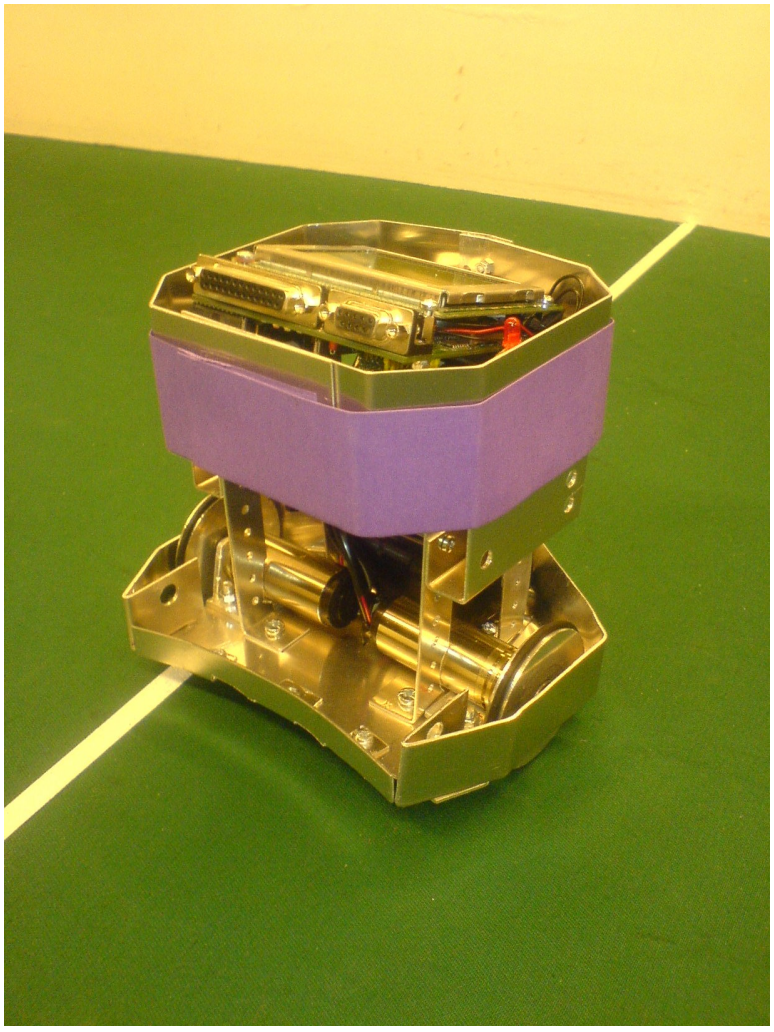
Project report
Robotics and autonomous systems (2D1426)
Grrobot



Anders Johansson
Johan Björk
Per Wennersten
Robert ter Vehn

June 14, 2007

1 Abstract

Grrobot is a robot built to be able to play soccer autonomously. It was created as a project in the course 2D1426 (Robotics and Autonomous Systems) given at the Royal Institute of Technology, spring 2007. This report will describe the technical details of the hardware and software used in grrobot. Grrobot achieved fifth place in the competition.

Contents

2 Background

This report is a rather technical description of our robot, Grrobot, that was created as part of the project in the course 2D1426 (Robotics and Autonomous Systems) given at the Royal Institute of Technology. Students who complete the course should have gained knowledge about the fundamental concepts and techniques used within the field of robotics and gained practical experience in building and developing software for an autonomous robot.

2.1 Project

The aim of the project is to build a robot that are capable of playing one-on-one soccer in the final competition of the course. We received the material for the robot almost nine weeks before the competition. We also got 24/7 access to both a building workshop and a computer lab. In the computer lab, there was a playfield setup, so we could practice with the robots.

2.2 Competition rules

A complete set of rules can be found at the course homepage [1]. Presented here are mainly the fundamental rules of construction and competition.

- The robots construction must be safe to itself, other robots, humans and may not harm the field.
- The robot must at all time fit inside a 180mm diameter vertical cylinder.
- No colors simliar to that of the ball, field or goals may be used on the robot. Each robot must have a purple marker identifying it as an opponent to any other robots on the field.
- All forms of communication with the robot is prohibited except when the robot is off the field.
- Only the provided camera is allowed on the robot.
- Beacons to help localisation is not allowed.
- In the seeding of the game the robot is alone on the field. In the competition there is at most one opponent on the field.
- The ball is an orange golf ball with a minimum diameter of 42.67mm.
- The matches will be three, four and five minutes long for group play, semi finals and final respectively.

- At the beginning of each round the robots must touch the extended line (between the corners of the field on the same side as their own goal) with some solid part of their body.
- A goal is scored when the full width of the ball has crossed the white goal line or if the robot holds the ball and enters its own goal with any part of its body (in the latter case a goal is awarded to the opponent).
- If the robot needs repair it must be taken of the field for at least 30 seconds.
- A robot holding or kicking the ball five centimeters over the field will be removed from the field for 20 seconds. Holding the ball is the same as removing a degree of freedom from the ball. A robot may not cover the ball so that less than 75 percent of the ball is outside the convex hull of the robot.

2.3 Competition procedure

The competition takes place on the game field shown in (1). and consists of

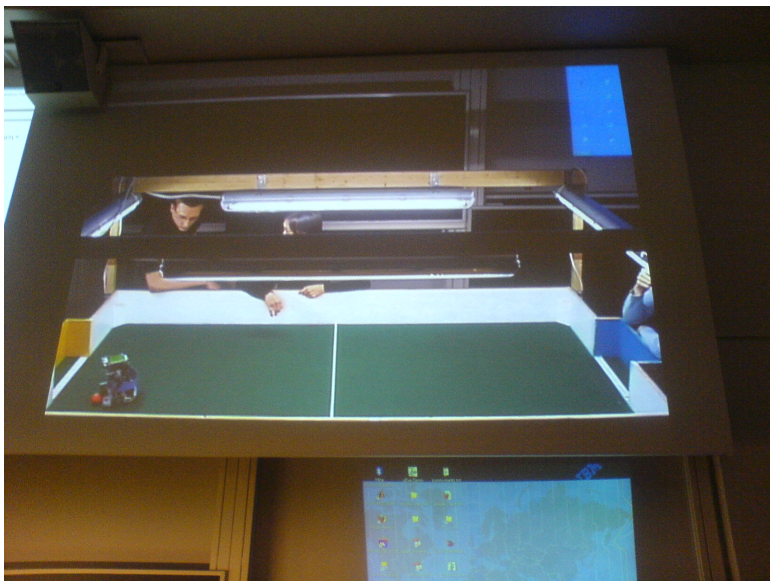


Figure 1: The game field

two parts. A qualification round and a group play. During the qualification round the robot is alone on the field and is supposed to make as many goals as possible. The qualification round is two minutes long. One additional minute is granted if the robot was unable to make any goals during the initial two minutes. When a goal is scored the referee takes the ball and puts it somewhere on the center line of the field. If the robot is successful

in making at least one goal, it has qualified for the group play. The rounds in the initial group play are three minutes long. During this time two robot will compete in making the most goals. A winning robot is awarded two point and each robot receive one point if its a tie. The robot with the least score will not make it further in the group play. Following is the semifinals and the final which is four and five minutes long respectively.

2.4 Material

We where provided with plenty of material to be able build a basic robot.

- An EyeBot microcontroller board with a camera
- 8x1.5V batteries with holders and a batterycharger
- Two motors with builtin encoders and gearbox
- RS-232 serial cable
- A servo engine
- Aluminium plates

In addition to these items, we had two motors that we took from an old CD-RW and an old Sony Walkman music player. Our plan was to use one of these motors to drive our roller, however, we never found any material good enough to use as a roller,thus the motors where never used in the final robot.

3 Technical description

3.1 Hardware

3.1.1 Design

We decided early on in the project that we wanted our robot to look a bit special. To achieve an aggressive look, since there were a competition to come, we wanted the robot to look a bit "sharp" and edgy. Furthermore we wanted to be able to us the ability to travel backwards since many of the other groups were probably not going to use that ability. By doing this it would make our robot look a bit more unique in its way of travel and - most of all - more fun to look at. As Electrolux learnt while developing the "Trilobite" (a robot vacuum cleaner): unpredictable is a lot more fun to look at then predictable movement patterns.

The ability to turn on the spot makes it a lot easier to get out of corners and other tight areas. To make this possible we placed the wheels on the center line of the circular shape we were allowed to build our robot within

(as defined by the 18 centimeter diameter competition rule constraining the outer chassi size).

To minimize the risk of backing in to the ball and scoring in our own goal we strived to create a sharp plow shape at the back of the robot. Our aim was to make the ball bounce of to the side rather than bouncing backwards if the robot backed into it. At the same time we wanted to keep the front of the robot as wide as possible to have a larger area of contact with the ball when driving forwards. We came up with a fearsome looking stingray shape, unfortunatly we had to cut the stinger off to stay within the rule constraining the outer chassi size.

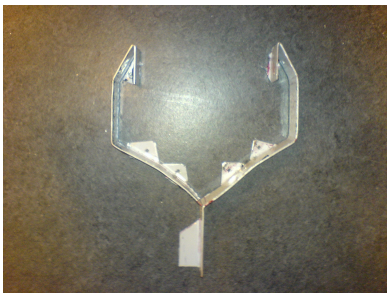


Figure 2: Our chassi

To protect the EyeBot microcontroller board we added an aluminium cover around it. This had the plus of at the same time making the robot look a bit more like a tank (even more intimidating to the other robots).

Figure 3: Final design of the robot

3.1.2 Roller

Like mostly all of the other groups we were quite keen on having a roller, creating the ultimate roller seemed like a goal every group was trying to achieve. As time passed we couldn't really find any appropriate material for a roller, we also didn't want to use something an other group had already used since that could have been regarded as idea stealing. Eventually we scrapped the idea since our locomotion algorithms didn't require the robot to turn on the spot while having the ball, thus making the roller quite unnecessary.

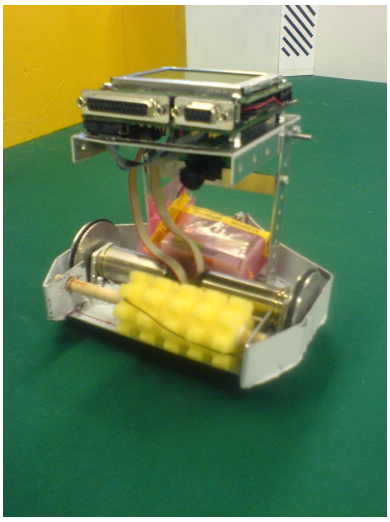


Figure 4: Our robot with one of the rollers we tried

3.1.3 Servo engine

Since each group was given a servo engine we felt a bit urged to use it, however we came to the conclusion that adding the servo to our design would make it more complex - and not worth the effort. Since our robot could turn on the spot we felt no need to have the servo engine involved in handling the ball. The other idea we had was to use the servo to control the camera, but we decided to skip this aswell because of the complexity in programming it would require.

3.1.4 LED

A robot must look nice and interesting and of course have some kind of blinking light, where's the fun otherwise? To make the robot look smarter and initially to help us know what the robot was doing during the development phase we added a red LED mounted beside the EyeBot. The place was choosen since the LED would only be visible when looking down at the

robot, and not from the direct side of it. This was done in compliance to the "no bright colors which can destract other robots"-rule. We decided to use the LED to indicate when the robot was processing a picture by switching it on when the picture was taken and of when the processing had finished. This also gave us a visual sense of how much time took processing images. In practice this showed us that processing an image seldom took longer than a second.

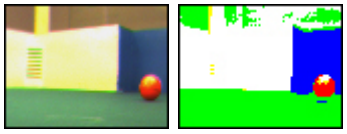
3.2 Software

3.2.1 Overview

3.2.2 Image processing

Due to the very limited processing power of the eyebot controller (33 MHz), there is room for very little processing per pixel. We decided to implement a simple nearest-neighbour classifier and convert it to a look-up table for fast access on the controller.

In order to do this, sample images were taken and typical points selected to represent each goal, the green of the field, the white of the walls and the orange colour of the ball. In order to stabilize the results in varying lighting conditions, these values were somewhat altered to make them all approximately the same brightness. Next, a 32x32x32 matrix was constructed to hold the nearest-neighbour classifications for different R, G and B values of a colour. The result can be seen in (5).



(a) Original image

(b) After nearest neighbour classifier detection

Figure 5: Image processing

3.2.3 Localization

From the start, we decided for a fairly ambitious robot control strategy, where each image taken by the robot could be used to determine its position on the field.

One idea was to attempt to identify the corners of the field, but we decided instead to rely on the goals. The idea is to find the two lower corners of either goal in the image, use that information to approximate

where they are positioned relative the robot in the real world, and finally use that information to triangulate the robot position.

Initial results were quite positive, with the robot usually able to roughly estimate its position on the pitch. Most images had the error within 10 or 20 centimetres or so of the actual robot position. An output of the robots estimated position using the input in (5) can be seen in (6).

Later on, two major problems were discovered with the localization. Firstly, in some images where the pixel classification failed or a corner of the goal was being blocked by another robot or the ball, the localization results were quite random. It proved difficult to filter out these erroneous values, making the localization much less reliable. Secondly, we were unable to take pictures while the robot was moving due to the nature of the VW-drive. This meant that we could not take as many images as we would have liked, further reducing the reliability of our localization estimates.

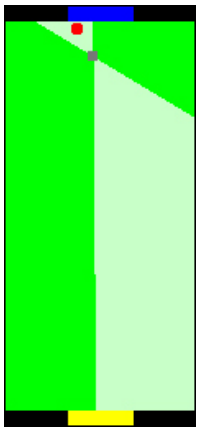


Figure 6: The resluting estimated localization from the input in fig (5)

3.2.4 State machine

Our first attempt at a state machine was quite simple, we relied completely on the localization system described in the previous section. First, the robot would drive forward look for the ball, turning around until it was found. Then, the robot would attempt to drive to a position 20 centimetres behind the ball on a line drawn from the target goal and through the ball. Finally, the robot would drive to the coordinates for the opposing goal. Because we could not take any pictures while driving, the robot would stop every 20 centimetres to take a picture. This early attempt worked some of the time, but was unfortunately found to be too unreliable. Around 10 images were taken in the time it took to reach the other end of the field, and if one of the images provided an incorrect localization the robot would start driving in a fairly random direction.

We considered employing Kalman filtering[2] to alleviate this problem, but we found that the uncertainty would be too large, requiring the robot to stop and take far too many images to get a somewhat reliable position estimate.

In our second and final state machine (7), a much simpler approach was used. The robot first assumes its position to be in the centre of its own goal-line and finds the ball in the same manner as our first attempt, but this time positioning itself 40 centimetres behind it. Next, the robot finds the ball again, turns until the ball is approximately in the center of the images, then drives 20 centimetres towards it. The robot then continues turning and driving towards the goal in 10-centimetre increments until the ball is no longer visible, at which point the robot assumes it is in posession of the ball. The next step is to use a similar strategy to drive towards the goal. The robot drives into the opposing goal, hopefully driving against and aligning itself towards the wall, then backs out 2 metres and restarts the state machine.

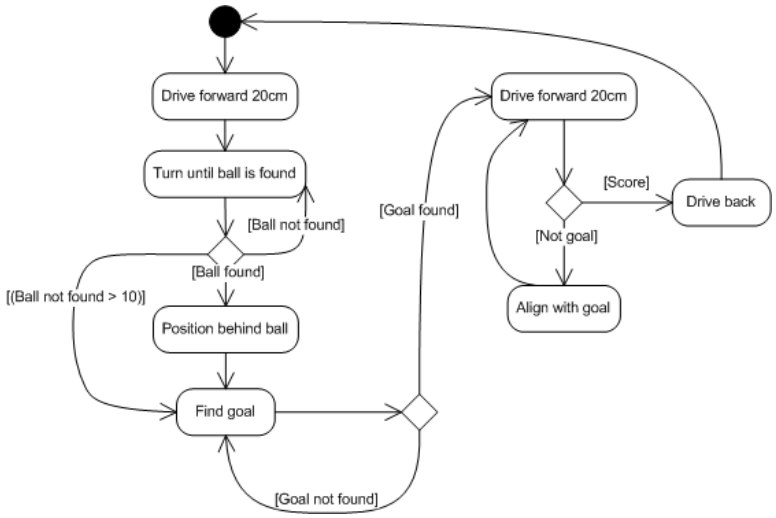


Figure 7: The state chart of the Grrobot

The design is made so that if any troubles are encountered, such as the ball not being found or the robot being lost, it drives into the opposing goal and then backs out in order to get to a known reference position.

This second state machine proved to be quite robust. There was, however, a few major problems. We were still forced to stop to take pictures due to the VW-drive, and because we had no roller, we had to drive slowly before stopping or risk losing the ball. The end result is a reliable but very slow robot.

Another issue is that due to the state machine design, the robot is simply unable to score for certain initial ball positions. The robot will not look

behind its initial position for the ball, exluding any position too close to its own goal, and it also cannot get the ball if the position 40 centimetres behind it is inside a wall, excluding any position close to a wall.

Figure 8: A sample picture taken while the vw was running

3.2.5 vw-drive

We used the V-Omega drive, part of the standard eyebot library [3], to control the movement of grrobot. The choice to use vw-drive is probably a good choice to get things up and running quick, but due to some limitations in the vw-drive (The inability to take pictures when in motion), it's probably not suited for a robot that is to score high in the competition. A sample picture with the vw on is shown in (8)

3.2.6 Debug code

We figured rather early that we wanted a system to be able to send debugdata over the serialport directly to our computer for use in debugging. We wrote a system that was able to send both binary (images) and ASCII data over the serialport. We also wrote a small application on the host that took coordinates and used the same algorithm as grrobot to build a map of what the robot's view of the world was.

This proved to work pretty bad when doing the actual testing, the robot was way to light, so the serialcable slightly adjusted the robots position, so it's localization got all confused. As we had to basically disable movement to get good information from this, it was rather useless for real debugging. There is a wireless module for eyebot [4], which you can setup to communicate directly to your host. The transfer rate of EyeNet is only 9600 baud, so our feature to send real RGB-data is not really an option. Eyenet would however have proved very useful for our debugmap feature, as we generate the image on the host, the data needed from eyebot is minimal.

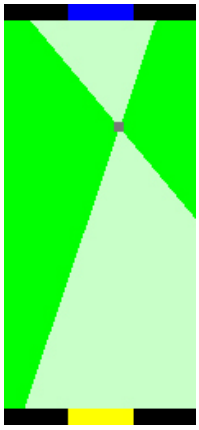


Figure 9: The output on the debughost

4 Results

4.1 Qualification

We managed to score one goal in the qualification in the two initial minutes. We where unable to score an additional goal because the slow movement of the robot.

4.2 Competition

We didn't score very well in the competition, we lost all our matches except one draw against Randalf Purple. We didn't expect to score very high against other robots using a more "insect like" behavior, as our robot instead tried to rely on localization, and therefor, if it gets lost, it needs to go back to a known state (7). We where really close to scoring in one of the matches, however, again because our robot is so slow, we got intercepted maybe 10cm from the goal.

5 Conclusions

To build an autonomus fotballplaying robot showed to be both challenging and educational. It was hard to implement well working localization and positioning algorithms with the sensors allowed. Simpler robust algorithms like: Find the ball; Go to the ball; Find the goal; Go to goal with ball, seems to be most efficient in this context. Grrobot became a crossing between a smart robot using localization and a stupid robot aming for the ball and the goal. With this approach we where able to test both techniques and at the same time, reach the goal of the cource.

A Bibliography

References

[1] Anders Dovervik Patric Jensfelt. 2d1426 competition rules. `http://www.csc.kth.se/utbildning/kth/kurser/2D1426/robot07/`.

[2] Illah R. Nourbakhsh Roland Siegwart. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[3] Klaus Schmi Thomas Bräunl. Eyebot library reference. Reference manual for eyebot controller, 2006.

[4] Klaus Schmi Thomas Bräunl. Eyenet - eyebot wireless communication network. Information about the eyebot wireless module, 2006.

B Source code

B.1 Robot code

B.1.1 Main statemachine

main.c

```
/*
  main.c
  Main controller workloop.
  Copyright ras9
*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#include "eyebot.h"
#include "main.h"
#include "debug.h"
#include "motor.h"
#include "helpers.h"
#include "image.h"
#include "camera.h"
#include "digitalout.h"

int             BLUEGOAL = 0;
int             YELLOWGOAL = 1;

int             eRobX = -1, eRobY = -1, eBallX = -1, eBallY = -1;
float           eRobAng = 0, eBallAng = 0;

int
sanityCheck(int x, int y)
{
  if (x < 0 || x > FIELDWIDTH)
    return 0;
  if (y < 0 || y > FIELDLEN)
    return 0;
  return 1;
}

void
shiftAng(float shift)
{
  eRobAng += shift;
  if (eRobAng < M_PI)
    eRobAng -= 2 * M_PI;
}

void
updateRobotPos(int x, int y, float ang)
{
  float           proc = TRUSTROBOTPOS;
  /* how much do we trust new camera estimates?  */
  printf("Est. pos.: %d,%d\n", x, y);
  if (!sanityCheck(x, y))
    return;

  if (eRobX < 0) {
    eRobX = x;
    eRobY = y;
    return;
  }
  eRobX = (int) ((proc * x) + (eRobX * (1 - proc)));
  eRobY = (int) ((proc * y) + (eRobY * (1 - proc)));
  eRobAng = ((proc * ang) + (eRobAng * (1 - proc)));
  dbgPrintf("updateRobotPos: %d,%d,%f\n", eRobX, eRobY, eRobAng);
  dbgCoord(eBallX, eBallY, eRobX, eRobY, eRobAng);
}

void
updateBallPos(int x, int y)
{
  float           proc = TRUSTBALLPOS;
  /* how much do we trust new camera estimates? */
  if (!sanityCheck(x, y))
    return;

  if (eBallX < 0) {
    eBallX = x;
```

```
    eBallY = y;
    return;
  }
  eBallX = (int) ((proc * x) + (eBallX * (1 - proc)));
  eBallY = (int) ((proc * y) + (eBallY * (1 - proc)));
  dbgPrintf("updateBallPos: %d,%d\n", eBallX, eBallY);
  dbgCoord(eBallX, eBallY, eRobX, eRobY, eRobAng);
}


int
getY(int x, int y)
{
  return 10 + (int) (DISTFACTOR * (float) (62 - y) / (float) (y + 5));
}

int
getX(int x, int y)
{
  int              dist = getY(x, y);
  return ((x - 42) * (dist / 3)) / XFACTOR;
}



void
faceGoal()
{
  double           ang = processImage();
  while (ang < -0.02 || ang > 0.02) {
    while (ang < -1000) {
      motorDriveTurn(0, 0.7, 0.5);
      shiftAng(0.7);
      ang = processImage();
      printf("No goal in sight!\n");
    }
    motorDriveTurn(0, ang, 0.2);
    shiftAng(ang);
    ang = processImage();
  }
}

int
goTo(int goalx, int goaly, float goalang)
{
  float            ang1;
  float            dist;
  float            ang2;
  float            goAng;
  int              dy = goaly - eRobY;
  int              dx = eRobX - goalx;
  float            distance;

  printf("goTo: from (%d,%d) to (%d,%d)\n", eRobX, eRobY, goalx, goaly);

  goAng = (ratan((float) dx / (float) dy));
  ang1 = goAng - eRobAng;
  dist = -dy / rsin(M_PI_2 - goAng);
  ang2 = goalang - goAng;

  dbgPrintf("turning %f, %f degrees\n", ang1, ang2);

  motorDriveTurn(0, ang1, 0.5);
  distance = motorDriveStraight(dist / 100, 0.5);
  if (distance > ABS((dist / 100)) - 0.05) {
    printf("We crashed into a wall!, %.4f\n", distance);
    OSWait(100);
    return -1;
  }
  printf("Distance: %.4f\n", distance);
  motorDriveTurn(0, ang2, 0.5);
  motorStop();
  eRobX = goalx;
  eRobY = goaly;
  eRobAng = goalang;
  dbgPrintf("GoTo: robx: %d, roby: %d, robang: %f\n", eRobX, eRobY, eRobAng);
  dbgCoord(eBallX, eBallY, eRobX, eRobY, eRobAng);
  return 0;
}

void
goToSlowNoTurn(int goalx, int goaly)
{
  float            ang1;
```

```
  float            dist;
  float            goAng;
  int              dy = goaly - eRobY;
  int              dx = eRobX - goalx;

  dbgPrintf("goToSlowNoTurn: from (%d,%d) to (%d,%d)\n", eRobX, eRobY, goalx, goaly);

  goAng = (ratan((float) dx / (float) dy));
  ang1 = goAng - eRobAng;
  dist = -dy / rsin(M_PI_2 - goAng);

  dbgPrintf("turning %f degrees\n", ang1);

  motorDriveTurn(0, ang1, 0.3);
  motorDriveStraight(dist / 100, 0.1);
  motorStop();
  eRobX = goalx;
  eRobY = goaly;
  eRobAng = goAng;
  dbgCoord(eBallX, eBallY, eRobX, eRobY, eRobAng);
}

void
findBall()
{
  int              time = 0;
  processImage();
  while (eBallX < 0) {
    if (time > 11)
      return;
    if (time % 3 == 1) {
      motorDriveTurn(0, -M_PI / 4.0 f, 0.5);
      shiftAng(-M_PI / 4.0 f);
    } else {
      motorDriveTurn(0, M_PI / 8.0 f, 0.5);
      shiftAng(M_PI / 8.0 f);
    }
    if (time % 3 == 2) {
      motorDriveStraight(0.2, 0.5);
      eRobY -= 20;
    }
    dbgCoord(eBallX, eBallY, eRobX, eRobY, eRobAng);
    processImage();
    time++;
  }
}

void
faceBall()
{
  processImage();
  while (eBallAng < -0.02 || eBallAng > 0.02) {
    if (eBallAng < -1000) {
      findBall();
    }
    if (eBallAng > -1000) {
      motorDriveTurn(0, eBallAng, 0.2);
      shiftAng(eBallAng);
      processImage();
    } else
      break;
  }
}

int
goBehindBall(int distance)
{
  float            targetang;
  int              targetx, targety;
  targetang = ratan((eBallX - FIELDWIDTH / 2.0 f) / (eBallY + 10));
  targetx = (int) (distance * rsin(targetang));
  targety = (int) (distance * rcos(targetang));
  targetx += eBallX;
  targety += eBallY + 10;

  if (goTo(targetx, targety, targetang) < 0)
    return -1;
  return 0;
}

void
slowTowardsGoal2(float spd)
{
  float            driven;
```

```
  while (1) {
    faceGoal();
    eRobY -= 28;
    driven = motorDriveStraight(0.3, spd);
    if (driven > 0.25) {
      eRobX = FIELDWIDTH / 2;
      eRobY = -5;
      eRobAng = 0;
      break;
    }
  }
}


void
controller()
{
  float            spd = 0.2;
  motorInitialize(5, 0.3, 2.0, 0.1);
  LCDMenu("   ", "   ", "   ", "   ");

  eRobX = FIELDWIDTH / 2;
  eRobY = FIELDLEN;
  eRobAng = 0;

  processImage();
  while (1) {
      if (goTo(FIELDWIDTH / 2, FIELDLEN - 30, 0) < 0) {
        goto robot_is_lost;
      }
      findBall();
    }
    if (eBallX > 0) {
      if (goBehindBall(45) < 0) {
        goto robot_is_lost;
      }
      processImage();
      faceBall();
      motorDriveStraight(0.3, 0.3);
      processImage();
      while (eBallAng > -1000) {
        motorDriveTurn(0, eBallAng, 0.2);
        shiftAng(eBallAng);
        motorDriveStraight(0.1, 0.1);
        processImage();
      }
      eRobY -= 35;
    }
    spd = 0.13;
robot_is_lost:
    blinkLED();
    slowTowardsGoal2(spd);
    printf("I think I scored!\n");

    //motor off !
    eBallX = -1;
    eBallY = -1;
    spd = 0.2;
  }
}

int
main()
{
  int              choice = 0;
  eRobX = -1;
  eRobY = -1;
  eBallX = -1;
  eBallY = -1;
  eRobAng = -1;
  printf("Grrbot at your service!     Please choose\ntarget goal.\n");
  LCDMenu("BLU", "YEL", "", "");
  choice = KEYGet();
  if (choice == KEY2) {
    BLUEGOAL = 1;
    YELLOWGOAL = 0;
  } else if (choice == KEY1) {
    BLUEGOAL = 0;
    YELLOWGOAL = 1;
  } else {
    return 0;
  }
```

```
  dbgInitialize();
  InitCam();
  controller();
  return 0;
}
```

main.h

```
/*
  main.h
  Copyright ras9
*/

#ifndef __MAIN_H__
#define __MAIN_H__

#define FIELDLEN 237
#define FIELDWIDTH 117
#define GOALDIST 38
#define GOALWIDTH 40
#define DISTFACTOR 40
#define XFACTOR 39
#define TRUSTROBOTPOS 0.1
#define TRUSTBALLPOS 1

#define BALL 2
#define FIELD 3
#define WALL 4
#define OPPONENT 5

extern int BLUEGOAL;
extern int YELLOWGOAL;

int getX(int x, int y);
int getY(int x, int y);
void updateRobotPos(int x, int y, float ang);
void updateBallPos(int x, int y);

extern int eRobX,eRobY;
extern int eBallX,eBallY;
extern float eRobAng,eBallAng;
#endif
```

B.1.2 Imageprocessing

image.c

```
/*
  image.c
  Routines for image processing.
  Copyright ras9
*/
#include "eyebot.h"
#include "helpers.h"
#include "debug.h"
#include "main.h"
#include "perkamera/lut.h"
#include <math.h>
#include "digitalout.h"
#include <stdio.h>

static char
rgbclassifyLUT(int r, int g, int b)
{
  return classifier[r / 8][g / 8][b / 8];
}

//Returns angle to goal located..
double
processImage(void)
{
  double          result = -1001;
  char            classed[82][62];
  int             x, y, ballpx, ballpy, ballpixels;
  int             bgrx, bgry, bglx, bgly, bvis;
  int             ygrx, ygry, yglx, ygly, yvis;
  int             robx = 0, roby = 0;
  int             ballx = -1, bally = -1;
  float           ang = 0;
  int             goaledge = 0;
```

```
int              centerx, centery;

turnOnLED();
eBallAng = -1001;
colimage         pic;

bvis = 0;
yvis = 0;
ballpixels = 0;
ballpx = -1;
ballpy = -1;
bgrx = -1;
bgry = -1;
bglx = -1;
bgly = -1;
ygrx = -1;
ygry = -1;
ygly = -1;
yglx = -1;

CAMGetColFrame(&pic, 0);

for (x = 1; x < 81; x++) {
  for (y = 1; y < 61; y++) {
    classed[x][y] = rgbclassifyLUT(pic[y][x][0], pic[y][x][1], pic[y][x][2]);
  }
}

//SendPic(&pic);
for (x = 1; x < 81; x++) {
  for (y = 2; y < 61; y++) {
    if (classed[x][y] == BALL) {
      ballpixels++;
      if (y > ballpy &&
          classed[x][y - 1] == BALL &&
          classed[x][y - 2] == BALL) {
        ballpx = x;
        ballpy = y;
      }
    } else if (classed[x][y] == YELLOWGOAL) {
      yvis++;

      if (x + y > ygrx + ygry &&
          classed[x - 1][y] == YELLOWGOAL &&
          classed[x][y - 1] == YELLOWGOAL &&
          classed[x][y - 2] == YELLOWGOAL) {
        ygrx = x;
        ygry = y;
      }
      if ((y - x > ygly - yglx || yglx < 0) &&
          classed[x + 1][y] == YELLOWGOAL &&
          classed[x][y - 1] == YELLOWGOAL &&
          classed[x][y - 2] == YELLOWGOAL) {
        yglx = x;
        ygly = y;
      }
    } else if (classed[x][y] == BLUEGOAL) {
      bvis++;

      if (x + y > bgrx + bgry &&
          classed[x - 1][y] == BLUEGOAL &&
          classed[x][y - 1] == BLUEGOAL &&
          classed[x][y - 2] == BLUEGOAL) {
        bgrx = x;
        bgry = y;
      }
      if ((y - x > bgly - bglx || bglx < 0) &&
          classed[x + 1][y] == BLUEGOAL &&
          classed[x][y - 1] == BLUEGOAL &&
          classed[x][y - 2] == BLUEGOAL) {
        bglx = x;
        bgly = y;
      }
    }
  }
}

if (bvis > 30 && bgrx > 0 && bglx > 0) {
  //blue goal visible..
    centerx = (bgrx + bglx) / 2;
  centery = (bgry + bgly) / 2;
  result = ratan((float) getX(centerx, centery) / ((float) getY(centerx, centery)));
  if (!goaledge && getY(bgrx, bgry) < 120) {
    ang = -ratan((float) (getY(bgrx, bgry) - getY(bglx, bgly)) / (float) (getX(bgrx, bgry) - getX(bglx, bgly)));
```

```
      robx = (int) (getX(bglx, bgly) * rcos(ang) - getY(bglx, bgly) * rsin(ang));
      roby = (int) (getX(bglx, bgly) * rsin(ang) + getY(bglx, bgly) * rcos(ang));
      robx = -robx;
      robx += GOALDIST;
      //roby = (int) (roby * 1.3 f);
      dbgPrintf("New pos!\n");
      updateRobotPos(robx, roby, ang);
    }
  } else if (0 && yvis > 10 && ygrx > 0 && yglx > 0 && !goaledge) {
    //yellow goal visible..
    ang = M_PI - ratan((float) (getY(ygrx, ygry) - getY(yglx, ygly)) / (float) (getX(ygrx, ygry) - getX(yglx, ygly)));
    robx = (int) (getX(yglx, ygly) * rcos(ang) - getY(yglx, ygly) * rsin(ang));
    roby = (int) (getX(yglx, ygly) * rsin(ang) + getY(yglx, ygly) * rcos(ang));
    robx = GOALDIST - robx;
    //roby = (int) (roby * 1.3 f);
    roby = FIELDLEN + roby;
    dbgPrintf("New pos!\n");
    updateRobotPos(robx, roby, ang);
  }
  if (ballpixels > 3 && ballpx > 0) {
    ballx = eRobX + (int) (getX(ballpx, ballpy) * rcos(eRobAng) - getY(ballpx, ballpy) * rsin(eRobAng));
    bally = eRobY - (int) (getX(ballpx, ballpy) * rsin(eRobAng) + getY(ballpx, ballpy) * rcos(eRobAng));
    printf("Ball found at abs %d,%d\n", ballx - eRobX, bally - eRobY);
    updateBallPos(ballx, bally);
    eBallAng = ratan(((float) getX(ballpx, ballpy)) / ((float) getY(ballpx, ballpy)));

  }
  printf("eballang: %lf\n", eBallAng);
  turnOffLED();
  return result;
}
```

image.h

```
/*
   image.h
   Copyright ras9.
*/
#ifndef __IMAGE_H__
#define __IMAGE_H__

double processImage (void);

#endif
```

B.1.3 Serial debug

debug.c

```
/*
  debug.c
  Routines to debug over the serialline.
  Copyright ras9
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>

#include "lcd.h"
#include "debug.h"
#include "eyebot.h"


#define BLOCKING_DEBUG 1
#define MAGIC_NUMBER 0xCAFEBABE

#define DEBUGUID 3

static struct sem debugSem;
static struct sem debugMutex;
static struct sem debugPrintfMutex;

static dbgEntry_t *debugCurrentEntry;
static dbgEntry_t *debugLastEntry;

static void SendIntRS232(int data,int interface)
{
  int i;
  for(i = 0; i < 4;i++) {
```

```
    OSSendCharRS232((data >> (i*8)) & 0xff,interface);
  }

}

//Send debug data. (blocking wait on a semafore if BLOCKING_DEBUG is zero.
static void processDbgEntry(void)
{
  int i;
  dbgEntry_t *current;

#if BLOCKING_DEBUG == 0
  OSSemP(&debugSem);
  //Okay, we have data in our debug linked list, let's spew it over the serial port
  OSSemP(&debugMutex);
#endif
  current = debugCurrentEntry;
  if(current == debugLastEntry) {
    debugLastEntry = 0;
  }
  debugCurrentEntry = current->next;
#if BLOCKING_DEBUG == 0
  OSSemV(&debugMutex);
#endif
  //Send it over serialport...
//  printf("Curr: %c\ntime:%d\nlen:%d,next: 0x%p\n",current->type,current->timestamp,current->len,debugCurrentEntry);
  SendIntRS232(MAGIC_NUMBER,SERIAL1);
  OSSendCharRS232(current->type,SERIAL1);
  SendIntRS232(current->timestamp,SERIAL1);
  SendIntRS232(current->len,SERIAL1);
  for(i = 0;i < current->len;i++) {
    if(i % 1000 == 0) {
      LCD_DrawBusy();
    }
    OSSendCharRS232(current->data[i],SERIAL1);
  }
  free(current);
}

//This is not used currently, as we decided not to use a multithreaded core for grrrobot.
static void debugThread(void)
{
  while(1) {
    processDbgEntry();
  }
}

//Initialize the debugcore.
void dbgInitialize(void)
{
#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  struct tcb *debugtcb;

  //Initialize debug structures and start the thread
  OSSemInit(&debugSem,0);
  OSSemInit(&debugMutex,1);
  OSSemInit(&debugPrintfMutex,1);
#endif

  //Init serialport
  OSInitRS232(SER115200,NONE,SERIAL1);

  debugCurrentEntry = 0;
  debugLastEntry = 0;
#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  debugtcb = OSSpawn("debugThread",debugThread,DEFAULT_STACKSIZE,MIN_PRI,DEBUGUID);
  OSReady(debugtcb);
#endif

}

#define DBGADD_COPY (1<<0)
int _dbgAdd(int type,int len,char *data,int flags) {
  dbgEntry_t *entry;

  if(flags & DBGADD_COPY) {
    entry = malloc(sizeof(dbgEntry_t)+len);
  } else {
    entry = malloc(sizeof(dbgEntry_t));
  }
  if(!entry) {
    OSPanic("Out␣of␣memory!\n");
  }
```

```
  entry->type = type;
  entry->len = len;
  entry->timestamp = OSGetCount();
  entry->next = 0;
  if(flags & DBGADD_COPY) {
    entry->data = (char*) (entry+1);
    memcpy(entry+1,data,len);
  } else {
    entry->data = data;
  }

#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  OSSemP(&debugMutex);
#endif

  if(debugLastEntry != 0) {
    debugLastEntry->next = entry;
  }

  debugLastEntry = entry;
  if(debugCurrentEntry == 0) {
    debugCurrentEntry = debugLastEntry;
  }
#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  OSSemV(&debugMutex);
  OSSemV(&debugSem);
#endif

  return 0;
}

int _dbgAddBlocking(int type,int len,char *data) {
  int err;

  err = _dbgAdd(type,len,data,0);
  if(err == 0) {
    processDbgEntry();
  }
  return err;
}

int dbgAdd(int type,int len,char *data) {
#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  return _dbgAddBlocking(type,len,data);
#else
  return _dbgAdd(type,len,data,DBGADD_COPY);
#endif
}

int dbgPrintf(const char * fmt, ...)
{
  int retval;
  va_list ap;
  static char tempbuf[1024];


#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  OSSemP(&debugPrintfMutex);
#endif
  va_start(ap,fmt);
  vsnprintf(tempbuf,sizeof(tempbuf),fmt,ap);

  retval = dbgAdd(DEBUG_TEXT,strlen(tempbuf)+1,tempbuf);

  va_end(ap);
#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  OSSemV(&debugPrintfMutex);
#endif
  return retval;
}

int _dbgPrintf(int type,const char * fmt, ...)
{
  int retval;
  va_list ap;
  static char tempbuf[1024];

#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  OSSemP(&debugPrintfMutex);
#endif
  va_start(ap,fmt);
  vsnprintf(tempbuf,sizeof(tempbuf),fmt,ap);

  retval = dbgAdd(type,strlen(tempbuf)+1,tempbuf);
```

```
  va_end(ap);
#if defined(BLOCKING_DEBUG) && BLOCKING_DEBUG == 0
  OSSemV(&debugPrintfMutex);
#endif
  return retval;
}

//This is used to send coordinates to the debughost.
//We are using this to draw a map of what the robot thinks the current state is.
void dbgCoord(int ballx,int bally,int robx,int roby,float robang)
{
  _dbgPrintf(DEBUG_COORDS,"%d,%d,%d,%d,%f\n",ballx,bally,robx,roby,robang);
}
```

debug.h

```
#ifndef __DEBUG_H__
#define __DEBUG_H__

/*
Protocol over the serial line
BYTE1: TYPE
BYTE2-5: TIMESTAMP
BYTE6-9: LEN
BYTE10-10+LEN: DATA
*/

#define DEBUG_IMAGE 'I'
#define DEBUG_TEXT 'T'
#define DEBUG_COORDS 'C'

typedef struct dbgentry {
  int len;
  int timestamp;
  char type;
  struct dbgentry * next;

  char *data;
} dbgEntry_t;


void dbgInitialize(void);

int dbgAdd(int type,int len,char *data);
int dbgPrintf(const char * restrict, ...);
void dbgCoord(int ballx,int bally,int robx,int roby,float robang);

#endif
```

B.1.4 Camera

camera.c

```
/*
  camera.c
  Routines to initialize and handle the camera.
  Copyright ras9
*/

#include <stdlib.h>

#include "eyebot.h"
#include "debug.h"

#include "camera.h"

void InitCam()
{
  int camera;
  int bright, hue, sat;

  camera = CAMInit(WIDE);
  OSWait(10);
  camera = CAMInit(WIDE);

  //Set values for brightness, hue and saturation. Values for brigh and hue are choosen by trial&error.
  CAMGet(&bright, &hue, &sat);
  CAMMode(NOAUTOBRIGHTNESS);
  CAMSet(160, 110, sat);
```

```
}
```

camera.h

```
/*
  camera.h
  Routines to initialize and handle the camera.
  Copyright ras9
*/
#ifndef __CAMERA_H__
#define __CAMERA_H__

void InitCam(void);

#endif
```

B.1.5 Motor

motor.c

```
/*
  motor.c
  Routines to control the motor.
  Copyright ras9
*/
#include <stdio.h>
#include <math.h>
#include "eyebot.h"
#include "motor.h"


static VWHandle vw;
static SpeedType s;
static PositionType start;

/*
* Initialize the motors for VW-driving with the specified constants
* This is just for testing
*/
void motorInitialize(float Vv, float Tv, float Vw, float Tw)
{
  vw=VWInit(VW_DRIVE,1);
  VWSetPosition(vw,0,0,0);
  VWGetPosition(vw,&start);
  VWStartControl(vw,Vv,Tv,Vw,Tw);
}

/*
* Avoid setting speed below 0 and avoid values that are really high
*/
void changeSpeed(float dSpeed)
{
  s.v += dSpeed;
  if(s.v>1){
    s.v = 0;
  }
}

/*
* Avoid setting rotational speed below 0 and avoid values that are really high
*/
void changeRotation(float dPhi)
{
  s.w += dPhi;
  if(s.w>1){
    s.w = 0;
  }
}

/*
* Avoid setting speed below 0 and avoid values that are really high
*/
void setSpeed(float dSpeed)
{
  s.v = dSpeed;
}

/*
* Avoid setting rotational speed below 0 and avoid values that are really high
*/
```

```
void setRotation(float dPhi)
{
  s.w = dPhi;
}

/*
* Returns current speed
*/
float getSpeed(void)
{
  return s.v;
}

/*
* Returns current rotational speed
*/
float getRotation(void)
{
  return s.w;
}


/*
* Turn angle radians on spot with rotational speed rotspeed
*/
void motorDriveTurn(int direction, float angle, float rotspeed)
{
  //temporary: calibrate turning (TODO: fix)
  angle=angle*1.03;

  while(angle>2*M_PI)
    angle-=2*M_PI;
  while(angle<-2*M_PI)
    angle+=2*M_PI;
  OSWait(40);
  s.w = rotspeed;
  int result = VWDriveTurn(vw, angle, rotspeed);
  if(result == -1){
  LCDSetPrintf(4,0,"Error Wrong VWhandle");
  }
  VWDriveWait(vw);
}

/*
* Turn angle radians on a segment that is diameter*angle long in the specified direction (1 = right, -1 =left)
*/
void motorCurveTurn(float diameter, float angle, int direction, float speed)
{
  OSWait(40);
  s.v = speed;
  int result = VWDriveCurve(vw,diameter*(angle/(2*M_PI)),direction*angle,speed);
  if(result == -1){
  LCDSetPrintf(4,0,"Error Wrong VWhandle");
  }
  VWDriveWait(vw);
}

/*
* Test function
*/
void motorDriveSquare(float height, float width, int direction, float speed)
{
  OSWait(40);
  s.v = speed;
  VWDriveStraight(vw,height,speed);
  VWDriveWait(vw);
  VWDriveTurn(vw,direction*M_PI/2,speed);
  VWDriveWait(vw);
  VWDriveStraight(vw,width,speed);
  VWDriveWait(vw);
  VWDriveTurn(vw,direction*M_PI/2,speed);
  VWDriveStraight(vw,height,speed);
  VWDriveWait(vw);
  VWDriveTurn(vw,direction*M_PI/2,speed);
  VWDriveWait(vw);
  VWDriveStraight(vw,width,speed);
  VWDriveWait(vw);
  VWDriveTurn(vw,direction*M_PI/2,speed);
  VWDriveWait(vw);
}

/*
* Stop the motors
*/
```

```
void motorStop(void)
{
  s.v = 0;
  s.w = 0;
  VWSetSpeed(vw,0,0);
  OSWait(40);
}
void motorRelease() {
  VWRelease(vw);
}
void motorGoTo(int startx, int starty, float startang, int goalx,int goaly, float goalang) {

}

/*
* Avoid setting speed below 0 and avoid values that are really high
*/
float motorDriveStraight(float length, float speed)
{
  float remain = 10.0;
  float newremain;

  OSWait(40);
  s.v = speed;
  int result = VWDriveStraight(vw, length, speed);
  if(result == -1){
  LCDSetPrintf(4,0,"Error Wrong VWhandle");
  }
  OSWait(100); /* Wait for VW to start */
  while(ABS(remain) > 0.01) {
    newremain = VWDriveRemain(vw);
  printf("Remains: %.4f\n", newremain);
    if(ABS(newremain-remain) < 0.01) {
      break;
    }
    remain = newremain;
    OSWait(20);
  }
  motorStop();
  return (ABS(newremain));
}
```

motor.h

```
/*
   motor.h
   Copyright ras9.
*/
#ifndef __MOTOR_H__
#define __MOTOR_H__

#define ABS(a) (((a) < 0) ? -(a) : (a))

void motorInitialize(float Vv, float Tv, float Vw, float Tw);
void changeSpeed(float dSpeed);
void changeRotation(float dPhi);
void setSpeed(float dSpeed);
void setRotation(float dPhi);
float getSpeed(void);
float getRotation(void);
float motorDriveStraight(float lenght, float speed);
void motorDriveTurn(int direction, float angle, float rotspeed);
void motorCurveTurn(float diameter, float angle, int direction, float speed);
void motorDriveSquare(float height, float width, int direction, float speed);
void motorStop(void);
void motorRelease(void);

#endif
```

B.1.6 GPIO

digitalout.c

```
/*
  digitalout.c
  Routines to write to GPIO ports (Where we have a LED connected for debug purposes).
  Copyright ras9
*/

#include "digitalout.h"
```

```
#include "eyebot.h"


void blinkLED()
{
  turnOnLED();
  OSWait(10);
  turnOffLED();
  OSWait(4);
  turnOnLED();
  OSWait(10);
  turnOffLED();
}

BYTE turnOnLED()
{
  return OSWriteOutLatch(0, 0xfe, 0x1);
}


BYTE turnOffLED()
{
  return OSWriteOutLatch(0, 0xfe, 0x0);
}


BYTE turnOnRoller()
{
  return OSWriteOutLatch(0, 0xfd, 0x2);
}


BYTE turnOffRoller()
{
  return OSWriteOutLatch(0, 0xfd, 0x0);
}
```

digitalout.h

```
/*
  digitalout.h
  Copyright ras9
*/

#ifndef _DIGITALOUT_H_
#define _DIGITALOUT_H_

#include "eyebot.h"

BYTE turnOnLED();
BYTE turnOffLED();
BYTE turnOnRoller();
BYTE turnOffRoller();

void blinkLED();


#endif
```

B.1.7 Helpers

helpers.c

```
/*
  helpers.c
  Math helpers routines.
  Copyright ras9
*/

#include "helpers.h"
#include "perkamera/trig.h"
#include <math.h>


int abs(int a) {
  if(a>0)
    return a;
  return -a;
}
```

```
float rsin(float ang) {
  int intang = (int) ((ang/(2*M_PI))*360);
  intang=intang%360;
  if(intang<0)
    intang+=360;
  return vsin[intang];
}

float rcos(float ang) {
  int intang = (int) ((ang/(2*M_PI))*360);
  intang=intang%360;
  if(intang<0)
    intang+=360;
  return vcos[intang];
}

float ratan(float inval) {
  int intval = (int) (inval*100);
  intval+=400;
  if(intval<0)
    return -1.325;
  else if(intval>=800)
    return 1.325;
  else
    return vatan[intval];
}
```

helpers.h

```
#ifndef __HELPERS_H__
#define __HELPERS_H__

int abs(int a);
float rsin(float ang);
float rcos(float ang);
float ratan(float inval);



#endif
```

B.2 Host side

B.2.1 Debug host

main.c

```
/* debug.c
   File to read and save debug data from grrobot. Currently also draws an SDL image with the coords received by grrrobot (ie,
   Copyright ras9
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <math.h>
#include <assert.h>
#include <limits.h>
#include <SDL/SDL_main.h>
#include <SDL/SDL.h>

#include "debug.h"


#define FIELDLEN 237
#define FIELDWIDTH 117

void fielddrawer(int matrix[FIELDWIDTH][FIELDLEN+20][3], int robx, int roby, float robang, int ballx, int bally) ;
static int readIntFromSer(FILE *pipe);
static float readFloatFromSer(FILE *pipe);

static void writePPM(FILE *fp,unsigned char *buffer,int w,int h,int len);

/*
 * Set the pixel at (x, y) to the given value
 * NOTE: The surface must be locked before calling this!
 */
void putpixel(SDL_Surface *surface, int x, int y, Uint32 pixel)
```

```
{
    int bpp = surface->format->BytesPerPixel;
    /* Here p is the address to the pixel we want to set */
    Uint8 *p = (Uint8 *)surface->pixels + y * surface->pitch + x * bpp;

    switch(bpp) {
    case 1:
        *p = pixel;
        break;

    case 2:
        *(Uint16 *)p = pixel;
        break;

    case 3:
        if(SDL_BYTEORDER == SDL_BIG_ENDIAN) {
            p[0] = (pixel >> 16) & 0xff;
            p[1] = (pixel >> 8) & 0xff;
            p[2] = pixel & 0xff;
        } else {
            p[0] = pixel & 0xff;
            p[1] = (pixel >> 8) & 0xff;
            p[2] = (pixel >> 16) & 0xff;
        }
        break;

    case 4:
        *(Uint32 *)p = pixel;
        break;
    }
}

void siginthandler(int a)
{
  exit(-1);
}

int fgetc_block(FILE *fp)
{
  int data = 0;
  while((data = fgetc(fp)) == EOF) {}
  return data;
}
#define MAX_MSG_SIZE 76033
#define MAGIC_NUMBER 0xCAFEBABE
int main(int argc,char*argv[])
{
  FILE *pipe;
  FILE *logfile;
  unsigned char buffer[MAX_MSG_SIZE];
  char path[PATH_MAX];
  int imageindex = 0;
  int mapindex = 0;
  int verbose = 0;
  int pathendpos;
  int magicnumber;
  SDL_Surface *screen;
  int surfaceBuffer[FIELDWIDTH][FIELDLEN+20][3];

  if(argc < 3) {
    printf("Usage: <%s> logfile imageprefix -v\n",argv[0]);
    return 0;
  }
  if(argc > 3) {
    if(strcmp(argv[3],"-v") == 0) {
      verbose = 1;
    }
  }
  /*Initialize SDL */

  SDL_Init ( SDL_INIT_VIDEO );
    atexit(SDL_Quit);
  screen = SDL_SetVideoMode(FIELDWIDTH, FIELDLEN+20, 32, SDL_SWSURFACE);

  logfile = fopen(argv[1],"w");
  signal(SIGINT,siginthandler);
  if(!logfile) {
    perror("Can't open logfile:");
    return -1;
  }

  //this is a security hole.
  strcpy(path,argv[2]);
  pathendpos = strlen(path);
```

```
  pipe = popen("./scriptread.sh","r");

  while(1) {
    int type;
    int timestamp;
    int len;
    int i;

    //read type
    //read timestamp
    //read len
    //read data
    //log to file or external image.

    magicnumber = readIntFromSer(pipe);
    if(magicnumber != MAGIC_NUMBER) {
      continue;
    }

    type = fgetc_block(pipe);

    timestamp = readIntFromSer(pipe);
    len = readIntFromSer(pipe);

    if(verbose) {
      printf("Received a message with type %c, timestamp: %d, len: %d\n",type,timestamp,len);
    }
    if(len > MAX_MSG_SIZE) {
      printf("Too big message(%d), aborting!\n",len);
      goto _exit;
      exit(-1);
    }
    for(i = 0; i < len; i++) {
      buffer[i] = fgetc_block(pipe);
    }

    if(type == DEBUG_COORDS) {
      int ballx,bally,robx,roby;
      float robang;
      int x,y;


      sscanf(buffer,"%d,%d,%d,%d,%f\n",&ballx,&bally,&robx,&roby,&robang);
      fielddrawer(surfaceBuffer /*matrix[FIELDWIDTH][FIELDLEN+20][3]*/, robx, roby, robang,
ballx, bally);
      SDL_LockSurface(screen);
      for(x = 0; x < FIELDWIDTH;x++) {
        for(y = 0; y < FIELDLEN+20;y++) {
          putpixel(screen,x,y,SDL_MapRGB(screen->format,surfaceBuffer[x][y][0],surfaceBuffer[x][y][1],surfaceBuffer[x][y][2]))
        }
      }
      SDL_UnlockSurface(screen);
        SDL_UpdateRect(screen, 0,0, 0, 0);
      sprintf(path,"%s_map%.3d.bmp",argv[2],mapindex);
      SDL_SaveBMP(screen,path);
      mapindex++;
    } else if(type == DEBUG_TEXT) {
      buffer[i+1] = '\0';
      fprintf(logfile,"[%c(%db)]@[%d]> %s",type,len,timestamp,buffer);
      if(verbose) {
        printf("[%c(%db)]@[%d]> %s",type,len,timestamp,buffer);
      }
    } else if(type == DEBUG_IMAGE) {
      int w,h;
      FILE *imagefp;
      //convert to correct format.
      sprintf(path,"%s%.3d.ppm",argv[2],imageindex);
      fprintf(logfile,"[%c(%db)]@[%d]> Saved at %s\n",type,len,timestamp,path);
      if(verbose) {
        printf("[%c(%db)]@[%d]> Saved at %s\n",type,len,timestamp,path);
      }
      imageindex++;
      imagefp = fopen(path,"w");
      assert(imagefp);
      if(len == 76032) {
        w = 176;
        h = 144;
      } else {
        w = 82;
        h = 62;
        }
      writePPM(imagefp,buffer,w,h,len);
      fclose(imagefp);
```

```
    }
  }

_exit:
  pclose(pipe);
  fclose(logfile);

  return 0;
}

static int readIntFromSer(FILE *pipe)
{
  unsigned char data[4];
  int i;

  for(i = 0;i < 4;i++) {
    data[i] = fgetc_block(pipe);
  }
  return data[0]<<0 | data[1]<<8 | data[2] << 16 | data[3] << 24;
}

static float readFloatFromSer(FILE *pipe)
{
  unsigned char data[4];
  int i;

  for(i = 0;i < 4;i++) {
    data[i] = fgetc_block(pipe);
  }
  return data[0]<<0 | data[1]<<8 | data[2] << 16 | data[3] << 24;
}

static void writePPM(FILE *fp,unsigned char *buffer,int w,int h,int len)
{
  int i,j;

  fprintf(fp,"P3\n%d %d\n255\n",w,h);
  if(len != w*h*3) {
    printf("Invalid image(%d,%d,%d)\n",w,h,len);
    return;
  }
  for(i=0,j=0;i<w*h;i++,j+=3) {
    fprintf(fp,"%u %u %u\n",buffer[j],buffer[j+1],buffer[j+2]);
  }
}
```

coltest.c

```
/*
usage:
gcc -lm coltest.c
./a.out < bildfil.ppm > nybildfil.ppm

*/
#define FIELDLEN 237
#define FIELDWIDTH 117
#define GOALDIST 38
#define GOALWIDTH 40
#define FOV (M_PI_4*1.3)
#define CLASSIFYRESOLUTION 32

#include <math.h>

int eRobX, eRobY, eBallX, eBallY;
float eRobAng;

void setpixelfield(int pic[FIELDWIDTH][FIELDLEN+20][3], int x, int y, int r, int g, int b) {
  pic[x][y][0]=r;
  pic[x][y][1]=g;
  pic[x][y][2]=b;
}

void fielddrawer(int matrix[FIELDWIDTH][FIELDLEN+20][3], int robx, int roby, float robang, int ballx, int bally) {
  int x,y;
  float ang;
  int balldist;
  //printf("ballx: %d, bally: %d",ballx,bally);
  bally+=10;
  for(x=0; x<FIELDWIDTH; x++) {
    for(y=0; y<FIELDLEN+20; y++) {
      setpixelfield(matrix,x,y,0,0,0);
      if(x>GOALDIST&&x<FIELDWIDTH-GOALDIST&&y<=10)
  setpixelfield(matrix,x,y,0,0,255);
      if(x>GOALDIST&&x<FIELDWIDTH-GOALDIST&&y>=FIELDLEN+10)
```

```
  setpixelfield(matrix,x,y,255,255,0);

    }
  }
  for(x=0; x<FIELDWIDTH; x++) {
    for(y=10; y<FIELDLEN+10; y++) {
      setpixelfield(matrix,x,y,0,255,0);
       ang = atan((float)(x-(robx))/(float)(y-(roby+10)));

       if(ang>robang-FOV/2&&ang<robang+FOV/2) {
   setpixelfield(matrix,x,y,200,255,200);
       }
       balldist=(x-ballx)*(x-ballx)+(y-bally)*(y-bally);
       if(balldist<15)
   setpixelfield(matrix,x,y,255,0,0);
     }
  }

  for(x=robx-3; x<robx+3; x++) {
     for(y=roby-3+10; y<roby+3+10; y++) {
       if(x>=0&&x<FIELDWIDTH&&y>=0&&y<FIELDLEN+10)
   setpixelfield(matrix,x,y,120,120,120);
     }
  }
}
```

scriptread.sh

```
#!/bin/sh
#Ugly script ripoff from "dl" to be able to receive data the same way as "dl" does.
stty -F /dev/ttyS0 "0:0:80001cb2:8a38:3:1c:7f:15:4:0:1:0:11:13:1a:0:12:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0"
cat /dev/ttyS0
```