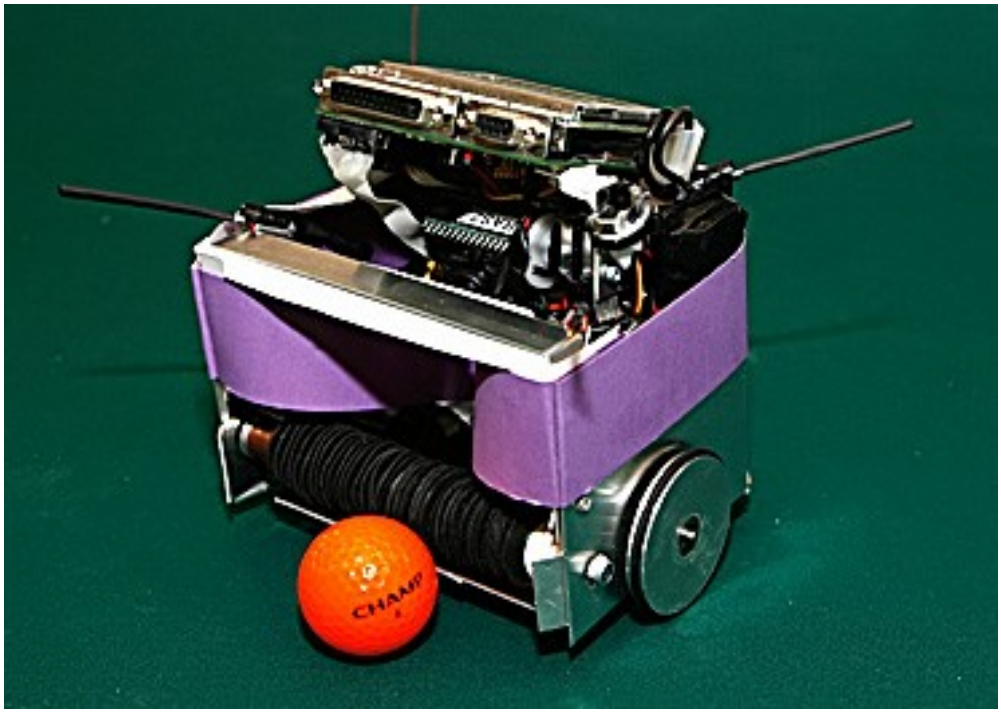
# RAAS

## Really Advanced Artificial Stupidity

Project report
2D1426 Robotics and Autonomus Systems




Per Eriksson
Johannes Edrén
Joakim Ekman
John Larsson

June 2007-06-12
Royal Institute of Technology Nada 2007

## Abstract

“Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.” Antoine De Saintexupery
This project was done as a part of the course 2D1426 Robotics and Autonomus Systems. The goal of this project was to create autonomus system that unopposed could score at least one goal in a known environment, followed by a friendly tournament between all the groups. Given the fact that the more code you write the more debugging will be needed, we decided to make the robot as simple and robust as possible. In a controlled environment like the 120x240 cm soccer field with walls a robot doesn’t need to know where it is it only needs to know where it is supposed to go. With this motto we created RAAS, Really Advanced Artificial Stupidity. Even though it only finished in second place by the overall performance we proved that less is more.

# Contents

# 1 Background

This project was done as a part of the course 2D1426 Robotics and Autonomus Systems or RAS. The name RAAS Really Advanced Artificial Stupidity was given to the robot because it got some advanced features and because the code is written to be as simple and robust as possible therefore resulting in a stupid behaviour sometimes.

## *1.1 Project Goals*

The goal of this project was to create autonomus system that unopposed could score at least one goal in a known environment but without knowing where the ball would be placed in this environment this was followed by a friendly tournament between all the groups.

## *1.2 Equipment*

- An Eyebot controller board with 35 MHz 32-bit Motorola 68332 microcontroller and 2 MB RAM and 512 KB flash ROM
- Digital camera with resolution 176x144 pixels
- Batteries and a charger
- Two motors with encoders and gearboxes
- RS-232 serial and BDM parallel cables
- An R/C servo
- 3 sheets of aluminum
- A few micro switches
- A sheet of neoprene rubber
- A motor from a toy
- Some ball bearings
- a central door locking actuator
- A small workshop

# 2 General Approach

Our goal was a very simple and effective robot with few parts and good design. We also included as the only team a separate kicker and an extremely good working roller witch gave the robot very good ball control.

# 3 The Robotics

## *3.1 Sensing*

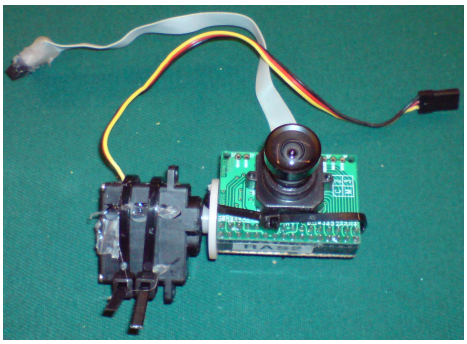


Figure 1 camera with tilting servo

A colour ccd camera witch is the main sensor for ball and goal localisation. See figure 1.
Four "whiskers" in each corner of the robot. Primary used to handle obstacles like opponent and walls. The roller is also a sensor. When the ball is in contact with the roller the current through the roller motor is higher, therefore the voltage over the transistor controlling the motor will also be higher. This voltage is measured with the onboard AD converter and easy ball detection without the camera is acquired. This sensor makes the code and programming much simpler. Wheel decoders to roughly estimate position on field and movement control.

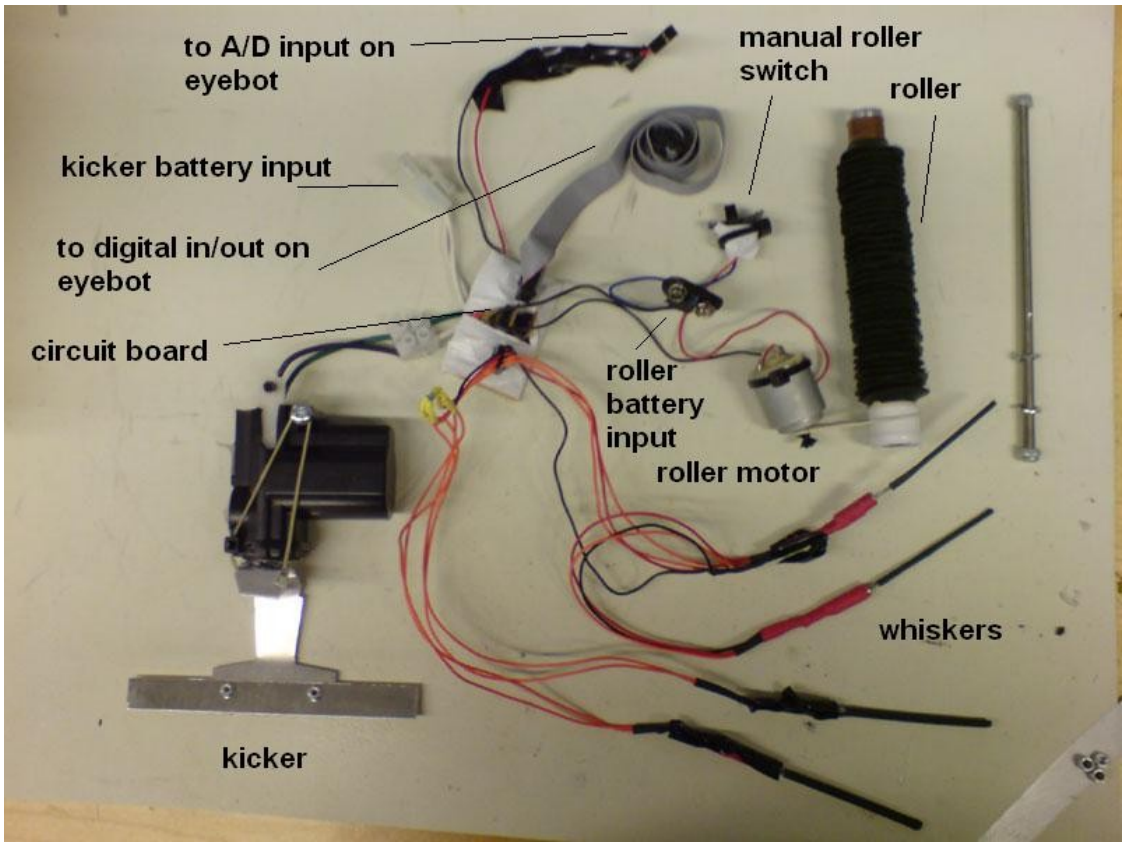



Figure 2. Extension circuit board controlling roller, kicker and handling whisker-sensors and roller sensor

## *3.2 Ball control*

### Roller

The roller is the “secret” weapon of RAAS. The rules of the roller design stated that a roller with variable diameter had to be mounted as the same height as the centre line of the ball. A roller with equal diameter was allowed to be mounted higher thus improving the backspin effect. The advantage of a roller with variable diameter is that the roller will have the effect of keeping the ball in the middle of the roller. The roller design of RAAS is a equal diameter roller with the centring effect that is allowed to be mounted at a different height for improved backspin.

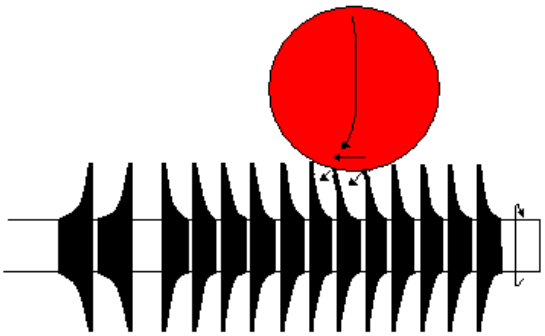


Figure 3. The function of the centring effect of the roller.

The roller was made from a piece of copper pipe with ball bearings. Over the pipe a number of neoprene rubber washers were placed so that 50% of them are facing to the right and the rest to the left. The direction of the washers is of much importance as giving the desired centring effect of the ball. See fig 3. The rubber washers will deflect when in contact with the ball and due to the direction of deflection push the ball in the same direction.

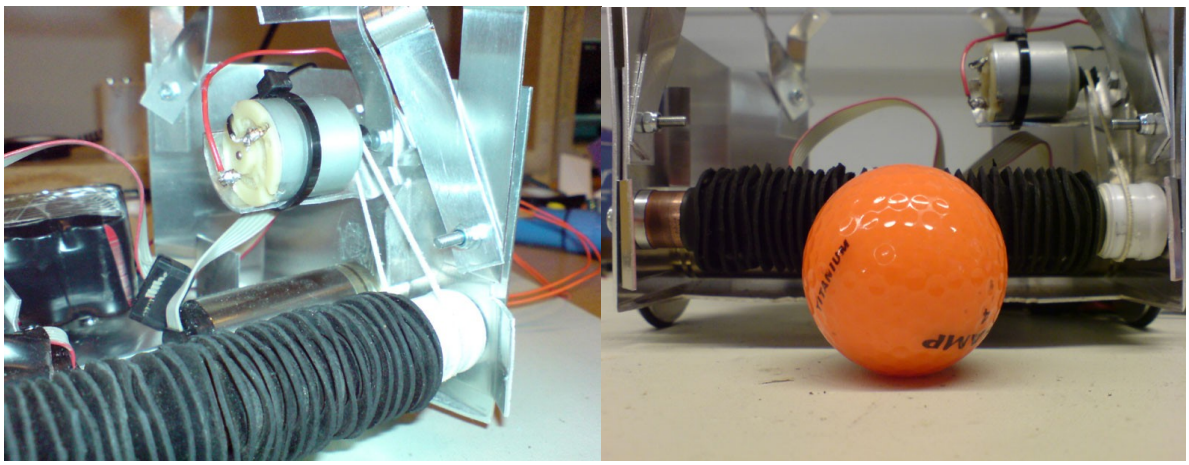


Figure 4. Roller with motor mounted in RAAS

The roller was driven with a motor via a rubber band giving a reduction of rotation speed between motor and roller rpm. The roller motor vas mounted to a bracket bolted together with the chassis. See fig 4. The roller has its own battery pack of 3.6V. The connection between motor and battery is controlled by a MOSFET and the voltage drop over the FET is connected to the onboard AD-converter.

### Kicker

The kicker gives the advantage to score a goal without having to drive all the way into the goal. The kicker has also its own battery pack with 8.4v. It is controlled I the same manner as the roller via the digital output controlling a transistor. The mechanism is a central door locking actuator modified to fit into the robot. See fig 5. The “shoe” of the kicker is made from aluminium and rests underneath the roller. When the robot “kicks” the backspin of the ball is not longer an advantage. To get a better kick the roller is therefore deactivated a short moment before the kick.

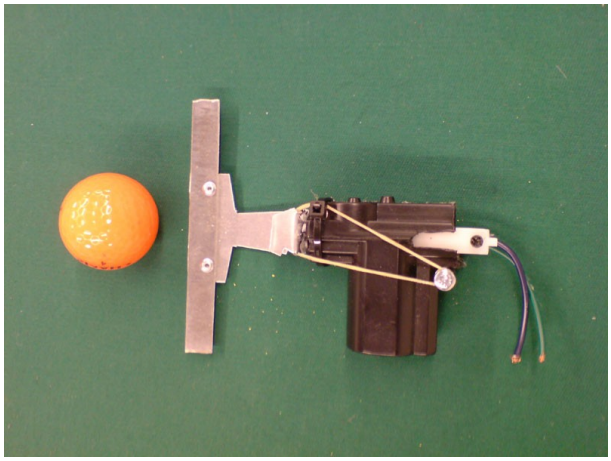


Figure 5 Kicker made from door locking mechanism with rubber return spring.

## *3.3 Hardware Construction*

### Chassis design

The chassis is made from basically one piece aluminium sheet. It was bent into an u-shape witch gave the robot a strong base with side protection, low centre of gravity and a lot of space for batteries, electronics, and actuators. See fig 6. The controller board was bolted on top with a few strips of aluminium. The roller was mounted directly in front of the motors and the kicker in between the motors along the centre line of the robot.

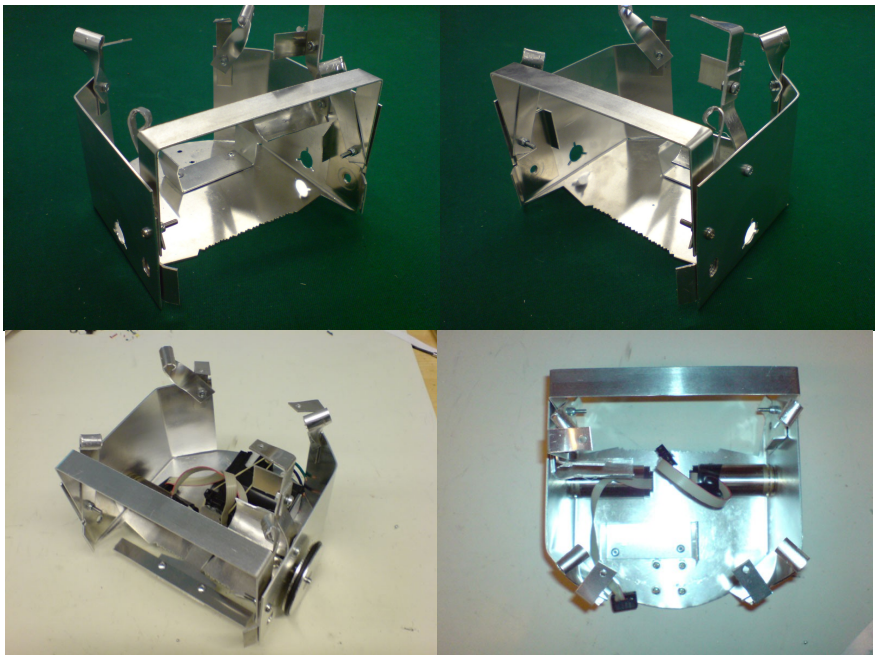


Figure 6. Aluminium skeleton if RAAS.

## Locomotion

Differential drive with two high quality motors with decoders. Aluminium wheels with o-ring tires. A third custom made castor wheel with ball bearings and very low friction. See fig 7.

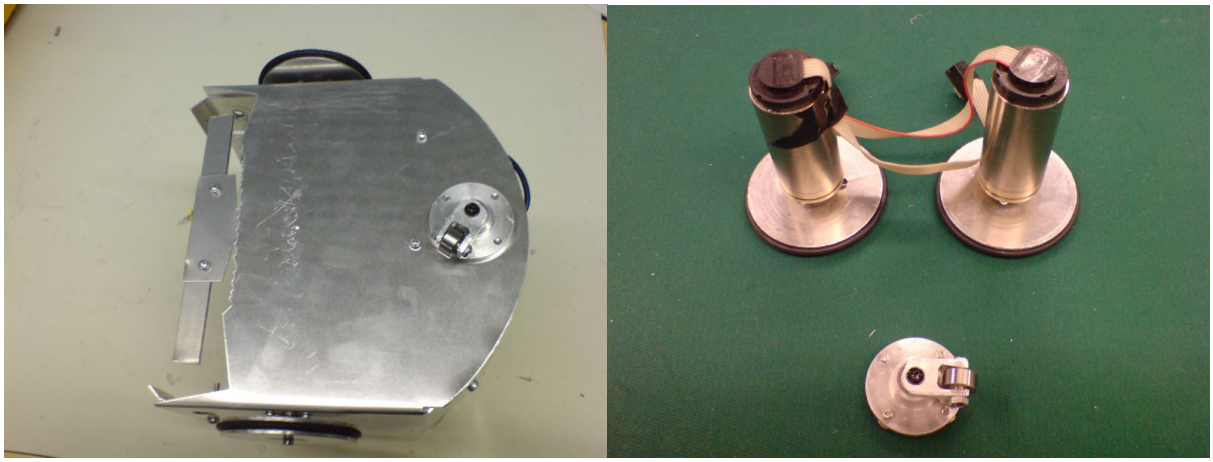


Figure 7. Wheels on RAAS

# 4 Programming and Tactics

## General idea

We decided early on to avoid making any part of the robot overly complex. This is reflected in the code, where ugly and stupid is preferred over elegant and complex, simply because the former was easier to get to a working state where things could be tested. Undoubtedly, elegant would have been better from a code maintainability viewpoint, but this was sacrificed to save some precious time.

The software solution consists of two general parts, a PC program and the actual robot software. The PC part was constructed in order to quickly get images from the robot, and construct a colour lookup table to send back to the robot. This table was then used to find objects in the images acquired with the camera. This solution allowed us to quickly switch lookup tables in the robot as lighting conditions varied. Communication was conducted over the serial connection to the EyeBot controller. The PC program (written in Java) will not be included in this report due to the fairly large amount of code involved in it, most of them dealing with Swing GUI building.

The robot software implements a simple state machine. No navigation scheme is implemented as we didn't feel we would gain much from it. The robot simply steers directly for its target when it appears in the image. We chose not to take the opponent into account, with the primary reason being that the number of different actions to perform, depending on the opponents and our own location, would become unreasonably large if we were to gain any practical advantage from it. Instead, we decided that stubbornness bordering on stupidity would be simpler and probably almost as effective. This proved to work well in both qualification and competition.

## EyeBot features used

We utilized the *vw*-drive to drive our robot around the field. Initially we had some problems running the *vw*-drive along with the camera, but a solution was found in the EyeBot documentation. For the camera we used the full frame on every occasion, since the time spent on processing the frames were negligible in comparison the time required to get the frame from the camera. The frames were then sub sampled and matched in the colour lookup table in one go, resulting in a smaller bitmap of interpreted pixels. This frame was then searched to find the appropriate matching objects, such as ball and goals. From these locations, an angle was estimated, and manoeuvring decisions made according to this. The ball was deemed to be acquired when the roller motor current passed a certain threshold, and the goal was judged to be close enough when it grew above a certain size in the frame.

## Control flow

Figure 8. illustrates the control flow in the robot software. The blue boxes in every state denotes something that occurs when entering the state, and the yellow boxes denotes something that occurs when leaving the state. The arrow colour codes indicates what triggers a state transition (either to another state or back to itself). As is apparent from the graph, the state machine is simple, and the robot focuses on the ball above all else. Driving with the ball is done in an asynchronous fashion, with *VWSetSpeed()* rather than *VWDriveStraight/Turn/Curve()*. We found it easier to get the robot responsive and efficient in this way, as the synchronous functions of the *vw*-drive API behaved rather oddly from time to time.

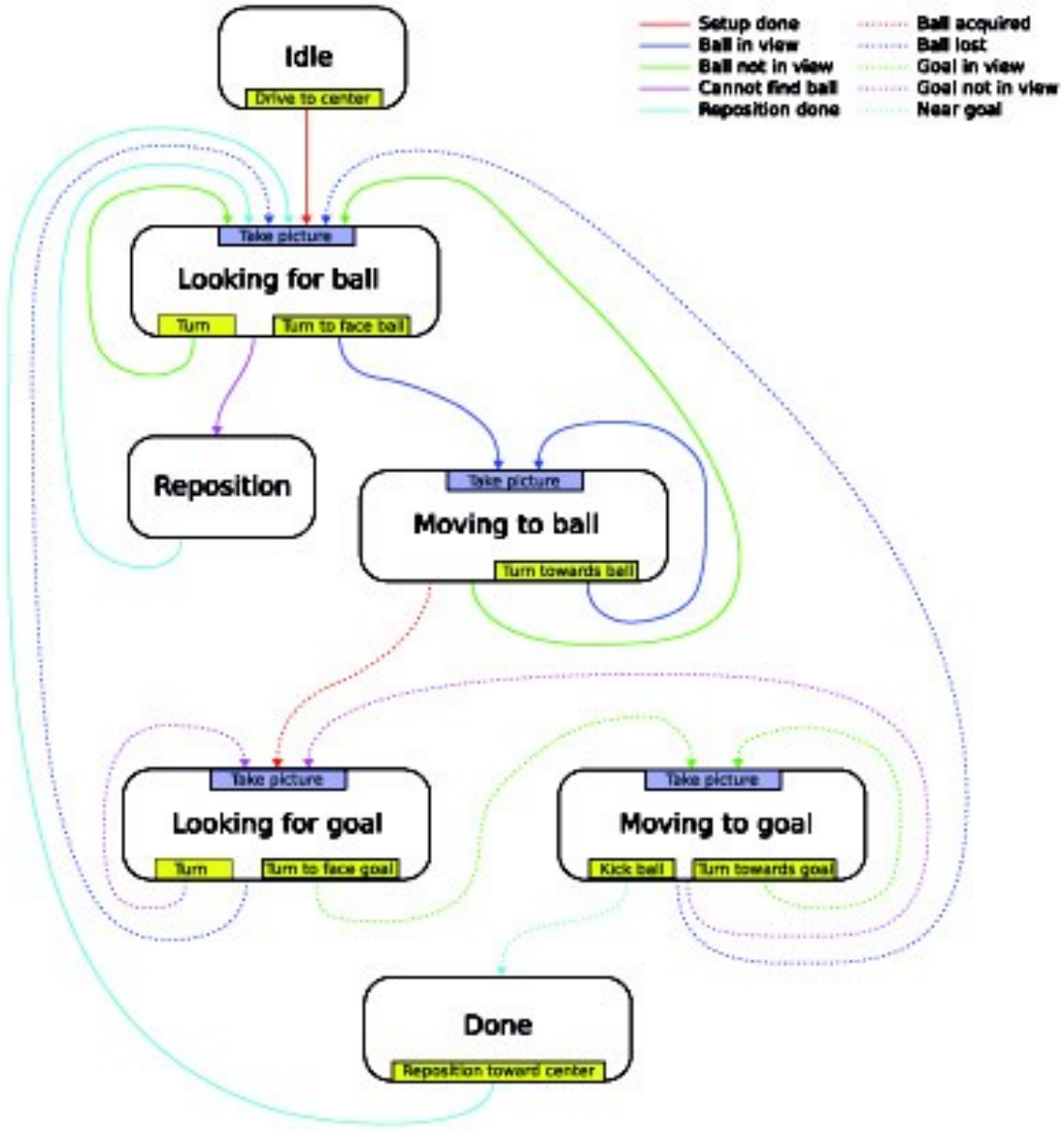



Figure 8. Control flow graph

# 5 Results

The final design of RAAS performed very well on the competition and was unmatched in its ability at making goals. It also worked very well against opponents.

Sadly we had extremely bad luck as the outcome of the final against Ballblaster resulted in an overall second place. But the overall performance of RAAS gave the general impression of beeing the best robot in the competition.

# 6 Conclusions

The idea of keeping simple and robust gave us time for testing and updating both hardware and software rather than spending time implementing new features. The relocating feature seen in the control flow graph on page 9 was implemented as drive 120cm and turn away from walls resulted in that the robot almost always ended up in an position where it could see the ball. Thus allot of time was saved in not using navigation and mapping both in the development and in execution time in the games.

The roller was also a success in both ball control and sensing ball possession. The very good ball control would enable us to move faster with the ball and to steal the ball from opponents. We also could pick up the ball from corners and walls without adding extra code.

The kicker could be thought of as a funny accessory, but it gave us a huge advantage when scoring a goal in the fact that the robot was closer to the centreline where the ball would be reinserted. This made it easier to locate the ball after a goal and a enabled us to move to the ball in less time and even pick it up without the opponent knowing what happened.

Judging from the results this was a successful design and the general idea the less is more turned out to be correct.

# 7 Appendix

```
/*
 ***************
 * hunter.c
 ***************
 */

#include "eyebot.h"
#include "fastmath.h"
#include "imageTools.h"
#include "commTools.h"
#include "system.h"
#include "eyebotError.h"
#include "ui.h"


#define STATE_NOSTATE -1
#define STATE_IDLE 0
#define STATE_LOOKING_FOR_BALL 1
#define STATE_REPOSITION 2
#define STATE_MOVING_TO_BALL 3
#define STATE_LOOKING_FOR_GOAL 4
#define STATE_MOVING_TO_GOAL 5
#define STATE_DONE 6

#define SPEED_NORMAL 0.17


fullframe_t fullframe;
interpretedframe_t interpreted;
lookuptable_t table;
ServoHandle servo;
short ballADThreshold = 280;
short stalledADThreshold = 510;

int ballX, ballY, ballSize, blueX, blueY, blueSize, yellowX, yellowY, yellowSize;
int goal = BLUE_COLOR;


int setup()
{
    while(1)
    {
        int ret = menu("Setup\n", "Go", "Comm", "Goal", "Exit");
        switch(ret)
        {
            case KEY1:
                return 0;
                break;
            case KEY2:
                while((ret = menu("Communication\n", "Full", "Int", "Tbl", "Back")) != KEY4)
                {
                    switch(ret)
                    {
                        case KEY1:
                            getFullFrame(fullframe);
                            LCDPrintf("Sending\n");
                            sendFullFrame(fullframe);
                            break;
                        case KEY2:
                            getFullFrame(fullframe);
                            getInterpretedFrame(interpreted, fullframe);
                            LCDPrintf("Sending\n");
                            sendInterpretedFrame(interpreted);
                            break;
                        case KEY3:
                            if(!recvLookupTable(table))
                            {
                                setLookupTable(table);
                            }
                            else
                            {
                                LCDPrintf("Error\n");
```

```
                            }
                            break;
                        }
                    }
                    break;
                case KEY3:
                    if(goal == BLUE_COLOR)
                    {
                        goal = YELL_COLOR;
                    }
                    else
                    {
                        goal = BLUE_COLOR;
                    }
                    LCDPrintf("%s goal\n", ((goal == BLUE_COLOR) ? "Blue" : "Yellow"));
                    OSSleep(100);
                    break;
                case KEY4:
                    return -1;
                    break;
            }
        }
    }

void init(VWHandle* vw)
{
    *vw = VWInit(VW_DRIVE, 1); /* init v-omega interface */
    servo = SERVOInit(CAM_SERVO);
    SERVOSet(servo, 165);
    if(vw == 0)
    {
        error("VWInit Error!\n");
        OSWait(200);
    }
    //VWStartControl(*vw, 5.0, 0.3, 2.9, 0.1);
    VWStartControl(*vw, 7, 0.3, 7, 0.1);
    initComms();
    initCam();
}


void rollerOn()
{
    OSWriteOutLatch(0, 0xFF, 0x80);
}

void rollerOff()
{
    OSWriteOutLatch(0, 0x7F, 0x00);
}

int rollerStalled()
{
    static short prev = 0;
    prev = (prev >> 1) + (OSGetAD(ROLLER_AD_CHANNEL) >> 1);
    return (prev > stalledADThreshold);
}

void kick()
{
    rollerOff();
    AUTone(400, 500);
    OSSleep(50);
    OSWriteOutLatch(0, 0xFF, 0x40);
    OSSleep(30);
    OSWriteOutLatch(0, 0xBF, 0x00);
}

unsigned char bumpCheck()
{
    return (OSReadInLatch(1) ^ 0xF0) & 0xF0;
}



void doCapture()
{
    getFullFrame(fullframe);
    getInterpretedFrame(interpreted, fullframe);
```

```
    findObjects(
            interpreted,
            &ballX,
            &ballY,
            &ballSize,
            &blueX,
            &blueY,
            &blueSize,
            &yellowX,
            &yellowY,
            &yellowSize);
}


// gives angle and distance to ball
int hasBall(float* angle, float* dist)
{
    static short prev = 0;
    int has = 0;
    prev = (prev >> 1) + (OSGetAD(ROLLER_AD_CHANNEL) >> 1);
    if(ballSize != -1)
    {
        has = 1;
        if(angle)
        {
            *angle = M_PI * (ballX - 40) / 180.0;
        }
        if(dist)
        {
            if(ballY >= 55)
            {
                *dist = 0;
            }
            else
            {
                *dist = 1;
            }
        }
    }
    if(prev > ballADThreshold)
    {
        has = 1;
        if(angle)
        {
            *angle = 0;
        }
        if(dist)
        {
            *dist = 0;
        }
    }
    return has;
}


// gives angle and distance to blue goal
int hasBlueGoal(float* angle, float* dist)
{
    if(blueSize != -1)
    {
        if(angle)
        {
            *angle = M_PI * (blueX - 40) / 180.0;
        }
        if(dist)
        {
            if(blueSize >= 68)
            {
                *dist = 0;
            }
            else if(blueSize >= 50)
            {
                *dist = 1;
            }
            else
            {
                *dist = 2;
            }
        }
```

```
        return 1;
    }
    else
    {
        return 0;
    }
}
// gives angle and distance to yellow goal
int hasYellowGoal(float* angle, float* dist)
{
    if(yellowSize != -1)
    {
        if(angle)
        {
            *angle = M_PI * (yellowX - 40) / 180.0;
        }
        if(dist)
        {
            if(yellowSize >= 68)
            {
                *dist = 0;
            }
            else if(yellowSize >= 50)
            {
                *dist = 1;
            }
            else
            {
                *dist = 2;
            }
        }
        return 1;
    }
    else
    {
        return 0;
    }
}
// gives angle and distance to selected goal
int hasGoal(float* angle, float* dist)
{
    if(goal == BLUE_COLOR)
    {
        return hasBlueGoal(angle, dist);
    }
    else
    {
        return hasYellowGoal(angle, dist);
    }
}
```

```
int main()
{
    int exit = 0, state = 0, driving = 0, iterations = 0;
    float remain;
    VWHandle vw;

    init(&vw);

    while(!exit)
    {
        float angle, dist;
        PositionType pos;
        LCDMenu("IDLE", "", "", "EXIT");
        int k = KEYRead();
        if(k == KEY4)
        {
            state = STATE_NOSTATE;
            exit = 1;
        }
        else if(k == KEY1)
        {
            state = STATE_IDLE;
            VWSetSpeed(vw, 0, 0);
            driving = 0;
        }
```

```
switch(state)
{
    case STATE_IDLE:
        rollerOff();
        if(setup() == -1)
        {
            exit = 1;
            break;
        }
        // Asks for starting point to make localisation and searching easier.
        int ret1 = menu("What side?", "Yelw", "", "", "Blue");
        if(ret1 == KEY1 || ret1 == KEY4)
        {
            int ret2 = menu("Where?", "Left", "Goal", "Goal", "Rght");
            switch (ret1)
            {
                case KEY1:
                    switch (ret2)
                    {
                        case KEY1:
                            VWSetPosition(vw, -1.1, 0.5, 0.0);
                            break;
                        case KEY2:
                        case KEY3:
                            VWSetPosition(vw, -1.2, 0.0, 0.0);
                            break;
                        case KEY4:
                            VWSetPosition(vw, -1.1, -0.5, 0.0);
                            break;
                    }
                    break;
                case KEY4:
                    switch (ret2)
                    {
                        case KEY1:
                            VWSetPosition(vw, 1.1, -0.5, M_PI);
                            break;
                        case KEY2:
                        case KEY3:
                            VWSetPosition(vw, 1.2, 0.0, M_PI);
                            break;
                        case KEY4:
                            VWSetPosition(vw, 1.1, 0.5, M_PI);
                            break;
                    }
                    break;
            }
        }
        rollerOn();
        VWDriveStraight(vw, 1.0, SPEED_NORMAL * 1.5);
        OSSleep(1);
        VWDriveWait(vw);
        OSSleep(1);
        VWDriveStraight(vw, 0.2, SPEED_NORMAL * 0.8);
        OSSleep(1);
        VWDriveWait(vw);
        OSSleep(1);
        VWDriveStraight(vw, -0.2, SPEED_NORMAL * 0.8);
        OSSleep(1);
        VWDriveWait(vw);
        OSSleep(1);
        VWSetSpeed(vw, 0, 0);
        state = STATE_LOOKING_FOR_BALL;
        break;
    case STATE_REPOSITION:
        //doCapture();

        LCDClear();
        LCDSetPos(0, 0);
        LCDPutString("Repositioning...\n");
        /*LCDPrintf(
            "Ball x%d y%d s%d\nBl x%d y%d s%d\nYe x%d y%d s%d\n",
            ballX, ballY, ballSize,
            blueX, blueY, blueSize,
            yellowX, yellowY, yellowSize);*/

        VWGetPosition(vw, &pos);
        //LCDPrintf("%3f %3f %3f\n", pos.x, pos.y, pos.phi);
```

```
        // If ball is found> go to ball
        VWDriveStraight(vw, 1.2, SPEED_NORMAL * 2);
        while(ABS((remain = VWDriveRemain(vw))) > 0.01)
        {
            unsigned char bump = bumpCheck();
            if((bump & BUMP_FRONT_LEFT) && (bump & BUMP_FRONT_RIGHT))
            {
                VWSetSpeed(vw, 0, 0);
                OSSleep(1);
                VWDriveTurn(vw, M_PI / 1.5, M_PI_4);
                OSSleep(1);
                VWDriveWait(vw);
                VWDriveStraight(vw, remain, SPEED_NORMAL * 1.6);
            }
            else if(bump & BUMP_FRONT_LEFT)
            {
                VWSetSpeed(vw, 0, 0);
                OSSleep(1);
                VWDriveTurn(vw, M_PI_2, M_PI_4);
                OSSleep(1);
                VWDriveWait(vw);
                VWDriveStraight(vw, remain, SPEED_NORMAL * 1.6);
            }
            else if(bump & BUMP_FRONT_RIGHT)
            {
                VWSetSpeed(vw, 0, 0);
                OSSleep(1);
                VWDriveTurn(vw, -M_PI_2, M_PI_4);
                OSSleep(1);
                VWDriveWait(vw);
                VWDriveStraight(vw, remain, SPEED_NORMAL * 1.6);
            }
        }
        state = STATE_LOOKING_FOR_BALL;
        break;
    case STATE_LOOKING_FOR_BALL:
        doCapture();

        LCDClear();
        LCDSetPos(0, 0);
        LCDPutString("Looking...\nNo ball\n");
        LCDPrintf(
            "Ball x%d y%d s%d\nBl x%d y%d s%d\nYe x%d y%d s%d\n",
            ballX, ballY, ballSize,
            blueX, blueY, blueSize,
            yellowX, yellowY, yellowSize);

        VWGetPosition(vw, &pos);
        //LCDPrintf("%3f %3f %3f\n", pos.x, pos.y, pos.phi);

        // If ball is found> go to ball
        if(hasBall(&angle, 0))
        {
            VWDriveTurn(vw, angle * 1.2, M_PI / 3);
            OSSleep(5);
            VWDriveWait(vw);
            iterations = 0;
            state = STATE_MOVING_TO_BALL;
        }
        // Turn and search again
        else
        {
            PositionType pos;
            VWGetPosition(vw, &pos);
            if((pos.x < 0 && pos.y < 0) || (pos.x > 0 && pos.y > 0))
            {
                VWDriveTurn(vw, M_PI / 6, M_PI_2);
            }
            else
            {
                VWDriveTurn(vw, -M_PI / 6, M_PI_2);
            }
            OSSleep(2);
            VWDriveWait(vw);
            if(iterations++ > 8)
            {
                iterations = 0;
                state = STATE_REPOSITION;
```

```
            }
        }
        break;
    case STATE_MOVING_TO_BALL:
        rollerOn();

        LCDClear();
        LCDSetPos(0, 0);
        LCDPutString("Moving to ball\n");
        LCDPrintf(
            "Ball x%d y%d s%d\nBl x%d y%d s%d\nYe x%d y%d s%d\n",
            ballX, ballY, ballSize,
            blueX, blueY, blueSize,
            yellowX, yellowY, yellowSize);

        VWGetPosition(vw, &pos);
        //LCDPrintf("%f %f %f\n", pos.x, pos.y, pos.phi);

        unsigned char bump = bumpCheck() & (BUMP_FRONT_LEFT | BUMP_FRONT_RIGHT);

        if(bump)
        {
            if((bump & BUMP_FRONT_LEFT) && (bump & BUMP_FRONT_RIGHT))
            {
                driving = 0;
                VWSetSpeed(vw, 0, 0);
                OSSleep(1);
                VWDriveStraight(vw, -0.2, 0.2);
                OSSleep(1);
                VWDriveWait(vw);
                state = STATE_LOOKING_FOR_BALL;
            }
            else if(bump & BUMP_FRONT_LEFT)
            {
                VWSetSpeed(vw, 0, 0);
                OSSleep(1);
                VWDriveTurn(vw, M_PI / 8, M_PI / 3);
                OSSleep(1);
                VWDriveWait(vw);
                VWSetSpeed(vw, SPEED_NORMAL, 0);
            }
            else if(bump & BUMP_FRONT_RIGHT)
            {
                VWSetSpeed(vw, 0, 0);
                OSSleep(1);
                VWDriveTurn(vw, -M_PI / 8, M_PI / 3);
                OSSleep(1);
                VWDriveWait(vw);
                VWSetSpeed(vw, SPEED_NORMAL, 0);
            }
        }
        else
        {
            doCapture();
            if(driving)
            {
                if(hasBall(&angle, &dist))
                {
                    // Ball is in posession > look for goal
                    if(dist == 0)
                    {
                        VWSetSpeed(vw, 0, 0);
                        driving = 0;
                        state = STATE_LOOKING_FOR_GOAL;
                    }
                    // Turn to home in on ball
                    else if((angle > M_PI / 16 && angle > 0) || (angle < -M_PI / 16 && angle < 0))
                    {
                        LCDPrintf("Angle: %f\n", angle);
                        VWSetSpeed(vw, 0, 0);
                        OSSleep(1);
                        VWDriveTurn(vw, angle, M_PI_2);
                        OSSleep(1);
                        VWDriveWait(vw);
                        OSSleep(1);
                        VWSetSpeed(vw, SPEED_NORMAL, 0);
                    }
                }
                else
```

```
            {
                VWSetSpeed(vw, 0, 0);
                driving = 0;
                state = STATE_LOOKING_FOR_BALL;
            }
        }
        else
        {
            driving = 1;
            VWSetSpeed(vw, SPEED_NORMAL, 0.0);
        }
    }
    break;
case STATE_LOOKING_FOR_GOAL:
    doCapture();

    LCDClear();
    LCDSetPos(0, 0);
    LCDPutString("Looking...\nHas ball\n");
    LCDPrintf(
        "Ball x%d y%d s%d\nBl x%d y%d s%d\nYe x%d y%d s%d\n",
        ballX, ballY, ballSize,
        blueX, blueY, blueSize,
        yellowX, yellowY, yellowSize);

    VWGetPosition(vw, &pos);
    //LCDPrintf("%f %f %f\n", pos.x, pos.y, pos.phi);

    // Ball is in posession
    if(hasBall(0, &dist) && dist == 0)
    {
        // If goal is found > go to goal
        if(hasGoal(&angle, 0))
        {
            VWDriveTurn(vw, angle, M_PI / 6);
            OSSleep(1);
            VWDriveWait(vw);
            state = STATE_MOVING_TO_GOAL;
        }
        // Turn and seach again
        else
        {
            VWGetPosition(vw, &pos);
            if((pos.x < 0 && pos.y < 0) || (pos.x > 0 && pos.y > 0))
            {
                VWDriveTurn(vw, M_PI_4 / 2, M_PI / 3);
            }
            else
            {
                VWDriveTurn(vw, -M_PI_4 / 2, M_PI / 3);
            }
            OSSleep(2);
            VWDriveWait(vw);
        }
    }
    else
    {
        state = STATE_LOOKING_FOR_BALL;
    }
    break;
case STATE_MOVING_TO_GOAL:
    LCDClear();
    LCDSetPos(0, 0);
    LCDPutString("Moving to goal\n");
    LCDPrintf(
        "Ball x%d y%d s%d\nBl x%d y%d s%d\nYe x%d y%d s%d\n",
        ballX, ballY, ballSize,
        blueX, blueY, blueSize,
        yellowX, yellowY, yellowSize);

    VWGetPosition(vw, &pos);
    //LCDPrintf("%f %f %f\n", pos.x, pos.y, pos.phi);

    if(rollerStalled())
    {
        VWSetSpeed(vw, 0, 0);
        driving = 0;
        if(hasGoal(0, 0))
        {
```

```
        kick();
        state = STATE_LOOKING_FOR_BALL;
    }
    else
    {
        VWDriveStraight(vw, -0.2, SPEED_NORMAL);
        OSSleep(1);
        VWDriveWait(vw);
    }
}
else if((VWStalled(vw)) || (bumpCheck() & (BUMP_FRONT_LEFT || BUMP_FRONT_RIGHT)))
{
    driving = 0;
    VWDriveStraight(vw, -0.15, SPEED_NORMAL / 2);
    OSSleep(1);
    VWDriveWait(vw);
    if(hasBall(0, 0))
    {
        VWDriveTurn(vw, M_PI_2, M_PI / 3);
        OSSleep(1);
        VWDriveWait(vw);
        OSSleep(1);
        VWDriveStraight(vw, -0.2, SPEED_NORMAL);
        float remain;
        while(ABS((remain = VWDriveRemain(vw))) > 0.01)
        {
            if(bumpCheck())
            {
                VWDriveTurn(vw, M_PI, M_PI / 3);
                OSSleep(1);
                VWDriveWait(vw);
                OSSleep(1);
                VWDriveStraight(vw, 0.2 + remain, SPEED_NORMAL);
                OSSleep(1);
                VWDriveWait(vw);
                OSSleep(1);
            }
        }
        state = STATE_LOOKING_FOR_GOAL;
    }
    else
    {
        state = STATE_LOOKING_FOR_BALL;
    }
    VWSetSpeed(vw, 0, 0);
}
else
{
    doCapture();
    if(hasBall(&angle, &dist))
    {
        if(dist == 0)
        {
            if(hasGoal(&angle, &dist))
            {
                if(dist == 1)
                {
                    driving = 0;
                    VWSetSpeed(vw, 0, 0);
                    kick();
                    state = STATE_DONE;
                }
                //TODO: Change
                else if((angle > M_PI / 16 && angle > 0) || (angle < -M_PI / 16 && angle < 0))
                {
                    LCDPrintf("Angle: %f\n", angle);
                    VWSetSpeed(vw, 0, 0);
                    OSSleep(1);
                    VWDriveCurve(vw, 0.12, angle / 2, SPEED_NORMAL * 0.75);
                    OSSleep(1);
                    VWDriveWait(vw);
                    OSSleep(1);
                    VWSetSpeed(vw, SPEED_NORMAL * 1.3, 0);
                }
                else
                {
                    VWSetSpeed(vw, SPEED_NORMAL * 1.3, 0);
                    driving = 1;
                }
```

```
                        }
                        else
                        {
                            VWSetSpeed(vw, 0, 0);
                            driving = 0;
                            state = STATE_LOOKING_FOR_GOAL;
                        }
                    }
                    else
                    {
                        VWSetSpeed(vw, 0, 0);
                        driving = 0;
                        state = STATE_LOOKING_FOR_BALL;
                    }
                }
                else
                {
                    state = STATE_LOOKING_FOR_BALL;
                }
            }
            break;
        case STATE_DONE:
            rollerOff();
            VWDriveStraight(vw, -0.5, SPEED_NORMAL);
            while(ABS((remain = VWDriveRemain(vw))) > 0.01)
            {
                unsigned char bump = bumpCheck();
                if(bump)
                {
                    if((bump & BUMP_BACK_LEFT) && (bump & BUMP_BACK_RIGHT))
                    {
                        VWSetSpeed(vw, 0, 0);
                        OSSleep(1);
                        state = STATE_LOOKING_FOR_BALL;
                    }
                    else if(bump & BUMP_BACK_LEFT)
                    {
                        VWSetSpeed(vw, 0, 0);
                        OSSleep(1);
                        VWDriveTurn(vw, -M_PI / 8, M_PI_4);
                        OSSleep(1);
                        VWDriveWait(vw);
                        VWDriveStraight(vw, remain, SPEED_NORMAL);
                    }
                    else if(bump & BUMP_BACK_RIGHT)
                    {
                        VWSetSpeed(vw, 0, 0);
                        OSSleep(1);
                        VWDriveTurn(vw, M_PI / 8, M_PI_4);
                        OSSleep(1);
                        VWDriveWait(vw);
                        VWDriveStraight(vw, remain, SPEED_NORMAL);
                    }
                }
            }
            VWSetSpeed(vw, 0, 0);
            OSSleep(1);
            VWDriveTurn(vw, M_PI, M_PI_2);
            OSSleep(1);
            VWDriveWait(vw);
            state = STATE_LOOKING_FOR_BALL;
            break;
        }
    OSSleep(1);
    }
    VWSetSpeed(vw, 0, 0);
    VWRelease(vw);
    return 0;
}
```

```
/*
 ***************
 * imageTools.c
 ***************
 */
```

```
#include "eyebot.h"
#include "imageTools.h"
#include "system.h"
#include "eyebotError.h"


unsigned char* _lookupTable;


inline unsigned char tableLookup(unsigned short r, unsigned short g, unsigned short b)
{
    return _lookupTable[(r & 0xF8) << 7 | (g & 0xF8) << 2 | (b & 0xF8) >> 3];
}


inline unsigned char red(fullframe_t image, unsigned short x, unsigned short y)
{
    return image[(y * IMAGE_WIDTH + x) * 3];
}

inline unsigned char green(fullframe_t image, unsigned short x, unsigned short y)
{
    return image[(y * IMAGE_WIDTH + x) * 3 + 1];
}

inline unsigned char blue(fullframe_t image, unsigned short x, unsigned short y)
{
    return image[(y * IMAGE_WIDTH + x) * 3 + 2];
}


void initCam()
{
    int camera;
    int frame_rate, width, height;

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

    if (camera != 19)
        error("Camera init error");
    else

    CAMGet(&frame_rate, &width, &height);
    CAMSet(FPS1_875, 0, 0);
}

int getFullFrame(fullframe_t buffer)
{
    return CAMGetFrameRGB(buffer);
}

int getSubSampledFrame(subsampledframe_t buffer, fullframe_t original)
{
    unsigned short x, y;
    for(y = 0; y < SUBSAMPLED_IMAGE_HEIGHT; y++)
    {
        for(x = 0; x < SUBSAMPLED_IMAGE_WIDTH; x++)
        {
            unsigned short r = red(original, 2 + x * 2, 2 + y * 2);
            r += red(original, 3 + x * 2, 2 + y * 2);
            r += red(original, 2 + x * 2, 3 + y * 2);
            r += red(original, 3 + x * 2, 3 + y * 2);
            r = (r << 5) & 0x7C00;
            unsigned short g = green(original, 2 + x * 2, 2 + y * 2);
            g += green(original, 3 + x * 2, 2 + y * 2);
            g += green(original, 2 + x * 2, 3 + y * 2);
            g += green(original, 3 + x * 2, 3 + y * 2);
            g = g & 0x03E0;
            unsigned short b = blue(original, 2 + x * 2, 2 + y * 2);
            b += blue(original, 3 + x * 2, 2 + y * 2);
            b += blue(original, 2 + x * 2, 3 + y * 2);
            b += blue(original, 3 + x * 2, 3 + y * 2);
            b = (b & 0x03E0) >> 5;
            buffer[(y * SUBSAMPLED_IMAGE_WIDTH + x) * 2]     = ((r | g | b) & 0xFF00) >> 8;
            buffer[(y * SUBSAMPLED_IMAGE_WIDTH + x) * 2 + 1] = (r | g | b) & 0x00FF;
        }
```

```
    }
    return 0;
}

int getInterpretedFrame(interpretedframe_t interpreted, fullframe_t original)
{
    short x, y;
    unsigned char buffer[3][3];
    for(y = 0; y < INTERPRETED_IMAGE_HEIGHT; y++)
    {
        int current = 1;
        buffer[0][0] = tableLookup(
                            red(original, 0, y * 2),
                            green(original, 0, y * 2),
                            blue(original, 0, y * 2));
        buffer[0][1] = tableLookup(
                            red(original, 0, y * 2 + 1),
                            green(original, 0, y * 2 + 1),
                            blue(original, 0, y * 2 + 1));
        buffer[0][2] = tableLookup(
                            red(original, 0, y * 2 + 2),
                            green(original, 0, y * 2 + 2),
                            blue(original, 0, y * 2 + 2));
        for(x = 0; x < INTERPRETED_IMAGE_WIDTH; x++)
        {
            buffer[current][0] = tableLookup(
                                    red(original, x * 2 + 1, y * 2),
                                    green(original, x * 2 + 1, y * 2),
                                    blue(original, x * 2 + 1, y * 2));
            buffer[current][1] = tableLookup(
                                    red(original, x * 2 + 1, y * 2 + 1),
                                    green(original, x * 2 + 1, y * 2 + 1),
                                    blue(original, x * 2 + 1, y * 2 + 1));
            buffer[current][2] = tableLookup(
                                    red(original, x * 2 + 1, y * 2 + 2),
                                    green(original, x * 2 + 1, y * 2 + 2),
                                    blue(original, x * 2 + 1, y * 2 + 2));
            current += 1; if(current >= 3) current = 0;
            buffer[current][0] = tableLookup(
                                    red(original, x * 2 + 2, y * 2),
                                    green(original, x * 2 + 2, y * 2),
                                    blue(original, x * 2 + 2, y * 2));
            buffer[current][1] = tableLookup(
                                    red(original, x * 2 + 2, y * 2 + 1),
                                    green(original, x * 2 + 2, y * 2 + 1),
                                    blue(original, x * 2 + 2, y * 2 + 1));
            buffer[current][2] = tableLookup(
                                    red(original, x * 2 + 2, y * 2 + 2),
                                    green(original, x * 2 + 2, y * 2 + 2),
                                    blue(original, x * 2 + 2, y * 2 + 2));
            current += 1; if(current >= 3) current = 0;

            int numBall = 0, numBlue = 0, numYellow = 0;
            int i, j;
            for(i = 0; i < 3; i++) {
                for(j = 0; j < 3; j++) {
                    if(buffer[i][j] == BALL_COLOR) {
                        numBall++;
                    }
                    if(buffer[i][j] == BLUE_COLOR) {
                        numBlue++;
                    }
                    if(buffer[i][j] == YELL_COLOR) {
                        numYellow++;
                    }
                }
            }

            if(numBall >= 4) {
                interpreted[y * INTERPRETED_IMAGE_WIDTH + x] = BALL_COLOR;
            } else if (numBlue >= 6) {
                interpreted[y * INTERPRETED_IMAGE_WIDTH + x] = BLUE_COLOR;
            } else if (numYellow >= 6) {
                interpreted[y * INTERPRETED_IMAGE_WIDTH + x] = YELL_COLOR;
            } else {
                interpreted[y * INTERPRETED_IMAGE_WIDTH + x] = 15;
            }

        }
```

```
    }
    return 0;
}

int findObjects(
            interpretedframe_t buffer,
            int* ballX,
            int* ballY,
            int* ballSize,
            int* blueX,
            int* blueY,
            int* blueSize,
            int* yellowX,
            int* yellowY,
            int* yellowSize)
{
    *ballX = *ballY = *ballSize = *blueX = *blueY = *blueSize = *yellowX = *yellowY = *yellowSize = -1;
    int baX = -1, baY = -1, baSize = -1, blX = -1, blY = -1, blSize = -1, yeX = -1, yeY = -1, yeSize = -1;
    int startBaX = 0, endBaX = 0, startBaY = 0, endBaY = 0;
    int startBlX = 0, endBlX = 0, startBlY = 0, endBlY = 0;
    int startYeX = 0, endYeX = 0, startYeY = 0, endYeY = 0;
    int x, y;
    for (y = 4; y < INTERPRETED_IMAGE_HEIGHT - 3; y++)
    {
        for (x = 0; x < INTERPRETED_IMAGE_WIDTH; x++)
        {
            if(buffer[y * INTERPRETED_IMAGE_WIDTH + x] == BALL_COLOR)
            {
                if (baSize == -1)
                {
                    startBaX = x; endBaX = x; startBaY = y; endBaY = y;
                    baSize = 1;
                    baX = x;
                    baY = y;
                    *ballSize = baSize;
                    *ballX = baX;
                    *ballY = baY;
                }
                else
                {
                    if (
                        ((ABS(x - startBaX) < 3) || (ABS(x - endBaX) < 3)) &&
                        ((ABS(y - startBaY) < 2) || (ABS(y - endBaY) < 2)))
                    {
                        startBaX = MIN(x, startBaX);
                        endBaX = MAX(x, endBaX);
                        startBaY = MIN(y, startBaY);
                        endBaY = MAX(y, endBaY);
                        *ballSize = MAX(endBaX - startBaX, endBaY - startBaY);
                        *ballX = (startBaX + endBaX) / 2;
                        *ballY = (startBaY + endBaY) / 2;
                    }
                    else
                    {
                        startBaX = x; endBaX = x; startBaY = y; endBaY = y;
                        if(baSize > *ballSize)
                        {
                            *ballX = baX;
                            *ballY = baY;
                            *ballSize = baSize;
                        }
                    }
                }
            }
            else if(buffer[y * INTERPRETED_IMAGE_WIDTH + x] == BLUE_COLOR)
            {
                if(y < 40)
                {
                    if (blSize == -1)
                    {
                        startBlX = x; endBlX = x; startBlY = y; endBlY = y;
                        blSize = 1;
                        blX = x;
                        blY = y;
                        *blueSize = blSize;
                        *blueX = blX;
                        *blueY = blY;
                    }
                    else
```

```
                    {
                        if (
                            ((ABS(x - startBlX) < 6) || (ABS(x - endBlX) < 6)) &&
                            ((ABS(y - startBlY) < 3) || (ABS(y - endBlY) < 3)))
                        {
                            startBlX = MIN(x, startBlX);
                            endBlX = MAX(x, endBlX);
                            startBlY = MIN(y, startBlY);
                            endBlY = MAX(y, endBlY);
                            blSize = MAX(endBlX - startBlX, endBlY - startBlY);
                            blX = (startBlX + endBlX) / 2;
                            blY = endBlY;
                        }
                        else
                        {
                            startBlX = x; endBlX = x; startBlY = y; endBlY = y;
                            if(blSize > *blueSize)
                            {
                                *blueX = blX;
                                *blueY = blY;
                                *blueSize = blSize;
                            }
                        }
                    }
                }
            }
            else if(buffer[y * INTERPRETED_IMAGE_WIDTH + x] == YELL_COLOR)
            {
                if(y < 40)
                {
                    if (yeSize == -1)
                    {
                        startYeX = x; endYeX = x; startYeY = y; endYeY = y;
                        yeSize = 1;
                        yeX = x;
                        yeY = y;
                        *yellowSize = yeSize;
                        *yellowX = yeX;
                        *yellowY = yeY;
                    }
                    else
                    {
                        if (
                            ((ABS(x - startYeX) < 6) || (ABS(x - endYeX) < 6)) &&
                            ((ABS(y - startYeY) < 3) || (ABS(y - endYeY) < 3)))
                        {
                            startYeX = MIN(x, startYeX);
                            endYeX = MAX(x, endYeX);
                            startYeY = MIN(y, startYeY);
                            endYeY = MAX(y, endYeY);
                            yeSize = MAX(endYeX - startYeX, endYeY - startYeY);
                            yeX = (startYeX + endYeX) / 2;
                            yeY = endYeY;
                        }
                        else
                        {
                            startYeX = x; endYeX = x; startYeY = y; endYeY = y;
                            if(yeSize > *yellowSize)
                            {
                                *yellowX = yeX;
                                *yellowY = yeY;
                                *yellowSize = yeSize;
                            }
                        }
                    }
                }
            }
        }
    }
    return 0;
}

void setLookupTable(lookuptable_t table)
{
    _lookupTable = table;
}
```

```
/*
 ***************
 * commTools.c
 ***************
 */

#include "eyebot.h"
#include "commTools.h"
#include "system.h"
#include "eyebotError.h"



void send(unsigned char* checksum, unsigned char data)
{
  *checksum += data;
  OSSendCharRS232(data, SERIAL_PORT);
}

void sendSOF(unsigned char* checksum)
{
    send(checksum, 0x73);
    send(checksum, 0x78);
    send(checksum, 0x75);
}

void sendEOF(unsigned char* checksum)
{
    send(checksum, 0x76);
    send(checksum, 0x74);
    if(*checksum <= 0x7F && *checksum >= 0x70)
    {
        send(checksum, 0x70);
    }
    OSSendCharRS232(*checksum, SERIAL_PORT);
}

void sendPayload(unsigned char* checksum, unsigned char image[], int size)
{
    int i;
    for(i = 0; i < size; i++)
    {
        if(image[i] <= 0x7F && image[i] >= 0x70)
        {
            send(checksum, 0x70);
        }
        send(checksum, image[i]);
    }
}

void initComms()
{
    int err = OSInitRS232(SER115200, RTSCTS, SERIAL_PORT);
    if(err == 8)
    {
        error("Illegal baud");
    }
    else if(err == 10)
    {
        error("Illegal port");
    }
}

int sendFullFrame(fullframe_t image)
{
    unsigned char checksum = 0;
    sendSOF(&checksum);

    send(&checksum, (char) IMAGE_WIDTH);
    send(&checksum, (char) IMAGE_HEIGHT);
    send(&checksum, (char) 24);

    int size = IMAGE_SIZE;
    sendPayload(&checksum, image, size);

    sendEOF(&checksum);
    return 0;
```

```
}

int sendSubSampledFrame(subsampledframe_t image)
{
    unsigned char checksum = 0;
    sendSOF(&checksum);

    send(&checksum, (char) SUBSAMPLED_IMAGE_WIDTH);
    send(&checksum, (char) SUBSAMPLED_IMAGE_HEIGHT);
    send(&checksum, (char) 16);

    int size = SUBSAMPLED_IMAGE_SIZE;
    sendPayload(&checksum, image, size);

    sendEOF(&checksum);
    return 0;
}

int sendInterpretedFrame(interpretedframe_t image)
{
    unsigned char checksum = 0;
    sendSOF(&checksum);

    send(&checksum, (char) INTERPRETED_IMAGE_WIDTH);
    send(&checksum, (char) INTERPRETED_IMAGE_HEIGHT);
    send(&checksum, (char) 4);

    int size = INTERPRETED_IMAGE_SIZE;
    sendPayload(&checksum, image, size);

    sendEOF(&checksum);
    return 0;
}



void flushRx()
{
    int i = OSCheckInRS232(SERIAL_PORT);
    while(i > 0)
    {
        OSFlushInRS232(SERIAL_PORT);
        OSSleep(10);
        i = OSCheckInRS232(SERIAL_PORT);
        //LCDPrintf("Chars left");
    }
    if(i < 0)
    {
        LCDPrintf("CheckIn error\n");
    }
}

short receive(unsigned char* ch)
{
    short ret = OSRecvRS232(ch, SERIAL_PORT);
    if(ret)
    {
        LCDPrintf("uart err: %d", ret);
        flushRx();
    }
    return ret;
}


int recvLookupTable(lookuptable_t table)
{
    int current = 0;
    unsigned char seq = 0;
    unsigned char checksum = 0, escaped = 0;
    unsigned char ch;
    while(seq < 8)
    {
        int ret = receive(&ch);
        if(ret)
        {
            LCDPrintf("error");
            OSSleep(100);
            return seq;
        }
```

```
switch(seq)
{
    case 0:
        checksum += ch;
        if(ch == 0x73)
        {
            seq = 1;
        }
        else
        {
            seq = 8;
        }
        break;
    case 1:
        checksum += ch;
        if(ch == 0x7A)
        {
            seq = 2;
        }
        else
        {
            seq = 8;
        }
        break;
    case 2:
        checksum += ch;
        if(ch == 0x75)
        {
            seq = 3;
        }
        else
        {
            seq = 8;
        }
        break;
    case 3:
        checksum += ch;
        if(escaped)
        {
            escaped = 0;
            table[current++] = ch;
        }
        else if(ch == 0x70)
        {
            escaped = 1;
        }
        else if((ch <= 0x7F) && (ch >= 0x71))
        {
            seq = 9;
        }
        else
        {
            table[current++] = ch;
        }
        if(current >= LOOKUP_TABLE_SIZE)
        {
            seq = 4;
        }
        if((current & 0x1FF) == 0)
        {
            LCDSetPos(3, 0);
            LCDPrintf("rx: %d     ", current);
        }
        break;
    case 4:
        checksum += ch;
        if(ch == 0x76)
        {
            seq = 5;
        }
        else
        {
            seq = 10;
        }
        break;
    case 5:
        checksum += ch;
        if(ch == 0x74)
```

```
                {
                    seq = 6;
                }
                else
                {
                    seq = 11;
                }
                break;
            case 6:
                if(escaped)
                {
                    seq = 13;
                }
                else if(ch == 0x70)
                {
                    escaped = 1;
                }
                else if((ch <= 0x7F) && (ch >= 0x71))
                {
                    seq = 12;
                }
                else
                {
                    seq = 13;
                }
                break;
        }
    }
    switch(seq)
    {
        case 8:
            LCDPrintf("Format error, %d", ch);
            flushRx();
            return -1;
            break;
        case 9:
            LCDPrintf("Escape error, %d", ch);
            flushRx();
            return -2;
            break;
        case 10:
            LCDPrintf("Escape error, %d", ch);
            flushRx();
            return -3;
            break;
        case 11:
            LCDPrintf("Escape error, %d", ch);
            flushRx();
            return -4;
            break;
        case 12:
            LCDPrintf("Escape error, %d", ch);
            flushRx();
            return -5;
            break;
        case 13:
            if(checksum == ch)
            {
                return 0;
            }
            else
            {
                LCDPrintf("Checksum error");
                flushRx();
                return -4;
            }
            break;
    }
    return 0;
}
```