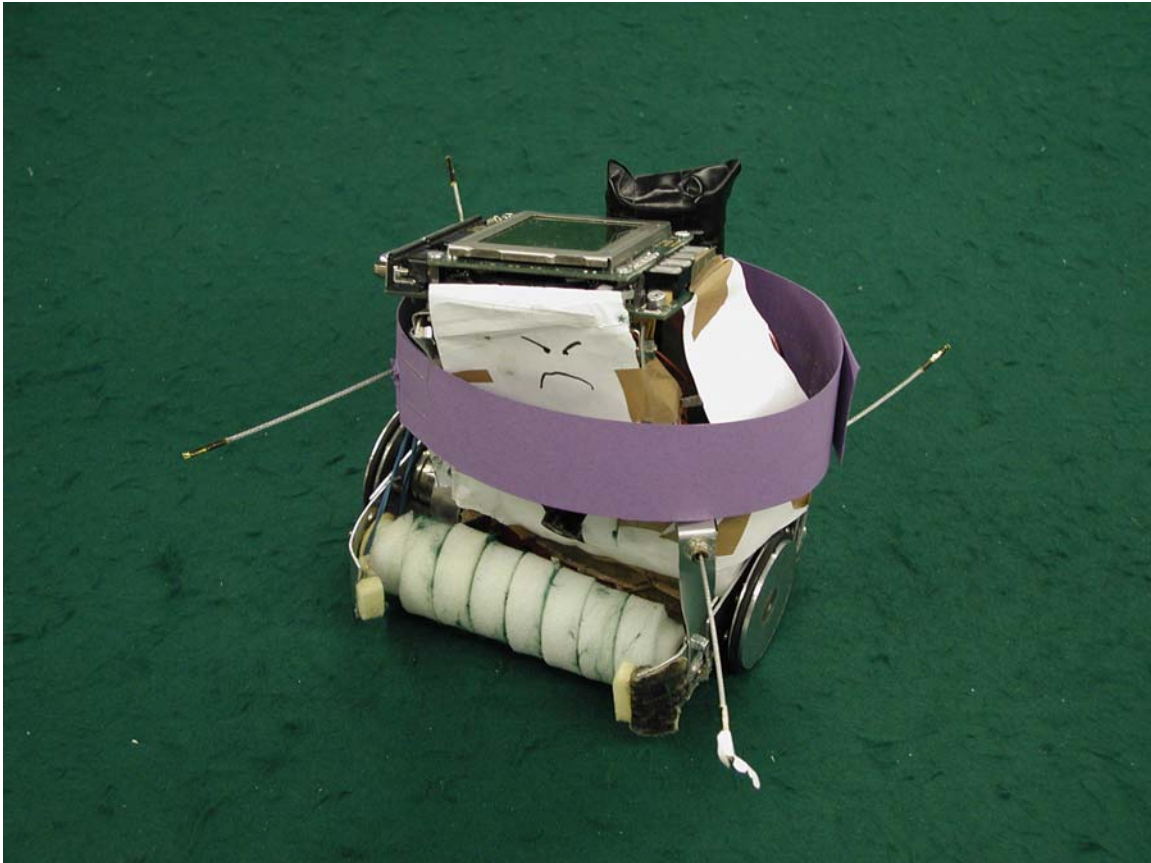


# Project report

for the course 2D1426, *Robotics and Autonomous Systems*

Team **Robo Romo**



Alexander Ekvall  
Jonas Frogvall  
Sebastian Geidenstam  
John Lindholm

### ***Abstract***

We have constructed a soccer playing robot for the course *2D1426, Robotics and Autonomous Systems* at KTH (the Royal Institute of Technology). We finished third in the annual tournament and this report handles the construction and conclusions around the robot Romo Robo.

## Table of contents

Introduction	2
The name	3
The Robot	3
Sensing	3
Camera	3
Whiskers	3
Motors	3
Manipulation	3
Locomotion	4
Detailed Functionality	4
Robot vision	4
VW-drive	5
Trajectory Control	5
Corrections to make the Robot go straight	5
Updating the HDT	5
Making the PID controller	6
Making curves	6
Curves from the book	6
Cubic Bezier curves	7
Using the v $\omega$ -interface	7
Drive straight and turn on the spot	8
Behavior programming	8
Conclusion	9
References	10
Appendix A – Source Code	11
Appendix B – Whiskers Control	47

## Introduction

This is the project report for team Robo Romo, concluding the 2006 KTH Championship of Robot Soccer. The team finished in third place. The report handles the specifics of this fantastic robot, the original plan, the issues we faced and how it all ended up.

We entered this course with high expectations and found out that the possibilities were more limited than we could ever have imagined. We tried many different ways of steering the robot, but none seemed to work as well as we wished, and we spent all time working on that and finally we had a solution that had to do.

We also shortly describe the support vector machine that we used for finding different objects on the field.

Sadly we had to spend too much time implement the basics so we couldn't implement the more entertaining parts.

The rules for the soccer game were specified by the course coordinator [1].

## The name

One of the most important things to know about Robo Romo is the origin of the name. The name defines his personality and view on the other robots in the tournament. To fully understand the name Robo Romo, you have to watch Futurama episode 54: *The 30% Iron Chef* [2].

## The Robot

To build our robot, we were supplied with an Eyebot controller [3], an EyeCam module [4], two DC motors with wheels, a battery, some mekano and some aluminum boards. We also provided some material on our own, including: Two smaller batteries, a motor for the front roller, the parts for the roller, a motor for the back wheel, the back wheel and some bolts.

## Sensing

The sensing system for our robot consisted of the camera module, four whiskers and the quadrature encoders in the motors.

### Camera

The camera position was tested in a few positions, but since we thought that it was most important to know if we had the ball in our possession we choose a pretty low position, pointing somewhat downward. This gave a great certainty that we had or had not possession of the ball, but we lost precision for objects far away and therefore had a hard time judging distances.

### Whiskers

The whiskers were pointed out from the robot, roughly following the 45° axes. This gave almost 360 ° coverage from running into obstacles. One sad exception was if we ran into the outer corner of one of the goals and was so unlucky that we hit the corner in such an angle that all whiskers missed the wall (see figure 1). We ignored this problem though, as we figured the risks for getting stuck like that were slim.

Another problem with the whiskers was the naïve behavior programming for it (see *Behavior programming*). In the final rounds we were running with the ball against the opposing goal, with no one that could stop us, except that the opposing robot ran into our right back whisker, which made us turn left and miss the goal completely.

### Motors

The motors of the robot had built-in quadrature encoders for knowing how far it has traveled. We used those in the qualifying round to score a fast first goal (in fact we were the fastest scoring robot in the qualifying round, scoring in 12 seconds). Otherwise we used them to check if we were stalled.

## Manipulation

To manipulate the ball we had a roller, made of the centerpiece of a receipt roll, which we embedded in a trim from a window. We mounted

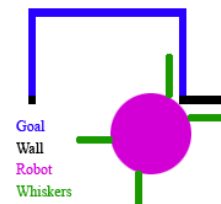


Figure 1: Stuck in corner.

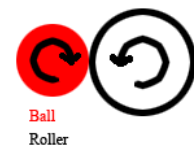


Figure 2: Roller manipulation.

the roller in the front of the robot, with the vertical axis at the same height as the balls vertical axis. We made the roller spin (with an motor, which was always turned on and connected to one of the batteries) as shown in figure 2, making the ball backspin towards us and stay with us. We also made the edges of the roller stand out to make sure we didn't drop the ball while turning.

We had one major problem with the rollers. The motor driving it made the whole robot shake, which made the whiskers react like they driven into a wall. This was solved by implementing a Faraday cage [5].

## Locomotion

For making the robot drive, we used two DC motors with encapsulated gears and quadrature encoders, using a differential drive configuration. The motors are connected to the EyeBot controller. We used a bolt at the back to stabilize the robot.

We also planned on having a third wheel, for extra burst, due to the rule that said that we can push an opponent into the goal and score a goal if the opponent touches the ball while inside his goal. We also had a bolt on a servo that could be raised, so that the extra wheel touched the ground. We couldn't have the extra wheel touching the ground all the time, due to the extra friction making us unable to turn. The wheel seemed to work fine, but had to be removed, along with the servo, at the end, due to space and time restrictions.

## Detailed functionality

### Robot vision

The robot Robo Romo's vision consisted of two main parts of which the first was to extract a proper image from the EyeBot's sensing unit EyeCam to prepare it for further processing, and secondly to classify those pixel values so that objects could easily be located.

Reading pixel data from the camera was done using the RoBIOS system calls for reading full size images from the camera, storing it in data arrays in memory for later processing. Still full size image processing for later classification is a quite heavy load considering EyeBot processing power. A way to reduce processing cost was simply to skip every other row and column of data in the image array. This technique lead to faster picture processing, but also to worse classification of objects lying to far away.

Secondly the extracted, and cropped, image data had to be classified to be able to find information about objects in the image essential for the robot path planning. In the early phase of development Robo Romo made use of a simple way for object classification. It read image pixels, converted them from the standard RGB values to HUE values (for a more stable color variation), and classified them according to pre-set classification intervals, say BALL had an H-value between 30-50degr. Unfortunately this method



Figure 3: EyeBot camera module.

appeared to have some indirect flaws. The EyeCam makes use of an auto-brightness functionality leaving the programmer with cam-pictures spanning over wide intervals of colors depending on earlier light exposure into the camera lens. This made the Robo Romo team to start considering in different ways of object classification, finally agreeing on a classification method known as SVM, Support Vector Machine, classification. What this meant in reality was that one takes a lot of pictures of objects one wants to classify, thereafter going through these pictures specifying what pixels in the images that can be associated with specific classes. For example all the balls found will have its corresponding image pixels chosen to classify a ball, more or less, thereafter inserting all classification data into a Support Vector Machine to find what pixel values are most likely to belong to a certain class. Team Robo Romo put together a small JAVA program for the pixel extraction and made use of an available, free of charge, SVM Matlab library, OSU SVM, for classification. The result of the classification was then used in a lookup-table transferred into the EyeBot's memory, for very fast classification of image pixels into detectable objects.

When creating lookup-tables one must consider the tradeoff between accuracy and size. For Robo Romo a total table size of 65kB was sufficient, letting each pixel from the image be classified by an accuracy of 5 bits per color.

## **VW-drive**

Included in the RoBIOS operating system for the mobile EyeBot platform there is a high level wheel control API that makes use of motor and quad primitives to drive the robot. The API is put together with respect to a double-wheeled differential drive robot in mind and provides a series of steering possibilities including driving alongside curves, straight lines, setting the driving/angular speed of the robot etc. By using a PI-controller (Proportional Integrational), correct wheel speed can be held, preventing course drifting and other effects due to physical differences.

## **Trajectory Control**

We had several approaches to trajectory control during the course. The first and very naïve one was to turn the robot by setting the motor inputs to the same but opposite velocities and the drive straight by setting both motors to the same input. This didn't work well at all since motor input is not the same as velocity, and not proportional either. The left motor tended to go slower than the right and we had to fix this before being able to move on.

## **Corrections to make the Robot go straight**

### **Updating the HDT**

The first thing we did in order to try and make the robot perform as we would like it to was to update the HDT-file. By changing in vectors corresponding to input for the left and right motor we should be able to say that i.e. 20 input for the left motor corresponded to 35 for the right one. By making a program in which input of both motors could be changed and speeds measured by the quads (distance) and TPU (time) were shown we were able to make a table of how the inputs should be set to produce the same speed. We also used a ROBIOS function already on the control board to measure ticks per meter by

dragging the robot 5 meters by hand and reading the number of tics from the screen we could update the HDT file with new values of tics per meter for the wheels [6].

### Making a PID controller

The second thing we did was to create our own PID controller, this to help us get the speeds up to what we wanted in both wheels. In simple terms the controllers three different components work as follows:

- **The P part** Proportional part, this part controls the difference between the wanted value of  $v$ ,  $v_{dec}$ , and it's actual value  $v_{act}$ , by changing the input proportional to this difference. This counteracts the error in part but always shoots a bit low since when there is almost no error no change will occur.

$$\Delta v = K_P(v_{dec} - v_{act})$$

- **The I part** Integral part, this part makes the controller reach it's desired velocity by summing the errors of the system. In actuality this is made by reading the quads for the actual distance traveled by the wheel and comparing it to the desired distance.

$$\Delta v = K_I(d_{dec} - d_{act})$$

- **The D part** Derivative part, this part prevents oscillation of the system by reducing the changes in input when fast changes occur.

$$\Delta v = K_D(d_{dec} - d_{act})$$

The most difficult part of this implementation was the tuning of control parameters. If the parameters were set to low the system would not change the speed fast enough or even to the correct speed. If the parameters were set to high the system would start oscillating and this would mean catastrophe on the field. For details on how to tune the PID see Embedded robotics [7].

### Making curves

Whilst effective the straight line and "turn-on-the-spot" was quite boring and so we wanted to make our robot move in more advanced patterns, preferably in smooth curves.

#### Curves from the book

In our course literature [8] we found a way to control the velocity and angular velocity of the robot in order to make smooth curves from point A to point B. By putting together the equations

$$v = k_\rho \rho$$

$$\omega = k_\alpha \alpha + k_\beta \beta$$

where

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\alpha = -\theta + \arctan 2(\Delta y, \Delta x) \quad [9]$$

$$\beta = -\theta - \alpha$$

With the equations

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} \frac{\dot{r}\varphi_1}{2} + \frac{\dot{r}\varphi_2}{2} \\ 0 \\ \frac{\dot{r}\varphi_1}{2l} - \frac{\dot{r}\varphi_2}{2l} \end{bmatrix} \quad [10]$$

to get the desired velocities,  $\dot{\varphi}_{1,2}$  for each of the wheels. In combination with our PID control this should make the robot go smooth curves with the possibility to tune its behavior with the parameters  $k_p$ ,  $k_\alpha$  and  $k_\beta$ .

This was all well in theory but when we tried it out it turned out that the curves often became too “curvy” and that the end heading very seldom ended up where we wanted it. We also had some problems with the robot missing the endpoint and these were the reasons we tried to combine this control law for making smooth curves with one where end heading could be decided and which could place a number of waypoints so that the curves would be smoother.

### Cubic Bezier Curves

When searching the web and our minds for suitable equations for curves we remembered, from an old course in numerical methods, that the Bezier curve satisfied our demands. Given a start point and start heading you could decide an end point and also the robots end heading. The Bezier curve also satisfies our demand on smoothness since its derivative is continuous at all points. The desire to be able to divide the curve into smaller with waypoints was also satisfied since the discrete form of the equation uses a discrete time vector and thus divides, in itself, the curve into small parts. The equation of the curve can be seen below.

$$r(t) = (1-t)^3 p_{start} + 3t(1-t)^2 b + 3t^2(1-t)c + t^3 p_{end} \quad [11]$$

where

$$0 \leq t \leq 1$$

### Using the vw-interface

Though better than the original curves alone, the Bezier- book curve combination didn't work flawless. Especially troublesome was the last distance of each curve. Here the change in angel was often very large in comparison with the distance to travel and sometimes the robots control system didn't have enough time to correct for these resulting in the robot missing the endpoint and going around in circles for a while trying to come back to it. While very good when they worked, the curves were deemed too unreliable to use in the tournament and since our time was running out we decided not to continue with the curves trying to make them work as desired, but to instead use the ready made solution provided by the vw-interface.



- **Start State** The robot drives straight ahead to the point (0.8, 0) while initializing the camera.
- **Search Ball** The robot takes a picture, searches for the ball and a specified goal in the picture, if the ball is not found it turns  $\pi/4$ . If the robot has found the specified goal twice and the total angle turned is more than  $3\pi/2$ . The robot turns towards the center of the specified goal and if that goal has not got more than a certain number of pixels in the picture the robot drives towards it 50 cms and starts over. If the goal has more than the specified pixels the robot turns  $\pi$  and drives forward 50 cms, changes the specified goal and starts over.
- **Drive to Ball** The robot has found the ball on the field and turns the angle given by  $\alpha\text{LUT}^1$  and drives straight the distance given by  $\text{distanceLUT}$  with the center of mass of the ball as indexes for the lookup tables.
- **Score** The robot has not yet been delocalized and got the ball (the ball is perceived to be closer than 4 cm from the camera). The robot turns towards where the goal should be and then drives straight into where the goal should be.
- **Score Second Try** The robot has been delocalized but got the ball. A picture is taken and if the opposite goal is not found in the picture the robot turns  $\pi/4$  to the right. If the opposite goal is found the robot starts turning towards its center. If the center of the goal has an x value above 43 the robot turns left with  $\pi/8$  and if the center of the goal has an x value below 39 the robot turns right with  $\pi/8$ . If the goal's center is between 39 and 43 (middle of the picture) the robot drives straight for 2 m or until a bumper is bumped. If the robot has turned once left and once right the robot starts turning with half the angle of before ( $\pi/16$ ). If once more the robot has turned once left and once right without the goal being in-between 39 and 43 it halves the step again, until the goal is in the right interval or the minimum turn angle has been reached ( $\pi/64$ ). If the goal is not found after two complete turns (total angle of  $> 4\pi$ ), the robot drives straight ahead 50cms so that it does not get stuck in an infinite loop.
- **Bump State** This state is a global state into which all the states go if a bumper is activated (by a collision). It turns away from the direction it has been bumped and drives 20 cms straight away from the direction. For example if the front left bumper is bumped, the robot turns  $\pi/2$  to the left and reverses 20 cms. If the back right bumper is bumped the robot turns  $\pi/2$  to the left and drives straight ahead 20cms. The state also sets the global variable `delocalized` to 1, and returns to the Search Ball State no matter what state it came from.

## Conclusions

We started out with a very simple idea of a robot, which turned out to be not that simple at all. The finesse with the wheel should have been easy to implement, but we got stuck for weeks with the steering programming (and to some extent the vision), which was a prerequisite for the wheel to function. Ten hours before the contests started we had a naïve steering that was enough for qualifying. Then the roller turned up broken, since we needed to paint it (it was yellow before), and panic arose. We had to put the rest of our

---

<sup>1</sup> For details on how to use `alphaLUT` and `distanceLUT` look at the source code in appendix A

effort to fix a new one, in the middle of the night before the contest. The extra wheel had to go. Thanks to another group, that lent us some of their trim, we managed to have a competing robot to the tournament (and we actually eliminated the same group (Lenna) from the first round).

Maybe if we had started with a naïve steering from the beginning, we could have implemented our primary idea and then perhaps made some advance upgrades to the steering by time. Now we had to restart the programming the last four days, and that is nothing we recommend.

At least we have tried all thinkable ways of steering and it turned out the simplest worked best.

Lesson learned: Avoid unnecessary complexity.

## References

- [1] <http://www.nada.kth.se/kurser/kth/2D1426/slides2006/proj1.pdf>, 2006-06-05
- [2] [http://en.wikipedia.org/wiki/The\\_30%25\\_Iron\\_Chef](http://en.wikipedia.org/wiki/The_30%25_Iron_Chef), 2006-06-05
- [3] <http://www.joker-robotics.com/eyebot/index.html>, 2006-06-05
- [4] <http://www.joker-robotics.com/eyebot/index.html#eyecam>, 2006-06-05
- [5] [http://en.wikipedia.org/wiki/Faraday\\_cage](http://en.wikipedia.org/wiki/Faraday_cage), 2006-06-05
- [6] The table and numbers can be found in the appended HDT-file, appendix A
- [7] Branl T. (2004), *Embedded Robotics : Mobile Robot Design and Applications with Embedded Systems*, Berlin: Springer
- [8] Siegwart R. and Nourbakhsh I. (2004), *Introduction to: Autonomous Mobile Robot*, The MIT Press: Cambridge
- [9] Ibid., pp. 84-86
- [10] Ibid., p. 52
- [11] <http://www.cs.umu.se/kurser/TDBB19/VT05/OH-bilder/Bezier.pdf>, 2006-06-05

Special thanks to group Lenna that saved us in our last hours of despair:

<http://www.nada.kth.se/kurser/kth/2D1426/reports2006/Lenna.pdf>

## Appendix A – Source Code

### Steering.h

```
#ifndef STEERING_H
#define STEERING_H

#include "robotMath.h"
#include "eyebot.h"

//Upper limit for the nr of jobs our steeringhandler
//can have on its queue
#define MAX_NR_OF_STEERING_OBJECTS 32

//Type of steeringobject
#define LINE 1
#define REVERSE 2
#define SPIN 3

//Is the coordinates relative to the robot or absolute in the inertial
frame
#define NA -1
#define ABSOLUTE 0
#define RELATIVE 1

//do we want to send any further information
//besides the fact that we are in the start state
//when a new object is added on the queue
#define NO_INFO 0
#define CALCULATE_ANGLE 1 << 1

#define START 1

//the global handle for the VW-drive system
//Tell the compiler that this variable will be available at link time
extern VWHandle vwhandle;

//Jobs that are sent to our steeringHandler
//are of type SteeringObject. These jobs
//control the robot's motion
typedef struct tagSteeringObject{
    short coordType;           //absolute or relative coordinates
    short id;                  //unique id for this steeringObject
    Vector2D pos;              //end position
    FIXP16 endHeading;        //end heading
    short driveType;          //do we want to rotate or drive along an
line
    short info;                //misc. info
}SteeringObject;

//The handler of execution of jobs concerning the motion of the robot
typedef struct tagSteeringHandler{
    SteeringObject* steeringQueue[MAX_NR_OF_STEERING_OBJECTS]; //the
job queue
```

```
    short nrOfObjects;
        //nr of jobs currently on the queue
}SteeringHandler;

//Tell the compiler that this variable will be available at link time
extern SteeringHandler steeringHandler;

//remove the first job in queue of the steeringHandler
void removeFirst(SteeringHandler* steeringHandler);

//remove all jobs on the queue of the steeringHandler
void clear(SteeringHandler* steeringHandler);

//add a new job to the steeringHandler
int addSteeringObject(short driveType, FIXP16 v1, FIXP16 v2, FIXP16
endHeading, short coordType, short additionalInfo);

//initialize a steeringObject
void initSteeringObject(short driveType, FIXP16 v1, FIXP16 v2, FIXP16
endHeading, SteeringObject* steeringObject, short coordType, short
additionalInfo);

//move the robot
int move(SteeringObject* pSteeringObject);

//handle the steering queue and start new jobs
void handleSteering();

#endif
```

## Steering.c

```
#include "steering.h"
#include "robotMath.h"
#include "simpleDrive.h"
#include "perception.h"

SteeringHandler steeringHandler;

float DRIVE_STRAIGHT_SPEED = 0.8; // W-Omega driving speed.
float TURNING_SPEED = 0.9; // W-Omega turning speed.

void handleSteering(){
    if(steeringHandler.nrOfObjects > 0){
        SteeringObject* steeringObject = steeringHandler.steeringQueue[0];
        // If we are done with this steering object
        // remove it
        if(move(steeringObject)){
            //Done
            VWSetSpeed(vwhandle, 0.0, 0.0);
            removeFirst(&steeringHandler);
        }
    }
}
```

```
int addSteeringObject(short driveType, FIXP16 v1, FIXP16 v2, FIXP16
endHeading, short coordType, short additionalInfo){

    SteeringObject* pSteeringObject =
    (SteeringObject*)malloc(sizeof(SteeringObject));
    initSteeringObject(driveType, v1, v2, endHeading, pSteeringObject,
coordType, additionalInfo);

    if(steeringHandler.nrofObjects < MAX_NR_OF_STEERING_OBJECTS){

        steeringHandler.steeringQueue[steeringHandler.nrofObjects] =
pSteeringObject;
        ++steeringHandler.nrofObjects;

        return 1;
    }else{
        return 0;
    }
}

void initSteeringObject(short driveType, FIXP16 v1, FIXP16 v2, FIXP16
endHeading, SteeringObject* steeringObject, short coordType, short
additionalInfo){
    static short uniqueID = 0;

    steeringObject->coordType = coordType;
    steeringObject->id = ++uniqueID;
    steeringObject->pos.v1 = v1;
    steeringObject->pos.v2 = v2;
    steeringObject->endHeading = endHeading;
    steeringObject->driveType = driveType;
    steeringObject->info = START | additionalInfo;
    steeringObject->angleToRobot = 0;
}

void clear(SteeringHandler* steeringHandler){
    VWSetSpeed(vwHandle,0,0);
    while(steeringHandler->nrofObjects > 0){
        removeFirst(steeringHandler);
    }
}

void removeFirst(SteeringHandler* steeringHandler){
    short i = 0;

    //if we don't have steeringObjects, return
    if(steeringHandler->nrofObjects == 0){
        return;
    }

    //if we only have one steering object. free the pointer,
    // set the entry to NULL and decrement the counter
    if(steeringHandler->nrofObjects == 1){
        free(steeringHandler->steeringQueue[0]);
    }
}
```

```

    steeringHandler->steeringQueue[0] = 0;
    --steeringHandler->nrOfObjects;
    return;
}

//free the first entry and set it to NULL
free(steeringHandler->steeringQueue[0]);
steeringHandler->steeringQueue[0] = 0;
for(i; i < steeringHandler->nrOfObjects-1; i++){
    //Move the remaining entries down one step
    steeringHandler->steeringQueue[i] = steeringHandler-
>steeringQueue[i+1];
}

//Set the last entry to NULL and decrement the pointer
steeringHandler->steeringQueue[steeringHandler->nrOfObjects] = 0;
--steeringHandler->nrOfObjects;
}

int move(SteeringObject* pSteeringObject){
    // do we want the robot to turn on the spot?
    if(pSteeringObject->driveType == SPIN){
        //Is it the first time this steering object is handled?
        if(pSteeringObject->info & START){
            if(pSteeringObject->info & CALCULATE_ANGLE){
                PositionType pos;
                VWGetPosition(vwhandle, &pos);

                FIXP16 dx1=pSteeringObject->pos.v1-pos.x*65536;
                FIXP16 dy1=pSteeringObject->pos.v2-pos.y*65536;
                FIXP16 v =-ATAN2(dy1,dx1) - pos.phi*65536;
                VWDriveTurn(vwhandle, (float)v/FIXP16_MAG, TURNING_SPEED);
            }else{
                VWDriveTurn(vwhandle, (float)pSteeringObject-
>endHeading/FIXP16_MAG, TURNING_SPEED);
            }
            pSteeringObject->info = 0;
        }
        //Are we done with this steering object
        else if(VWDriveDone(vwhandle)){
            return 1;
        }
        else{
            return 0;
        }
    }
    //do we want to drive forward along a line
    else if(pSteeringObject->driveType == LINE){
        //If we are stalling, break here and deal with it
        if(stallCheck()){
            return 0;
        }
        //Is it the first time this steering object is handled?
        if(pSteeringObject->info & START){
            pSteeringObject->info = 0;
            float distance = 0;

```

```

    //are the coordinates absolute
    if(pSteeringObject->coordType == ABSOLUTE){
        float x2 = (float)pSteeringObject->pos.v1/FIXP16_MAG;
        float y2 = (float)pSteeringObject->pos.v2/FIXP16_MAG;

        PositionType pos;
        VWGetPosition(vwhandle, &pos);

        float x1 = pos.x;
        float y1 = pos.y;

        distance = sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
    }
    //or are the coordinates relative
    else if(pSteeringObject->coordType == RELATIVE){
        distance = (float)pSteeringObject->pos.v1/FIXP16_MAG;
    }

    int res = VWDriveStraight(vwhandle, distance,
DRIVE_STRAIGHT_SPEED);
}
//Are we done with this steering object
else if(VWDriveDone(vwhandle)){
    return 1;
}
else{
    return 0;
}
}
//do we want to reverse along a line
else if(pSteeringObject->driveType == REVERSE){
    //Is it the first time this steering object is handled?
    if(pSteeringObject->info & START){
        pSteeringObject->info = 0;
        float distance = 0;
        //are the coordinates absolute
        if(pSteeringObject->coordType == ABSOLUTE){
            float x2 = (float)pSteeringObject->pos.v1/FIXP16_MAG;
            float y2 = (float)pSteeringObject->pos.v2/FIXP16_MAG;

            PositionType pos;
            VWGetPosition(vwhandle, &pos);

            float x1 = pos.x;
            float y1 = pos.y;

            distance = sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
        }
        //or are the coordinates relative
        else if(pSteeringObject->coordType == RELATIVE){
            distance = (float)pSteeringObject->pos.v1/FIXP16_MAG;
        }
        int res = VWDriveStraight(vwhandle, -distance,
DRIVE_STRAIGHT_SPEED);
    }
    //Are we done with this steering object
    else if(VWDriveDone(vwhandle)){

```

```
    return 1;
  }
  else{
    return 0;
  }
}

return 0;
}
```

## robotMath.h

```
#ifndef ROBOTMATH_H
#define ROBOTMATH_H

//defines for signed Fixed-Point math
typedef short FIXP8;
typedef int    FIXP16;

#define FIXP16_SHIFT      16
#define FIXP16_MAG        65536
#define FIXP16_DP_MASK    0x0000ffff
#define FIXP16_WP_MASK    0xffff0000
#define FIXP16_ROUND_UP   0x00008000
#define FLOAT_TO_FIXP16(x) ((int)(x*65536.0+0.5))
#define FIXP16_TO_INT(x)   (x >> FIXP16_SHIFT)

#define FIXP8_SHIFT       8
#define FIXP8_MAG          256
#define FIXP8_DP_MASK     0x00ff
#define FIXP8_WP_MASK     0xff00
#define FIXP8_ROUND_UP    0x0080
#define FLOAT_TO_FIXP8(x) ((short)(x*256.0+0.5))
#define FIXP8_TO_INT(x)   (x >> FIXP8_SHIFT)

//Fixed-Point PI defines
#define PI FLOAT_TO_FIXP16(3.14159265)
#define TWOPI FLOAT_TO_FIXP16(6.2831853)
#define HALFPI FLOAT_TO_FIXP16(1.5707963)
#define QUARTERPI FLOAT_TO_FIXP16(0.785398)

//Helpers for the Trigonometric function defines
#define NR_OF_COS_ELEMENTS 512
#define NR_OF_ATAN_ELEMENTS 512
#define NR_OF_TAN_ELEMENTS 512

#define ATAN_RANGE 10.0f
#define ATAN_INDEX_SCALE FLOAT_TO_FIXP16(10.0)

#define COS_WRAP_MASK (NR_OF_COS_ELEMENTS-1)
#define COS_LUT_INDEX(x)
FIXP16_TO_INT(divFIXP16(x,TWOPI)*NR_OF_COS_ELEMENTS)
#define ATAN_LUT_INDEX(x)
min(FIXP16_TO_INT(divFIXP16(x,ATAN_INDEX_SCALE)*NR_OF_ATAN_ELEMENTS),NR_OF_ATAN_ELEMENTS-1)
```

```

//Tell the compiler that these variables will be available at link
time
extern FIXP16 cosLut[];
extern FIXP16 atanLut[];
extern FIXP16 tanLut[];

//2-Demensional vector
typedef struct tagVector2D
{
  union{
    struct{
      FIXP16 v1, v2;
    };
    FIXP16 v[2];
  };
}Vector2D;

//3-Demensional vector
typedef struct tagVector3D
{
  union{
    struct{
      FIXP16 v1, v2, v3;
    };
    FIXP16 v[3];
  };
}Vector3D;

//3-Demensional matrix
typedef struct tagMatrix3D
{
  union {
    struct{
      FIXP16 m11, m12, m13;
      FIXP16 m21, m22, m23;
      FIXP16 m31, m32, m33;
    };
    FIXP16 m[3][3];
  };
}Matrix3D;

#define min(a, b) (a < b? a : b)
#define max(a, b) (a > b? a : b)
#define abs(x) (x < 0 ? -x : x)
#define sgn(x) (x < 0 ? -1 : 1)

//defines for the Trigonometric functions atan, atan2, tan, cos, sin
#define ATAN(x) (x < 0 ? -atanLut[ATAN_LUT_INDEX(-(x))] :
atanLut[ATAN_LUT_INDEX(x)])
#define ATAN2(y, x) ( x < 0 ? ATAN(divFIXP16(y , x)) + PI : (x == 0 ?
(y < 0? -HALFPI : HALFPI) : ATAN(divFIXP16(y , x)) ))
#define TAN(x) ( x < 0 ? -tanLut[FIXP16_TO_INT(divFIXP16(-x ,
QUARTERPI) * NR_OF_TAN_ELEMENTS)] : tanLut[FIXP16_TO_INT(divFIXP16(x ,
QUARTERPI) * NR_OF_TAN_ELEMENTS)])

```

```
#define SIN(x) ( x < 0 ? -cosLut[COS_LUT_INDEX(-(x)) +  
COS_LUT_INDEX(PI+HALFPI) & COS_WRAP_MASK] : cosLut[COS_LUT_INDEX((x)) +  
COS_LUT_INDEX(PI+HALFPI) & COS_WRAP_MASK])  
#define COS(x) ( x < 0 ? cosLut[COS_LUT_INDEX(-(x)) & COS_WRAP_MASK] :  
cosLut[COS_LUT_INDEX((x)) & COS_WRAP_MASK])  
  
//set all the elements of the vector to zero  
void setZeroVector3D(Vector3D* out_pVec);  
//set all the elements of the matrix to zero  
void setZeroMatrix3D(Matrix3D* out_pMatrix);  
//multiply a 3D matrix with a 3D vector. The result is returned in  
out_pVector  
void matrix3x3MulVector3x1(Matrix3D in_matrix, Vector3D in_vector,  
Vector3D* out_pVector);  
//build an Inverse rotation matrix. The matrix is returned in  
out_pMatrix  
void buildInverseRotMatrix(Matrix3D* out_pMatrix, FIXP16 in_theta);  
//build an rotation matrix. The matrix is returned in out_pMatrix  
void buildRotMatrix(Matrix3D* out_pMatrix, FIXP16 in_theta);  
  
//Fixed-Point functions for multiplication, division and square root  
FIXP16 mulFIXP16(FIXP16 fp1, FIXP16 fp2);  
FIXP16 divFIXP16(FIXP16 fp1, FIXP16 fp2);  
  
FIXP8 mulFIXP8(FIXP8 fp1, FIXP8 fp2);  
FIXP8 divFIXP8(FIXP8 fp1, FIXP8 fp2);  
  
FIXP16 sqrtFIXP16(FIXP16 a);  
  
//the dot product of two vectors v1 and v2  
FIXP16 dot2D(Vector2D v1, Vector2D v2);  
//the projection of v1 on to v1. The answer is returned in proj  
void proj2D(Vector2D v1, Vector2D v2, Vector2D* proj);  
//the sqaure length of vector v  
FIXP16 lengthSquare2D(Vector2D v);  
//the length of vector v  
FIXP16 length2D(Vector2D v);  
//normalize vector v. The normalized vector is returned in n  
FIXP16 normalize2D(Vector2D v, Vector2D* n);  
  
#endif
```

## robotMath.c

```
#include "robotMath.h"  
  
void setZeroVector3D(Vector3D* out_pVec){  
    out_pVec->v1 = 0;  
    out_pVec->v2 = 0;  
    out_pVec->v3 = 0;  
}  
  
void setZeroMatrix3D(Matrix3D* out_pMatrix){
```

```

    int i = 0;
    int j = 0;
    for(i = 0; i < 3; ++i){
        for(j = 0; j < 3; ++j){
            out_pMatrix->m[i][j] = 0;
        }
    }
}

FIXP16 dot2D(Vector2D v1, Vector2D v2){
    return mulFIXP16(v1.v1 , v2.v1) + mulFIXP16(v1.v2 , v2.v2);
}

void proj2D(Vector2D v1, Vector2D v2, Vector2D* proj){
    proj->v1 = mulFIXP16(divFIXP16(dot2D(v2,v1) , dot2D(v2 , v2)) ,
v2.v1);
    proj->v2 = mulFIXP16(divFIXP16(dot2D(v2,v1) , dot2D(v2 , v2)) ,
v2.v2);
}

FIXP16 lengthSquare2D(Vector2D v){
    return mulFIXP16(v.v1 , v.v1) + mulFIXP16(v.v2 , v.v2);
}

FIXP16 length2D(Vector2D v){
    return sqrtFIXP16(mulFIXP16(v.v1 , v.v1) + mulFIXP16(v.v2 , v.v2));
}

FIXP16 normalize2D(Vector2D v, Vector2D* n){
    FIXP16 l = length2D(v);
    n->v1 = divFIXP16(v.v1 , l);
    n->v2 = divFIXP16(v.v2 , l);
}

//use 32-bit math so we don't loose any precision
FIXP8 mulFIXP8(FIXP8 fp1, FIXP8 fp2){
    FIXP16 tempResult;
    FIXP8 result;

    tempResult = ((int)(fp1)*(fp2));

    result = (tempResult&0xff00)>>FIXP8_SHIFT;
    result |= (tempResult&0x00ff0000)>>FIXP8_SHIFT;

    return result;
}

//use 32-bit math so we don't loose any precision
FIXP8 divFIXP8(FIXP8 fp1, FIXP8 fp2){
    FIXP16 tempNumerator;
    FIXP8 result;

    tempNumerator = ((int)fp1<<FIXP8_SHIFT);

    result = tempNumerator/fp2;
    return result;
}

```

```
}

//use 64-bit math so we don't loose any precision
FIXP16 mulFIXP16(FIXP16 fp1, FIXP16 fp2){
    FIXP16 result = 0;
    long long tempResult = (long long)fp1 * (long long)fp2;
    result = tempResult >> FIXP16_SHIFT;
    return result;
}

//use 64-bit math so we don't loose any precision
FIXP16 divFIXP16(FIXP16 fp1, FIXP16 fp2){
    long long tempNumerator;
    FIXP16 result;

    tempNumerator = ((long long)fp1<<FIXP16_SHIFT);
    result = tempNumerator/fp2;

    return result;
}

void matrix3x3MulVector3x1(Matrix3D in_matrix, Vector3D in_vector,
                           Vector3D* out_pVector){
    FIXP16 tempVal = 0;
    int i = 0;
    int j = 0;
    for(i = 0; i < 3; ++i){
        for(j = 0; j < 3; ++j){
            tempVal += mulFIXP16(in_matrix.m[i][j] , in_vector.v[j]);
        }
        out_pVector->v[i] = tempVal;
        tempVal = 0;
    }
}

void buildInverseRotMatrix(Matrix3D* out_pMatrix, FIXP16 in_theta){
    out_pMatrix->m11 = COS(in_theta);
    out_pMatrix->m12 = -SIN(in_theta);
    out_pMatrix->m13 = 0;

    out_pMatrix->m21 = SIN(in_theta);
    out_pMatrix->m22 = COS(in_theta);
    out_pMatrix->m23 = 0;

    out_pMatrix->m31 = 0;
    out_pMatrix->m32 = 0;
    out_pMatrix->m33 = 1 << FIXP16_SHIFT;
}

void buildRotMatrix(Matrix3D* out_pMatrix, FIXP16 in_theta){
    out_pMatrix->m11 = COS(in_theta);
    out_pMatrix->m12 = SIN(in_theta);
    out_pMatrix->m13 = 0;
}
```

```
    out_pMatrix->m21 = -SIN(in_theta);
    out_pMatrix->m22 = COS(in_theta);
    out_pMatrix->m23 = 0;

    out_pMatrix->m31 = 0;
    out_pMatrix->m32 = 0;
    out_pMatrix->m33 = 1 << FIXP16_SHIFT;
}

FIXP16 getAngle(Vector2D p1, Vector2D p2){
    return ATAN2(p2.v2-p1.v2, p2.v1-p1.v1);
}

FIXP16 sqrtFIXP16(FIXP16 x){
    register unsigned long root, remHi, remLo, testDiv, count;

    root = 0;           // Clear root
    remHi = 0;          // Clear high part of partial remainder
    remLo = x;          // Get argument into low part of partial
remainder
    count = (15 + (FIXP16_SHIFT >> 1)); // Load loop counter

    do {
        remHi = (remHi << 2) | (remLo >> 30); // get 2 bits of arg
        remLo <<= 2;
        root <<= 1; // Get ready for the next bit in the root
        testDiv = (root << 1) + 1; // Test radical
        if (remHi >= testDiv) {
            remHi -= testDiv;
            root += 1;
        }
    } while (count-- != 0);

    return(root);
}
```

## Perception.h

```
#ifndef PERCEPTION_H
#define PERCEPTION_H

#include "eyebot.h"
#include "robotMath.h"

//Tell the compiler that these variables will be available at link
time
extern int yLut[];
extern int alphaLut[];
extern short objectLut[];

extern BYTE *imgRGB;

//The objects that can be identified
#define BALL 1
#define BLUE_GOAL 2
```

```
#define YELLOW_GOAL 3
#define OPPONENT 4

//The index to the objects in the object array
#define BALL_INDEX 0
#define BLUE_GOAL_INDEX 1
#define YELLOW_GOAL_INDEX 2
#define OPPONENT_INDEX 3

#define NR_OF_OBJECTS 4

//The location of each object is kept in a ObjectLoc
typedef struct tagObjectLoc
{
    short step;
    Vector2D pos;
    short found;
    FIXP16 distance;
    unsigned short count;
    unsigned short sumX;
    unsigned short sumY;
    unsigned short meanX;
    unsigned short meanY;
    FIXP16 alpha;
    BYTE detectablePixels;
}ObjectLoc;

//Tell the compiler that these variables will be available at link
time
extern ObjectLoc opponentGoalLoc;
extern ObjectLoc* targetLoc;

//Identify and locate the objects in the current FOV
void locateObjects(image* lcdImage);
#endif
```

## Perception.c

```
#include "perception.h"
#include "eyebot.h"
#include "robotMath.h"
#include "simpleDrive.h"
#include "steering.h"

colimage frameImage;
BYTE *imgRGB;

ObjectLoc opponentGoalLoc;
ObjectLoc* targetLoc;

void locateObjects(image* lcdImage){

    int i, j, i2, j2;
    short class = -1;

    //loop through the image only checking every other pixel to save time
```

```

//count the occurrences of each object and sum the x and y coordinates
for(i=0, i2=1; i<62; i++, i2+=2){
  for (j=0, j2=3; j<82; j++, j2+=6) {
    int lutIndex = ((imgRGB[i2*176*3+j2 ] & 0xF8) << 7) |
      ((imgRGB[i2*176*3+j2+1] & 0xF8) << 2) |
      ((imgRGB[i2*176*3+j2+2] & 0xF8) >> 3);
    class = objectLut[lutIndex];
    if(class > 0 && class <= NR_OF_OBJECTS){
      targetLoc[class-1].sumX = targetLoc[class-1].sumX + j;
      targetLoc[class-1].sumY = targetLoc[class-1].sumY + i;
      if(class == BLUE_GOAL || class == BALL)
        (*lcdImage)[i][j] = 255;
      else
        (*lcdImage)[i][j] = 0;
      ++targetLoc[class-1].count;
    }
    else{
      (*lcdImage)[i][j] = 0;
    }
  }
}

//calculate the location the each one of the locates objects
i = 0;
for(i; i < NR_OF_OBJECTS; i++){
  if(targetLoc[i].count == 0){
    continue;
  }
  targetLoc[i].meanX = targetLoc[i].sumX/targetLoc[i].count;
  targetLoc[i].meanY = targetLoc[i].sumY/targetLoc[i].count;

  if(targetLoc[i].count > targetLoc[i].detectablePixels){ // If the
number of target pixels is enough

    targetLoc[i].found = 1;

    FIXP16 alpha = alphaLut[targetLoc[i].meanX];
    FIXP16 x = yLut[targetLoc[i].meanY];
    FIXP16 y = mulFIXP16(TAN(alpha) , x);

    FIXP16 xi1 = mulFIXP16(x, COS(pos.phi*65536));
    FIXP16 yi1 = mulFIXP16(x, SIN(pos.phi*65536));
    FIXP16 xi2 = mulFIXP16(y, -SIN(pos.phi*65536));
    FIXP16 yi2 = mulFIXP16(y, COS(pos.phi*65536));

    targetLoc[i].pos.v1 = xi1 + xi2 + pos.x*65536;
    targetLoc[i].pos.v2 = yi1 + yi2 + pos.y*65536;
    targetLoc[i].distance = x;
    targetLoc[i].alpha = alpha;
  }
}
}
//-----//

```

```
// RESET TARGET LOCS //
//-----//
void resetTargetLocs(){
    targetLoc[BALL_INDEX].step = 0;
    targetLoc[BALL_INDEX].screenX = 0;
    targetLoc[BALL_INDEX].screenY = 0;
    targetLoc[BALL_INDEX].pos.v1 = 0;
    targetLoc[BALL_INDEX].pos.v2 = 0;
    targetLoc[BALL_INDEX].found = 0;
    targetLoc[BALL_INDEX].distance = 0;
    targetLoc[BALL_INDEX].count = 0;
    targetLoc[BALL_INDEX].sumX = 0;
    targetLoc[BALL_INDEX].sumY = 0;
    targetLoc[BALL_INDEX].meanX = 0;
    targetLoc[BALL_INDEX].meanY = 0;
    targetLoc[BALL_INDEX].detectablePixels = 2;

    targetLoc[BLUE_GOAL_INDEX].step = 0;
    targetLoc[BLUE_GOAL_INDEX].screenX = 0;
    targetLoc[BLUE_GOAL_INDEX].screenY = 0;
    targetLoc[BLUE_GOAL_INDEX].pos.v1 = 0;
    targetLoc[BLUE_GOAL_INDEX].pos.v2 = 0;
    targetLoc[BLUE_GOAL_INDEX].found = 0;
    targetLoc[BLUE_GOAL_INDEX].distance = 0;
    targetLoc[BLUE_GOAL_INDEX].count = 0;
    targetLoc[BLUE_GOAL_INDEX].sumX = 0;
    targetLoc[BLUE_GOAL_INDEX].sumY = 0;
    targetLoc[BLUE_GOAL_INDEX].meanX = 0;
    targetLoc[BLUE_GOAL_INDEX].meanY = 0;
    targetLoc[BLUE_GOAL_INDEX].detectablePixels = 30;

    targetLoc[YELLOW_GOAL_INDEX].step = 0;
    targetLoc[YELLOW_GOAL_INDEX].screenX = 0;
    targetLoc[YELLOW_GOAL_INDEX].screenY = 0;
    targetLoc[YELLOW_GOAL_INDEX].pos.v1 = 0;
    targetLoc[YELLOW_GOAL_INDEX].pos.v2 = 0;
    targetLoc[YELLOW_GOAL_INDEX].found = 0;
    targetLoc[YELLOW_GOAL_INDEX].distance = 0;
    targetLoc[YELLOW_GOAL_INDEX].count = 0;
    targetLoc[YELLOW_GOAL_INDEX].sumX = 0;
    targetLoc[YELLOW_GOAL_INDEX].sumY = 0;
    targetLoc[YELLOW_GOAL_INDEX].meanX = 0;
    targetLoc[YELLOW_GOAL_INDEX].meanY = 0;
    targetLoc[YELLOW_GOAL_INDEX].detectablePixels = 40;

    targetLoc[OPPONENT_INDEX].step = 0;
    targetLoc[OPPONENT_INDEX].screenX = 0;
    targetLoc[OPPONENT_INDEX].screenY = 0;
    targetLoc[OPPONENT_INDEX].pos.v1 = 0;
    targetLoc[OPPONENT_INDEX].pos.v2 = 0;
    targetLoc[OPPONENT_INDEX].found = 0;
    targetLoc[OPPONENT_INDEX].distance = 0;
    targetLoc[OPPONENT_INDEX].count = 0;
    targetLoc[OPPONENT_INDEX].sumX = 0;
    targetLoc[OPPONENT_INDEX].sumY = 0;
    targetLoc[OPPONENT_INDEX].meanX = 0;
    targetLoc[OPPONENT_INDEX].meanY = 0;
}
```

```
    targetLoc[OPPONENT_INDEX].detectablePixels = 20;  
  
}
```

## simpleDrive.h

```
#ifndef SIMPLEDRIVE_H  
#define SIMPLEDRIVE_H  
  
#include "eyebot.h"  
#include "robotMath.h"  
  
//defines used for interval reduction when the robot starts  
//to oscillate between to angles on opposite sides of a goal  
#define LOCATE_GOAL_NO_TURN 1  
#define LOCATE_GOAL_LEFT_TURN 2  
#define LOCATE_GOAL_RIGHT_TURN 3  
  
//the states that the state machine can be in  
#define SEARCH_BALL 1 << 1  
#define DRIVE_TO_BALL 1 << 2  
#define SCORE 1 << 3  
#define SCORE_SECOND_TRY 1 << 4  
#define START_STATE 1 << 5  
#define BUMP_STATE 1 << 6  
  
//state of the state machine  
//Tell the compiler that this variable will be available at link time  
extern int state;  
  
//Flags that correspond to bumper states  
//Tell the compiler that these variables will be available at link  
time  
extern int resbl;  
extern int resbr;  
extern int resfl;  
extern int resfr;  
  
//the main loop  
void mainLoop();  
  
//initilize the camera  
void initCamera(void);  
//release the camera  
void releaseCamera();  
  
//plot the targeted object on the LCD-screen  
int displayImage(int key);  
//mark the position of the targeted object  
void drawCross(int row, int col);  
  
//reset the information from the former picture and take a new one  
void takePicture();  
  
//Check if we are stalling
```

```
int stallCheck();  
//Check if any bumper is registering a hit  
int bumpCheck();  
  
void playSound();  
  
#endif
```

## simpleDrive.c

```
#include "simpleDrive.h"  
#include "robotMath.h"  
#include "steering.h"  
#include "perception.h"  
#include "bumpdefs.h"  
  
// VW  
VWHandle vwHandle;  
PositionType vwpos;  
  
image lcdImage;          // gray-image to display when testing robot.  
  
MotorWrapperStruct motorLeft;  
MotorWrapperStruct motorRight;  
  
ServoHandle serv0;  
  
static int time = -1;  
static int lastTime = -1;  
  
int resbl = -1;  
int resbr = -1;  
int resfl = -1;  
int resfr = -1;  
  
// BUMPER DECLARATION //  
BumpHandle backLeftBumper, backRightBumper, frontLeftBumper,  
frontRightBumper;  
  
int timeStampl = 0;  
int timeStampr = 0;  
int timeStampfl = 0;  
int timeStampfr = 0;  
  
//stall check variables  
int stallStart = 0;  
int stallTime = 0;  
int startStallTime = 1;  
int hr = 0, min = 0, sec = 0, ticks = 0;  
int dTicks = 0, dSecs = 0;  
int startHr = 0, startMin = 0, startSec = 0, startTicks = 0;  
  
//bump check variables  
int bumpStallStart = 0;  
int bumpStallTime = 0;  
int bumpStartStallTime = 1;
```

```
int bumpHr = 0, bumpMin = 0, bumpSec = 0, bumpTicks = 0;
int bumpDTicks = 0, bumpDsecs = 0;
int bumpStartHr = 0, bumpStartMin = 0, bumpStartSec = 0, bumpStartTicks
= 0;

//flag that tells us if we are localized or not
int delocalized = 0;

//state machine variable
int state = START_STATE;
int searchBallFirst = 0;
int driveToBallFirst = 0;
int scoreFirst = 0;
int scoredFirst = 0;
int scoreSecondTryFirst = 0;
int startFirst = 0;
int bumpFirst = 0;

int lastState = 0;

int goalCount = 0;
FIXP16 turnAngle = 0;
FIXP16 lastAngle = 0;

FIXP16 locateGoalStepSize = QUARTERPI>>1;
short lastTurningDirection = LOCATE_GOAL_NO_TURN;

//variables that are set in the beginning of the program to tell us
//which goal to score on
int goalToApproachIndex = BLUE_GOAL_INDEX;
int opponentGoalIndex = BLUE_GOAL_INDEX;
int ownGoalIndex = YELLOW_GOAL_INDEX;

//-----//
//  MAIN  //
//-----//
int main(){
    targetLoc = (ObjectLoc*)malloc(sizeof(ObjectLoc)*NR_OF_OBJECTS);

    imgRGB      = (BYTE*)malloc(sizeof(BYTE)*176*144*3); // 144*176 image,
from perception.c.

    //the location of the opponents goal is fixed and very valuable to us
    opponentGoalLoc.pos.v1 = FLOAT_TO_FIXP16(2.4);
    opponentGoalLoc.pos.v2 = FLOAT_TO_FIXP16(0);

    LCDMode(SCROLLING|NOCURSOR);
    LCDClear();

    // INIT BUMPERS //
    backLeftBumper = BUMPInit(BUMP_LEFT);
    backRightBumper = BUMPInit(BUMP_RIGHT);
    frontLeftBumper = BUMPInit(BUMP_LEFT_FRONT);
    frontRightBumper = BUMPInit(BUMP_RIGHT_FRONT);
```

```
// BUMPER ERROR //
if( frontLeftBumper == 0 ) {
    OSErrror("KAN EJ INITIERA F_LEFT / Kasst!", 4133,0);
}
if( frontRightBumper == 0 ){
    OSErrror("KAN EJ INITIERA F_RIGHT / Kasst!", 4133,0);
}
if( (0 == backLeftBumper) ){
    OSErrror("KAN EJ INITIERA LEFT / Kasst!", 4133,0);
}
if( backRightBumper == 0 ){
    OSErrror("KAN EJ INITIERA RIGHT / Kasst!", 4133,0);
}

initCamera();

//VW
vwhandle = VWInit(VW_DRIVE, 2);
VWStartControl(vwhandle, 7.0, 0.3, 7.0, 0.1);
int res = VWSetPosition(vwhandle, 0.0, 0.0, 0.0);
if(res == -1)
    OSErrror("VWSetPosition error:", 4133, 0);

//Wait for autobrightness
OSWait(100);

LCDMenu("BGL", "YGL", "...", "...");
int key = KEYGet();
if(key == KEY1){
    opponentGoalIndex = BLUE_GOAL_INDEX;
    ownGoalIndex = YELLOW_GOAL_INDEX;
}
else{
    opponentGoalIndex = YELLOW_GOAL_INDEX;
    ownGoalIndex = BLUE_GOAL_INDEX;
}

goalToApproachIndex = opponentGoalIndex;

mainLoop();

//VW - stop the omega drive before exit.
VWSetSpeed(vwhandle, 0.0, 0.0);
VWDriveStraight(vwhandle, 0, 0.1);
while(!VWDriveDone(vwhandle))
    OSWait(20);
VWStopControl(vwhandle);
VWRelease(vwhandle);

releaseCamera();

// RELEASE BUMPERS //
BUMPRRelease(frontLeftBumper);
BUMPRRelease(frontRightBumper);
BUMPRRelease(backLeftBumper);
BUMPRRelease(backRightBumper);
free(imgRGB);
```

```
    return 0;
}

//-----//
//  MAIN LOOP  //
//-----//
void mainLoop(){
    int key = -1;
    int target = 0;

    int dist = FLOAT_TO_FIXP16(0.5);
    int zero = FLOAT_TO_FIXP16(0.01);

    while((key = KEYRead()) != KEY4){

        // READ BUMPERS //
        resbl = BUMPCheck( backLeftBumper , &timeStamp1);
        resbr = BUMPCheck( backRightBumper , &timeStampR);
        resfl = BUMPCheck( frontLeftBumper , &timeStampfl);
        resfr = BUMPCheck( frontRightBumper , &timeStampfr);

        if(0 == resfl){ // BUMP OCCURED //
            AUTone(3100, 20);
            LCDSetPrintf(5, 0, "FRONT LEFT");
        }
        if(0 == resfr){ // BUMP OCCURED //
            AUTone(4200, 20);
            LCDSetPrintf(5, 0, "FRONT RIGHT");
        }
        if(0 == resbl){ // BUMP OCCURED //
            AUTone(5300, 20);
            LCDSetPrintf(5, 0, "LEFT");
        }
        if(0 == resbr){ // BUMP OCCURED //
            AUTone(6400, 20);
            LCDSetPrintf(5, 0, "RIGHT");
        }
    }

    if(bumpCheck() && state != BUMP_STATE && state != SCORE){
        lastState = state;
        state = BUMP_STATE;
    }

    switch (state){
    case START_STATE:{
        PositionType pos;
        VWGetPosition(vwhandle, &pos);
        LCDSetPrintf(2,0, "%f",pos.x);
        LCDSetPrintf(3,0, "%f",pos.y);
        LCDClear();
        LCDSetPrintf(0, 0, "START_STATE");
        if(startFirst == 0){
            FIXP16 y = FLOAT_TO_FIXP16(0.8);

```

```

    FIXP16 x = FLOAT_TO_FIXP16(0);
    addSteeringObject(LINE,x,y, NA, ABSOLUTE, NO_INFO );
    startFirst = 1;
  }

  if(steeringHandler.nrofObjects == 0){
    lastState = START_STATE;
    state = SEARCH_BALL;
    startFirst = 0;
  }
}break;

//locate the ball
case SEARCH_BALL:{
  LCDClear();
  LCDSetPrintf(0, 0, "SEARCH_BALL");
  if(searchBallFirst == 0){
    locateGoalStepSize = QUARTERPI>>1;
    lastTurningDirection = LOCATE_GOAL_NO_TURN;
    searchBallFirst = 1;
  }
  PositionType pos;
  VWGetPosition(vwhandle, &pos);
  LCDSetPrintf(2,0, "%f",pos.x);
  LCDSetPrintf(3,0, "%f",pos.y);
  if(steeringHandler.nrofObjects == 0){
    takePicture();
    if(targetLoc[goalToApproachIndex].found == 1 && (goalCount == 0
|| (turnAngle - lastAngle) < -(3*PI>>1))){
      goalCount++;
      lastAngle = turnAngle;
    }
    //This is also done after we have turned around while deciding
which goal to approach
    //while searching for the ball
    if(targetLoc[BALL_INDEX].found == 1){
      if(targetLoc[BALL_INDEX].distance < FLOAT_TO_FIXP16(0.055)){
        lastState = SEARCH_BALL;
        if(delocalized){
          state = SCORE_SECOND_TRY;
        }else{
          state = SCORE;
        }
      }
      searchBallFirst = 0;
    }else{
      LCDSetPrintf(1, 0, "SEARCH_BALL 1");

      lastState = SEARCH_BALL;
      state = DRIVE_TO_BALL;
      searchBallFirst = 0;
      goalCount = 0;
      lastAngle = 0;
    }
  }else{
    LCDSetPrintf(1, 0, "SEARCH_BALL 2");
    if (goalCount < 2){
      addSteeringObject(SPIN,NA,NA,-QUARTERPI,NA, NO_INFO);

```

```

    turnAngle+=-QUARTERPI;
  }else if(turnAngle < -(TWOPI * 3)){
    addSteeringObject(LINE, FLOAT_TO_FIXP16(0.6), 0, NA, RELATIVE,
NO_INFO);
    turnAngle = 0;
    goalCount = 0;
    lastAngle = 0;
  }else{
    PositionType pos;
    VWGetPosition(vwHandle, &pos);
    if(targetLoc[goalToApproachIndex].found != 1){
      addSteeringObject(SPIN, NA, NA, -QUARTERPI, NA, NO_INFO);
      turnAngle+=-QUARTERPI;
    }
    //if we have found the goal, turn towards it
    else if(targetLoc[goalToApproachIndex].found == 1 &&
targetLoc[goalToApproachIndex].meanX >42
      && locateGoalStepSize >= QUARTERPI>>5){

      if(lastTurningDirection == LOCATE_GOAL_RIGHT_TURN){
        locateGoalStepSize = locateGoalStepSize>>1;
      }
      lastTurningDirection = LOCATE_GOAL_LEFT_TURN;
      addSteeringObject(SPIN, NA, NA, QUARTERPI>>3, NA,
NO_INFO);
      turnAngle+=QUARTERPI>>3;
    }
    else if(targetLoc[goalToApproachIndex].found == 1 &&
targetLoc[goalToApproachIndex].meanX <39
      && locateGoalStepSize >= QUARTERPI>>5){
      if(lastTurningDirection == LOCATE_GOAL_LEFT_TURN){
        locateGoalStepSize = locateGoalStepSize>>1;
      }
      lastTurningDirection = LOCATE_GOAL_RIGHT_TURN;
      addSteeringObject(SPIN, NA, NA,-QUARTERPI>>3 , NA,
NO_INFO);
      turnAngle+=-QUARTERPI>>3;
    }
    //if we have found the goal and are facing it, try to score
    else{
      turnAngle=0;
      goalCount = 0;
      lastAngle = 0;
      //Are we to close to goal?
      if(targetLoc[goalToApproachIndex].count > 700){

        //Turn around
        if(goalToApproachIndex == opponentGoalIndex){
          goalToApproachIndex = ownGoalIndex;
        }else{
          goalToApproachIndex = opponentGoalIndex;
        }
        addSteeringObject(SPIN, NA, NA, PI, NA, NO_INFO);
        //Take a new photo just in case
        takePicture();
        //Did we find the ball?
        if(targetLoc[BALL_INDEX].found == 1){

```

```

        if(targetLoc[BALL_INDEX].distance <
FLOAT_TO_FIXP16(0.055)){
            lastState = SEARCH_BALL;
            if(delocalized){
                state = SCORE_SECOND_TRY;
            }else{
                state = SCORE;
            }
            searchBallFirst = 0;
        }else{
            LCDSetPrintf(1, 0, "SEARCH_BALL 1");

            lastState = SEARCH_BALL;
            state = DRIVE_TO_BALL;
            searchBallFirst = 0;
            goalCount = 0;
            lastAngle = 0;
        }
    } //if targetLoc[BALL_INDEX].found == 1
} //targetLoc[goalToApproachIndex].count > 1000

//start approaching the other goal
addSteeringObject(LINE, FLOAT_TO_FIXP16(0.5), 0, NA,
RELATIVE, NO_INFO);
} //else
} //else SEARCH BALL 2
} //if steeringHandler.nrOfObjects == 0
} else{
    LCDSetPrintf(1, 0, "SEARCH_BALL 3");
}
}break;

case SCORE:{
    LCDClear();
    LCDSetPrintf(0, 0, "SCORE");
    PositionType pos;
    VWGetPosition(vwhandle, &pos);
    LCDSetPrintf(2,0, "%f",pos.x);
    LCDSetPrintf(3,0, "%f",pos.y);
    if(scoreFirst == 0){

        checkForGoalScored = 0;
        PositionType pos;
        VWGetPosition(vwhandle, &pos);
        FIXP16 dx1=blueGoalLoc.pos.v1-pos.x*65536;
        FIXP16 dy1=blueGoalLoc.pos.v2-pos.y*65536;
        FIXP16 v = (+ATAN2(dy1,dx1) - pos.phi*65536);

        addSteeringObject(SPIN, NA, NA, v, NA, NO_INFO);
        addSteeringObject(LINE, blueGoalLoc.pos.v1, blueGoalLoc.pos.v2,
0, ABSOLUTE, NO_INFO);

        scoreFirst = 1;
    }
    if(resbl == 0 || resbr == 0){
        lastState = SCORE;
        state = BUMP_STATE;
    }
}

```

```

    scoreFirst = 0;
  }
  //Have we hit the wall before we think that we have arrived at
the goal?
  if(resfl == 0 || resfr == 0){
    delocalized = 1;
    //Stop
    clear(&steeringHandler);
    OSWait(30);
    addSteeringObject(REVERSE,FLOAT_TO_FIXP16(0.30) , 0, 0,
RELATIVE, NO_INFO);
    lastState = SCORE;
    state = SEARCH_BALL;
    scoreFirst = 0;
  }
  else if(steeringHandler.nrofObjects == 0){
    delocalized = 1;
    lastState = SCORE;
    state = SEARCH_BALL;
    scoreFirst = 0;
  }
}break;

case SCORED:{
  LCDClear();
  LCDSetPrintf(0, 0, "SCORED");
}break;

case SCORE_SECOND_TRY:{
  LCDClear();
  LCDSetPrintf(0, 0, "SCORE_SECOND_TRY");
  PositionType pos;
  VWGetPosition(vwhandle, &pos);
  LCDSetPrintf(2,0, "%f",pos.x);
  LCDSetPrintf(3,0, "%f",pos.y);
  delocalized = 1;
  if(scoreSecondTryFirst == 0){
    scoreSecondTryTurnAngle = 0;
    locateGoalStepSize = QUARTERPI>>1;
    lastTurningDirection = LOCATE_GOAL_NO_TURN;
    clear(&steeringHandler);
    if(lastState != SEARCH_BALL && lastState != BUMP_STATE){
      PositionType pos;
      VWGetPosition(vwhandle, &pos);
      Vector2D endPoint;
      endPoint.v1 = pos.x*65536 - FLOAT_TO_FIXP16(0.5);
      endPoint.v2 = 0;

      FIXP16 dx1= -FLOAT_TO_FIXP16(0.5);
      FIXP16 dy1= -pos.y*65536;
      FIXP16 v = (ATAN2(dy1,dx1) - pos.phi*65536)-PI;
      if(v > PI){
        v = v - TWOPI;
      }else if(v < -PI){
        v = v + TWOPI;
      }
    }
  }
}

```

```

    addSteeringObject(SPIN, NA, NA, v, NA, NO_INFO);
    addSteeringObject(REVERSE, endPoint.v1, endPoint.v2, NA,
ABSOLUTE, NO_INFO);
  }
  scoreSecondTryFirst = 1;
}

if(steeringHandler.nrofObjects == 0){
  takePicture();
  if(targetLoc[BALL_INDEX].found == 1 &&
targetLoc[BALL_INDEX].distance <= FLOAT_TO_FIXP16(0.055)){
    //if we have lost the blue goal find it again
    if(targetLoc[opponentGoalIndex].found != 1){
      AUTone(3000,100);
      OSWait(10);
      AUTone(100,100);
      OSWait(10);
      AUTone(100, 100);
      if(scoreSecondTryTurnAngle < -5*PI &&
targetLoc[ownGoalIndex].found != 1){
        addSteeringObject(LINE, FLOAT_TO_FIXP16(0.5), 0, NA,
RELATIVE, NO_INFO);
      }else{
        addSteeringObject(SPIN, NA, NA, -QUARTERPI, NA, NO_INFO);
        scoreSecondTryTurnAngle+=-QUARTERPI;
      }
    }
    //if we have found the goal, turn LEFT towards it
    else if(targetLoc[opponentGoalIndex].found == 1 &&
targetLoc[opponentGoalIndex].meanX > 42
      && locateGoalStepSize >= QUARTERPI>>5){
      AUTone(10000,100);
      OSWait(10);
      AUTone(1000,100);
      OSWait(10);
      AUTone(100, 100);

      if(lastTurningDirection == LOCATE_GOAL_RIGHT_TURN){
        locateGoalStepSize = locateGoalStepSize>>1;
      }
      lastTurningDirection = LOCATE_GOAL_LEFT_TURN;
      addSteeringObject(SPIN, NA, NA, locateGoalStepSize, NA,
NO_INFO);
    }
    else if(targetLoc[opponentGoalIndex].found == 1 &&
targetLoc[opponentGoalIndex].meanX < 39
      && locateGoalStepSize >= QUARTERPI>>5){
      AUTone(1000,100);
      OSWait(10);
      AUTone(10000,100);
      OSWait(10);
      AUTone(100, 100);
      if(lastTurningDirection == LOCATE_GOAL_LEFT_TURN){
        locateGoalStepSize = locateGoalStepSize>>1;
      }
      lastTurningDirection = LOCATE_GOAL_RIGHT_TURN;

```

```

        addSteeringObject(SPIN, NA, NA, -locateGoalStepSize , NA,
NO_INFO);
    }
    //if we have found the goal and are facing it, try to score
    else{
        AUTone(1000,100);
        OSWait(10);
        AUTone(2000,100);
        OSWait(10);
        AUTone(100, 100);
        LCDSetPrintf(0, 0, "DLSCST: %d: ",
targetLoc[opponentGoalIndex].count );
        LCDSetPrintf(1, 0, "scX: %d: ",
targetLoc[opponentGoalIndex].meanX );
        LCDSetPrintf(2, 0, "scY: %d: ",
targetLoc[opponentGoalIndex].meanY );
        addSteeringObject(LINE, FLOAT_TO_FIXP16(2), 0, NA, RELATIVE,
NO_INFO);
    }
    }else{
        state = SEARCH_BALL;
        scoreSecondTryFirst = 0;
        lastState = SCORE_SECOND_TRY;
    }
}
}break;

case DRIVE_TO_BALL:{
    LCDClear();
    LCDSetPrintf(0, 0, "DRIVE_TO_BALL");
    PositionType pos;
    VWGetPosition(vwhandle, &pos);
    LCDSetPrintf(2,0, "%f",pos.x);
    LCDSetPrintf(3,0, "%f",pos.y);
    //drive to the ball

    if(driveToBallFirst == 0){
        AUTone(200, 20);
        OSWait(20);
        AUTone(2000, 20);
        PositionType pos;
        VWGetPosition(vwhandle, &pos);
        addSteeringObject(SPIN, NA, NA, -
targetLoc[BALL_INDEX].alpha, NA, NO_INFO);
        addSteeringObject(LINE, targetLoc[BALL_INDEX].pos.v1,
targetLoc[BALL_INDEX].pos.v2, NA, ABSOLUTE, NO_INFO);
        driveToBallFirst = 1;
    }
    if(steeringHandler.nrOfObjects == 0){
        lastState = DRIVE_TO_BALL;
        state = SEARCH_BALL;
        driveToBallFirst = 0;
        break;
    }
}break;

```

```
case BUMP_STATE:{
    delocalized = 1;
    LCDClear();
    LCDSetPrintf(0, 0, "BUMP %d", bumpFirst);
    if(bumpFirst == 0){
        bumpStallTime = 0;
        OSGetTime(&bumpStartHr, &bumpStartMin, &bumpStartSec,
&bumpStartTicks);
        bumpStartStallTime = 0;
        //Reset all the states
        ///////////////////////////////////
        searchBallFirst = 0;
        driveToBallFirst = 0;
        scoreFirst = 0;
        scoredFirst = 0;
        scoreSecondTryFirst = 0;
        startFirst = 0;

        //SEARCH BALL
        goalCount = 0;
        turnAngle = 0;
        lastAngle = 0;
        locateGoalStepSize = QUARTERPI>>1;
        lastTurningDirection = LOCATE_GOAL_NO_TURN;

        //StallCheck
        bumpStartStallTime = 1;
        bumpStallTime = 0;
        bumpDTicks = 0;

        ///////////////////////////////////

        clear(&steeringHandler);
        OSWait(30);
        bumpFirst = 1;
        //BOTH FRONT BUMPERS
        if(resfr == 0 && resfl == 0){
            addSteeringObject(REVERSE, FLOAT_TO_FIXP16(0.3), 0, NA,
RELATIVE, NO_INFO);
        }
        //BOTH BACK BUMPERS
        else if(resbr == 0 && resbl == 0){
            addSteeringObject(LINE, FLOAT_TO_FIXP16(0.3), 0, NA,
RELATIVE, NO_INFO);
        }
        //FRONT LEFT
        else if(resfl == 0){
            //Turn LEFT and put the gear in reverse
            addSteeringObject(SPIN, NA, NA, -QUARTERPI, NA, NO_INFO);
            addSteeringObject(REVERSE, FLOAT_TO_FIXP16(0.3), 0, NA,
RELATIVE, NO_INFO);
        }
        //FRONT RIGHT
        else if(resfr == 0){
            //Turn RIGHT and put the gear in reverse
            addSteeringObject(SPIN, NA, NA, QUARTERPI, NA, NO_INFO);
        }
    }
}
```

```

        addSteeringObject(REVERSE, FLOAT_TO_FIXP16(0.3), 0, NA,
RELATIVE, NO_INFO);
    }
    //BACK LEFT
    else if(resbl == 0){
        //Turn RIGHT and drive forward
        addSteeringObject(SPIN, NA, NA, QUARTERPI, NA, NO_INFO);
        addSteeringObject(LINE, FLOAT_TO_FIXP16(0.3), 0, NA,
RELATIVE, NO_INFO);
    }
    //BACK RIGHT
    else if(resbr == 0){
        //Turn LEFT and drive forward
        addSteeringObject(SPIN, NA, NA, -QUARTERPI, NA, NO_INFO);
        addSteeringObject(LINE, FLOAT_TO_FIXP16(0.3), 0, NA,
RELATIVE, NO_INFO);
    }
}
//Are we done recovering from the collision?
else{
    OSGetTime(&bumpHr, &bumpMin, &bumpSec, &bumpTicks);
    bumpDSecs = bumpSec - bumpStartSec;
    bumpDTicks = (bumpTicks+bumpDSecs*100) - bumpStartTicks;
    LCDSetPrintf(2,0,"%d,%d",bumpDSecs ,bumpDTicks);
    if(steeringHandler.nrOfObjects == 0 || bumpDTicks > 250 ){
        //return to the SEARCH_BALL state
        bumpFirst = 0;
        state = SEARCH_BALL;
        lastState = BUMP;
    }
}
}break;
} //switch

handleSteering();
}
}

```

```

int stallCheck(){

    //if we have stalled, deal with it and break from this switch
    if(VWStalled(vwhandle) == 1){
        if(startStallTime){
            stallTime = 0;
            OSGetTime(&startHr, &startMin, &startSec, &startTicks);
            startStallTime = 0;
        }else if(dTicks > 20){ // STALLED
            AUTone(100, 20);
            OSWait(20);
            AUTone(100, 20);
            OSWait(20);
            clear(&steeringHandler);
            LCDSetPrintf(1, 0, "STALLING");
            dTicks = 0;
            delocalized = 1;
            OSWait(30);
        }
    }
}

```

```
        addSteeringObject(REVERSE, FLOAT_TO_FIXP16(0.4), 0, NA,
RELATIVE, NO_INFO);

        //Reset all the states
        //////////////////////////////////////
        searchBallFirst = 0;
        driveToBallFirst = 0;
        scoreFirst = 0;
        scoredFirst = 0;
        scoreSecondTryFirst = 0;
        startFirst = 0;

        //SEARCH BALL
        goalCount = 0;
        turnAngle = 0;
        lastAngle = 0;
        locateGoalStepSize = QUARTERPI>>1;
        lastTurningDirection = LOCATE_GOAL_NO_TURN;

        //StallCheck
        startStallTime = 1;
        stallTime = 0;
        dTicks = 0;
        //////////////////////////////////////

        state = SEARCH_BALL;
        return 1; //-- END OF STALL HANDLER //
        startStallTime = 1;
    }else{
        OSGetTime(&hr, &min, &sec, &ticks);
        dSecs = sec - startSec;
        dTicks = (ticks+dSecs*100) - startTicks;
        LCDSetPrintf(2,0,"%d,%d",dSecs ,dTicks);
    }
}
else{ // If not Stalled at all
    startStallTime = 1;
    stallTime = 0;
    dTicks = 0;
}
return 0;
}

int bumpCheck(){
    return ((resfr == 0) || (resfl == 0) || (resbr == 0) || (resbl ==
0));
}

void takePicture(){
    resetTargetLocs();

    CAMGetFrameRGB(imgRGB); // Take picture: 176*144 pixels
    locateObjects(&lcdImage);
    displayImage(0);
}
```

```
int displayImage(int key){
    static int target = 0;

    if(key == KEY1){
        AUTone(200, 50);
        ++target;
    }
    if(key == KEY2){
        --target;
    }
    if(targetLoc[BALL_INDEX].found){
        drawCross(targetLoc[BALL_INDEX].screenY,
targetLoc[BALL_INDEX].screenX);
    }
    LCDPutGraphic((image*)&lcdImage);

    target = (target % NR_OF_OBJECTS);

    return target+1;
}

void drawCross(int row, int col){
    short rs = 10;
    short cs = 14;
    int i = row -rs;
    int j = col -cs;
    for(;i< row+rs;++i){
        if(j>-1 && j<82 ){
            if( lcdImage[i][col] != 255 )
                lcdImage[i][col] = 255;
            else
                lcdImage[i][col] = 0;
        }
    }
    for(;j<col+cs;++j){
        if(i>-1 && i<62){
            if( lcdImage[row][j] != 255 )
                lcdImage[row][j] = 255;
            else
                lcdImage[row][j] = 0;
        }
    }
}

//-----//
//  INIT CAMERA  //
//-----//
void initCamera(void){
    int camera;
    camera = CAMInit(NORMAL);
    CAMSet(FPS3_75,0,0);
    if (camera == NOCAM)
        OSErrror("Camera not detected!\n Press any key to continue.\n", 4133,
0);
    else
```

```
    if (camera == INITERROR)
        OSErrror("Camera Initialisation Error!\n Press any key to
continue.\n", 4133, 0);
}

//-----//
//  RELEASE CAMERA  //
//-----//
void releaseCamera(){
    short status;
    status = CAMRelease();
    if(status != 0){
        OSErrror("An error ocured while releasing the camera", 4133,0);
    }
}

//-----//
//  PLAY SOUND  //
//-----//
void playSound(){
    AUTone(1000, 50);
    OSWait(5);
    AUTone(2000, 50);
    OSWait(5);
    AUTone(3000, 50);
    OSWait(5);
    AUTone(4000, 50);
    OSWait(5);
    AUTone(5000, 50);
    OSWait(5);
    AUTone(6000, 50);
    OSWait(5);
    AUTone(7000, 50);
    OSWait(5);
    AUTone(8000, 50);
    OSWait(5);
    AUTone(7000, 50);
    OSWait(5);
    AUTone(6000, 50);
    OSWait(5);
    AUTone(5000, 50);
    OSWait(5);
    AUTone(4000, 50);
    OSWait(5);
    AUTone(5000, 50);
    OSWait(5);
    AUTone(6000, 50);
    OSWait(5);
    AUTone(7000, 50);
    OSWait(5);
    AUTone(8000, 50);
    OSWait(5);
    AUTone(8000, 200);
    OSWait(40);
}
```

## bumpDefs.h

```
#ifndef BUMPDEFS_H
#define BUMPDEFS_H

//Bumpers used in addition to the ones defined in the eyebot library
#define BUMP_LEFT_FRONT -502
#define BUMP_RIGHT_FRONT -503

#endif
```

## Look-Up-Tables

```
//Fixed-Point Look-Up-Table for the metric distance to ball
//given the pixel coordinates for the ball on the image taken
//by the camera

int yLut[] = { 198431, 139729, 101108, 75553, 58508, 47010, 39136,
33633, 29688,
                ...
                ...
                ...
                2332, 2227, 2127 };

//Fixed-Point Look-Up-Table for atan

const int tanLut[512] = { 0, 101, 201, 302, 402, 503, 603, 704, 804,
905,
                        ...
                        ...
                        ...
                        65135, 65335 };

//Look-Up-Table for the classification of the blue goal, yellow goal
//opponent and the ball given the rgb-value of a pixel

const short objectLut[] = { 1,0,0,2,2,2,2,2,2,2,2,
                            ...
                            ...
                            ...
                            0,0,0,0,0,0,0 };

//Fixed-Point Look-Up-Table for cos

const int cosLut[512] = { 65536, 65531, 65516, 65492, 65457, 65413,
65358, 65294, 65220, 65137,
                        ...
                        ...
                        ...
                        65516, 65531 };

//Fixed-Point Look-Up-Table for atan
```

```
const int atanLut[512] = { 0, 1280, 2559, 3836, 5110, 6380, 7645, 8905,
10158, 11403,
                                ...
                                ...
                                ...
                                96386, 96399 };

//Fixed-Point Look-Up-Table for the angle between the vector with the
same
//heading as the robot and the ball, i.e. how much the robot must turn
//to be facing the ball straight on given the distance between the
robot and the
//ball along the y-axis in the robot's reference frame

int alphaLut[] = { 42277, 41320, 40364, 39407, 38450, 37493, 36536,
35579,
                                ...
                                ...
                                ...
                                -32355, -33312, -34269, -35226 };
```

## hdt.c

```
/*
| -----
|-----
|  Filename: hdttest.c
|
|  Author:      Thomas Braunl   (braunl@ee.uwa.edu.au)
|              Klaus Schmitt   (kschmitt@informatik.uni-kl.de)
|
|  Description:
|  -----
|  STANDARD HDT data for EyeBot MK4, UWA Aug. 2000
|-----
*/

#define VERSION 1.1      /* User specific version nr. for information
*/
#define NAME "Eye-MK4"  /* controller name max. 7 chars, inform. only
*/
#define ID 3           /* Unique platform id (int), just for inform.
*/

#include "hdt.h"
#include "hdt_sem.h"
#include "IRnokia.h"
#include "irtv.h"
#include "bumpdefs.h"

/* The HDT structure */
/*-----*/
int magic = 123456789;
extern HDT_entry_type HDT[];
HDT_entry_type *hdtbase = &HDT[0];
```

```
#include "types.h"
#include "const.h"
#include "rs232.h"
#include "cam.h"
#include "librobi/librobi.h"

/* infrared remote control on Servo S12 */
/* the final 1 is to state that it is used for ROBIOS */
irtv_type irtv = {1, 13, TPU_HIGH_PRIO,
                  REMOTE_ON, SPACE_CODE, 15, 0x0000, 0x03FF,
  DEFAULT_MODE,
                  1, -1, RC_RED, RC_GREEN, RC_YELLOW, RC_BLUE};

short dist0[128] = { 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
                    60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
                    65, 69, 70, 73, 75,
                    76, 77, 79, 81, 84, 85, 87, 88, 89, 90,
                    93, 95, 96, 98, 100,
                    103, 105, 107, 109, 110, 112, 114, 116, 117, 119,
                    121, 126, 127, 129, 131,
                    134, 137, 140, 144, 146, 149, 151, 156, 159, 167,
                    170, 175, 179, 183, 188,
                    195, 200, 205, 210, 220, 225, 235, 240, 250, 265,
                    275, 285, 295, 310, 320,
                    340, 350, 370, 380, 400, 410, 440, 460, 490, 530,
                    600, 610, 630, 680, 750,
                    830, 900, 999, 999, 999, 999, 999, 999, 999, 999, 999,
                    999, 999, 999, 999, 999, 999, 999, 999};

BYTE motconvL[101]= { 0, 3, 3, 3, 4, 5, 6, 7, 8,
                    9,10,11,12,13,14,15,16,17,18,19,
                    20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
                    40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,
                    60,61,62,63,64,65,66,67,68,69,69,69,69,69,69,69,69,69,69,
                    69,69,69,69,69,69,69,69,69,69,69,69,69,69,69,69,69,69,
                    69};
BYTE motconvR[101]= { 0, 1, 1, 1, 4, 5, 6, 7,
                    8,10,11,12,13,14,15,16,17,18,20,21,
                    22,24,25,27,28,29,30,31,32,33,35,36,38,40,42,46,49,51,52,54,
                    56,57,58,59,61,62,64,66,70,71,72,73,74,75,75,76,77,78,79,82,
                    85,87,89,94,95,96,97,98,99,100,100,100,100,100,100,100,100,100,100,
                    100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,
                    100,100,
                    100};

/* DC motors */
/* Motor A = left, Motor B = right */
```

```

motor_type motorA = {2, 0, TIMER1, 32000, (void*)sim_porte, 0, 1,
  (BYTE*)0};
motor_type motorB = {2, 1, TIMER1, 32000, (void*)sim_porte, 3, 2,
  (BYTE*)0};
quad_type  encoderA = {0, 2, 3, MOTOR_LEFT, 6227, 2.0}; // 6121
wheeltics per meter // 6227
quad_type  encoderB = {0, 5, 4, MOTOR_RIGHT, 6187, 2.0}; // 6121
whelltics. // 6187
vw_type drive = {0, DIFFERENTIAL_DRIVE, {QUAD_LEFT, QUAD_RIGHT,
0.163}};

/* SERVOS */
/* type, TPU-channel, Timer2, 20ms period, 0.7ms min, 1.7ms max */
/* here: servo 1 has TPU channel 2, and so on */
servo_type servo1 = {2, 2, TIMER2, 20000, 700, 1700};
servo_type servo2 = {2, 3, TIMER2, 20000, 700, 1700};
servo_type servo3 = {2, 4, TIMER2, 20000, 700, 1700};
servo_type servo4 = {2, 5, TIMER2, 20000, 700, 1700};
servo_type servo5 = {2, 6, TIMER2, 20000, 700, 1700};
//servo_type servo6 = {2, 7, TIMER2, 20000, 700, 1700};
servo_type servo7 = {2, 8, TIMER2, 20000, 700, 1700};
servo_type servo8 = {2, 9, TIMER2, 20000, 700, 1700};
//servo_type servo9 = {2, 10, TIMER2, 20000, 700, 1700};
//servo_type servo10 = {2, 11, TIMER2, 20000, 700, 1700};
//servo_type servo11 = {2, 12, TIMER2, 20000, 700, 1700};
servo_type servo12 = {2, 13, TIMER2, 20000, 700, 1700};

//bump_type bumper0 = {0, 9, TIMER2, FALLING};
bump_type bumper0 = {0, 7, TIMER2, FALLING};
bump_type bumper1 = {0, 10, TIMER2, FALLING};
bump_type bumper2 = {0, 11, TIMER2, FALLING};
bump_type bumper3 = {0, 12, TIMER2, FALLING};

/* PSD-infrared */
psd_type psd0 = {0, 14, (BYTE*)IOBase, 0, AH, (BYTE*)IOBase, 0, AH,
  (short*)&dist0};
psd_type psd1 = {0, 14, (BYTE*)IOBase, 1, AH, (BYTE*)IOBase, 0, AH,
  (short*)&dist0};
psd_type psd2 = {0, 14, (BYTE*)IOBase, 2, AH, (BYTE*)IOBase, 0, AH,
  (short*)&dist0};
psd_type psd3 = {0, 14, (BYTE*)IOBase, 3, AH, (BYTE*)IOBase, 0, AH,
  (short*)&dist0};
psd_type psd4 = {0, 14, (BYTE*)sim_porte, 5, AH, (BYTE*)IOBase, 0,
  AH, (short*)&dist0};
psd_type psd5 = {0, 14, (BYTE*)sim_porte, 4, AH, (BYTE*)IOBase, 0,
  AH, (short*)&dist0};

compass_type compass = {0,13,(void*)IOBase, 2,(void*)IOBase, 4,
  (BYTE*)IOBase, 0};

/* EyeBot SUMMARY */
info_type roboinfo =
{0,MK4,SER115200,RTSCTS,SERIAL1,0,0,AUTOBRIGHTNESS,BATTERY_ON,33,VERSIO
N,NAME,ID};

```

```

/* waitstates = 0..13, Fast Termination = 14, External = 15 */
/* Version, ROM, RAM, LCD, IO, Parallel+Serial2/3 */
/* 16MHz - 20MHz */
/* waitstate_type waitstates = {0,3,0,1,0,2}; */
/* 21MHz - 41MHz */
waitstate_type waitstates = {0,3,1,2,1,2};

/* include startup sound and image */
#include "standard.h"

/* this is a test HDT structure which contains ALL EyeBot hardware
devices */
HDT_entry_type HDT[] =
{
  {MOTOR,MOTOR_LEFT,"MotA-L", (void *)&motorA},
  {MOTOR,MOTOR_RIGHT,"MotB-R", (void *)&motorB},
  {QUAD, QUAD_LEFT,"LEFT", (void *)&encoderA},
  {QUAD, QUAD_RIGHT,"RIGHT", (void *)&encoderB},

  {VW,VW,"Drive", (void *)&drive},

  {BUMP, BUMP_LEFT,"LEFT", (void *)&bumper1},
  {BUMP, BUMP_RIGHT,"RIGHT", (void *)&bumper0},
  {BUMP, BUMP_LEFT_FRONT,"L_FRONT", (void *)&bumper2},
  {BUMP, BUMP_RIGHT_FRONT,"R_FRONT", (void *)&bumper3},
  /* {IR, IR_LF,"LF", (void *)&ir0}, */

  {SERVO,SERVO1, "SERV01", (void *)&servo1},
  {SERVO,SERVO2, "SERV02", (void *)&servo2},
  {SERVO,SERVO3, "SERV03", (void *)&servo3},
  {SERVO,SERVO4, "SERV04", (void *)&servo4},
  {SERVO,SERVO5, "SERV05", (void *)&servo5},
  //{SERVO,SERVO6, "SERV06", (void *)&servo6},
  {SERVO,SERVO7, "SERV07", (void *)&servo7},
  {SERVO,SERVO8, "SERV08", (void *)&servo8},
  //{SERVO,SERVO9, "SERV09", (void *)&servo9},
  //{SERVO,SERVO10,"SERV10", (void *)&servo10},
  //{SERVO,SERVO11,"SERV11", (void *)&servo11},
  {SERVO,SERVO12,"SERV12", (void *)&servo12},

  {PSD,PSD_FRONT, "P0-F", (void *)&psd0},
  {PSD,PSD_LEFT, "P1-L", (void *)&psd1},
  {PSD,PSD_RIGHT, "P2-R", (void *)&psd2},
  {PSD,PSD_LEFTDIAG, "P3-LD", (void *)&psd3},
  {PSD,PSD_RIGHTDIAG, "P4-RD", (void *)&psd4},
  {PSD,PSD_BACK, "P5-BK", (void *)&psd5},
  {COMPASS,COMPASS,"COMPAS", (void *)&compass},

  {WAIT,WAIT,"WAIT", (void *)&waitstates},
  {INFO,INFO,"INFO", (void *)&roboinfo},

  {IRTV,IRTV,"IRTV", (void *)&irtv},
  {STARTMELODY,STARTMELODY,"Melody", (void *)&startmelody},
  {STARTIMAGE,STARTIMAGE,"Image", (void *)&startimage},

  {END_OF_HDT,UNKNOWN_SEMANTICS,"END", (void *)0}
}

```

};

## Appendix B – Whiskers control

Input signals into the EyeBot board can be read in from a servo signal pin by using a RoBIOS system call BUMPCheck(). The idea is to have a whisker change the voltage of the input pin by short circuiting the whisker-circuit when in contact with a nearby object. At the same time we want to avoid draining the main-battery from to much power since the voltage is taken from the main board. This is done by a resistance between each whisker and its corresponding +5V power source that lower the short-circuit current. We created a circuit layout [fig 1], and soldered it onto a matrix circuit plate (having horizontal and vertical soldering tracks with holes).

