballpos,goallpos,goal2pos;
```

```
    Point* goalpos;
```

```
    if(findgoal == BLUE_GOAL)
```

```
        goalpos = &goallpos;
```

```
    else
```

```
        goalpos = &goal2pos;
```

```
    int LastGoalPixels = -1;
```

```
    while(1)
```

```
    {
```

```
        // Search for goal
```

```
        OSWait(PICTURE_WAIT);
```

```
        int res = FindAllAuto(&ballpos,&goallpos,&goal2pos);
```

```
        if(res != -1 && goalpos->x != -1)
```

```
        {
```

```
            LCDPrintf("FOUND GOAL p:%d\n",goalpos->pixels);
```

```
            LastGoalPixels = goalpos->pixels;
```

```
            find_result->pos = *goalpos;
```

```
            find_result->degs_turned = degs_turned;
```

```
            find_result->degs_in_image = PosInImageToDegs(goalpos);
```

```
            break;
```

```
        }
```

```
        else
```

```
        {
```

```
            makeTurnNoStall(TURN_ANGLE, vw, SCAN_TURN_SPEED);
```

```
            degs_turned += TURN_ANGLE;
```

```
            if(degs_turned > 360)
```

```
                break;
```

```
        }
```

```
    }
```

```
    // Turn once more
```

```
    makeTurnNoStall(EXTRA_TURN_ANGLE, vw, 0.3);
```

```
    degs_turned += EXTRA_TURN_ANGLE;
```

```
    if(degs_turned > 360)
```

```
        degs_turned-=360;
```

```
    // Search for goal again
```

```
    OSWait(PICTURE_WAIT);
```

```
    int res = FindAllAuto(&ballpos,&goallpos,&goal2pos);
```

```
    // If new goal is found and better match, use that one instead
```

```
    if(res != -1 && goalpos->x != -1 && goalpos->pixels > LastGoalPixels)
```

```
    {
```

```
        find_result->pos = *goalpos;
```

```
        find_result->degs_turned = degs_turned;
```

```

        find_result->degs_in_image = PosInImageToDegs(goalpos);
    }
    else
        // Turn back, the new goal pose was worse
        makeTurnNoStall(-EXTRA_TURN_ANGLE,vw,0.3);
}

int ScanForBallNoTurn(ScanResult* find_result){
    Point ballpos,goallpos,goal2pos;

    //LCDPrintf("ScanForBallNT\n");

    OSWait(PICTURE_WAIT);
    int res = FindAllAuto(&ballpos,&goallpos,&goal2pos);

    if(res == -1)
        return -1;

    if(ballpos.x != -1 && ballpos.y != -1){
        find_result->pos = ballpos;
        find_result->degs_in_image = PosInImageToDegs(&ballpos);
        return 0;
    }

    return -2;
}

int ScanForBall(VWHandle* vw, ScanResult* ball_result, ScanResult* goall_result, ScanResult*
goal2_result){

    LCDPrintf("ScanForBall\n");
    int degs_turned = 0;

    Point ballpos,goallpos,goal2pos;

    //ScanResult goall_result,goal2_result;
    goall_result->pos.x = -1;
    goall_result->pos.y = -1;
    goall_result->pos.pixels = -1;

    goal2_result->pos.x = -1;
    goal2_result->pos.y = -1;
    goal2_result->pos.pixels = -1;

    int goall_turn = 0;
    int goal2_turn = 0;

    while(1)
    {
        OSWait(PICTURE_WAIT);
        int res = FindAllAuto(&ballpos,&goallpos,&goal2pos);

        if(goallpos.x != -1 && goall_result->pos.pixels < goallpos.pixels)
        {
            LCDPrintf("G1 = %d\n",degs_turned);
            goall_result->degs_turned = degs_turned;
            goall_result->pos = goallpos;
            goall_result->degs_in_image = PosInImageToDegs(&goallpos);
            goall_turn = 0;
        }
        if(goal2pos.x != -1 && goal2_result->pos.pixels < goal2pos.pixels)
        {
            LCDPrintf("G2 = %d\n",degs_turned);
            goal2_result->degs_turned = degs_turned;
            goal2_result->pos = goal2pos;
            goal2_result->degs_in_image = PosInImageToDegs(&goal2pos);
            goal2_turn = 0;
        }

        if(ballpos.x != -1 && ballpos.y != -1)
        {
            ball_result->pos = ballpos;
            ball_result->degs_turned = degs_turned;
            ball_result->degs_in_image = PosInImageToDegs(&ballpos);
            return 0;
        }
    }
}

```

```

else
}
{
    if(res == -1)
    {
        LCDPrintf("FINDALL -1\n");
        VWDriveStraight(*vw, -0.05, 0.2);
        VWDriveWaitAndCheckTilt(*vw);
        makeTurnNoStall(-15,vw,0.3);
        return -1;
    }
    else
    {
        makeTurnNoStall(TURN_ANGLE, vw, SCAN_TURN_SPEED);
        degs_turned += TURN_ANGLE;
        goall_turn += TURN_ANGLE;
        goal2_turn += TURN_ANGLE;

        // We have turned 360 and seen to ball
        if(degs_turned > 360)
        {
            LCDPrintf("360 NO BALL!\n");
            if(goal1_result->pos.x == -1 && goal2_result->pos.x == -1)
            {
                LCDPrintf("TOTALLY LOST!\n");
                DriveForward(TOWARDS_GOAL_DISTANCE, vw, TOWARDS_GOAL_SPEED);
                return -1;
            }
            else
            {
                // We choose to drive towards the farthest goal (smallest y)
                if(goal1_result->pos.y != -1 && goal1_result->pos.y <
goal2_result->pos.y)
                {
                    LCDPrintf("FARGOAL GOAL1\n");
                    LCDPrintf("%d %d\n",-goal1_turn,goal1_result-
>degs_in_image);
                    makeTurnNoStall(-goal1_turn+goal1_result-
>degs_in_image,vw,SCAN_TURN_SPEED);
                }
                else if (goal1_result->pos.y != -1)
                {
                    LCDPrintf("FARGOAL GOAL2\n");
                    LCDPrintf("%d %d\n",-goal2_turn,goal2_result-
>degs_in_image);
                    makeTurnNoStall(-goal2_turn+goal2_result-
>degs_in_image,vw,SCAN_TURN_SPEED);
                }
                else // We have seen no goals!
                {
                    LCDPrintf("NO FARGOAL!!\n");
                }
                DriveForward(TOWARDS_GOAL_DISTANCE, vw, TOWARDS_GOAL_SPEED);
                return -1;
            }
        }
        degs_turned = 0;
    }
}
}

return -1;
}

int PosInImageToDegs(Point* pos)
{
    return (pos->x - IMAGE_COLUMNS/2)*X_TO_DEGREES;
}

```