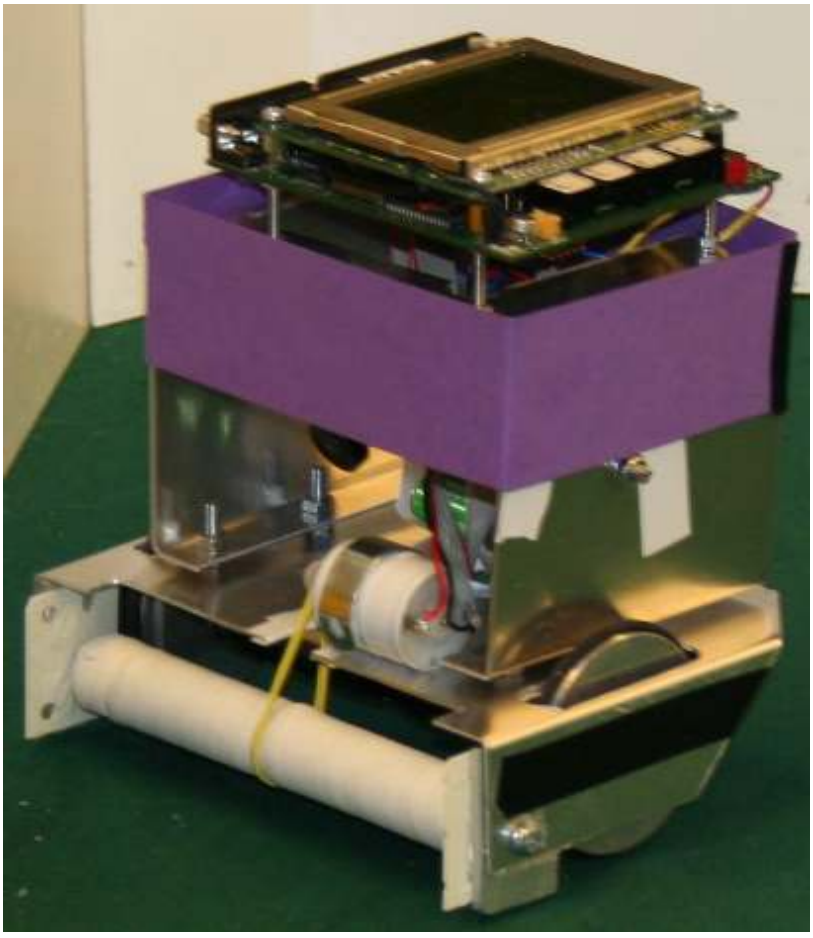
RAS3 Project Report

“Nummer 1”



Robotics and Autonomous Systems
June 2007

John Lindberg
Daniel Banksell
Björn Ingeson
Jonas Feldt

Abstract

The football playing robot “Nummer 1” was built as a part of the course Robotics and Autonomous Systems at KTH. The robot was built and programmed having the phrase “quick and dirty” in mind, an idea that originally surfaced because of some initial hardware problems. By avoiding such time-consuming tasks as positioning and the processor heavy VW-drive, the robot got really fast at scoring goals. Alas, some serious camera tuning problems made the robot unable to score in the competition. The general concept though was a success.

Table of contents

Introduction

We decided to keep the design of the robot as simple as possible. Only the most necessary parts were added. No whiskers were added because of the uncertainty and unwanted behaviour seen on the robots from previous years. No servo was used either, as it only complicated things. Because of the initial problems with the encoders on the drive motors, we decided not to use vw-drive. We also decided not to use positioning because of the same reason. Instead we chose to rely heavily on camera vision for finding out what to do next. Our advantage over the opponents was supposed to be the agility of the robot; seeing the ball first, getting to it first and then to score without the opponent even realising what happened.

Construction

To build the robot we got:

- 2 sheets of aluminium plate
- EyeCon Eyebot controller (it has a 35MHz 32-bit Motorola 68332 microcontroller, 2MB RAM and 512KB flash ROM)
- Digital mini camera (176x144pixels)
- 2 Faulhaber micro motors with gearbox and vw-drive
- 2 aluminium wheels.
- Small electrical servo
- 2 battery packages

We also added some parts of our own:

- A 5V electric motor for driving the roller
- 2 ball bearings for the roller
- An NPN transistor used to build the roller controller
- A diode and some resistors

Base

When building our robot we wanted to make sure it worked properly and that it could take a punch or two from other robots. We also decided to go with an as easy approach as possible due to the fact that our time was limited and our processor was quite slow. We decided to build an easy rectangular robot with the wheels on the inside to protect them and tried to make it as wide as possible without losing balance to make more space on which the ball could hit the robot. We used a differential drive technique as it was easy to build and implement in software. So we built a base like an upside down box under which we put the motors and just inside its edges we made holes for the wheels.

Roller

To be able to dribble with the ball we constructed a roller, a cylinder at the front of the base which was spinning so that the ball when touching it would spin with it in a way which made it roll against the robot. We tested several materials for the roller, it couldn't be to hard cause then the ball would bounce to much either roll away or just don't get enough speed towards the robot. It also had to be somewhat sticky so that it would have good friction against the ball. After testing several materials we came to the conclusion that a rubber stem clad with a

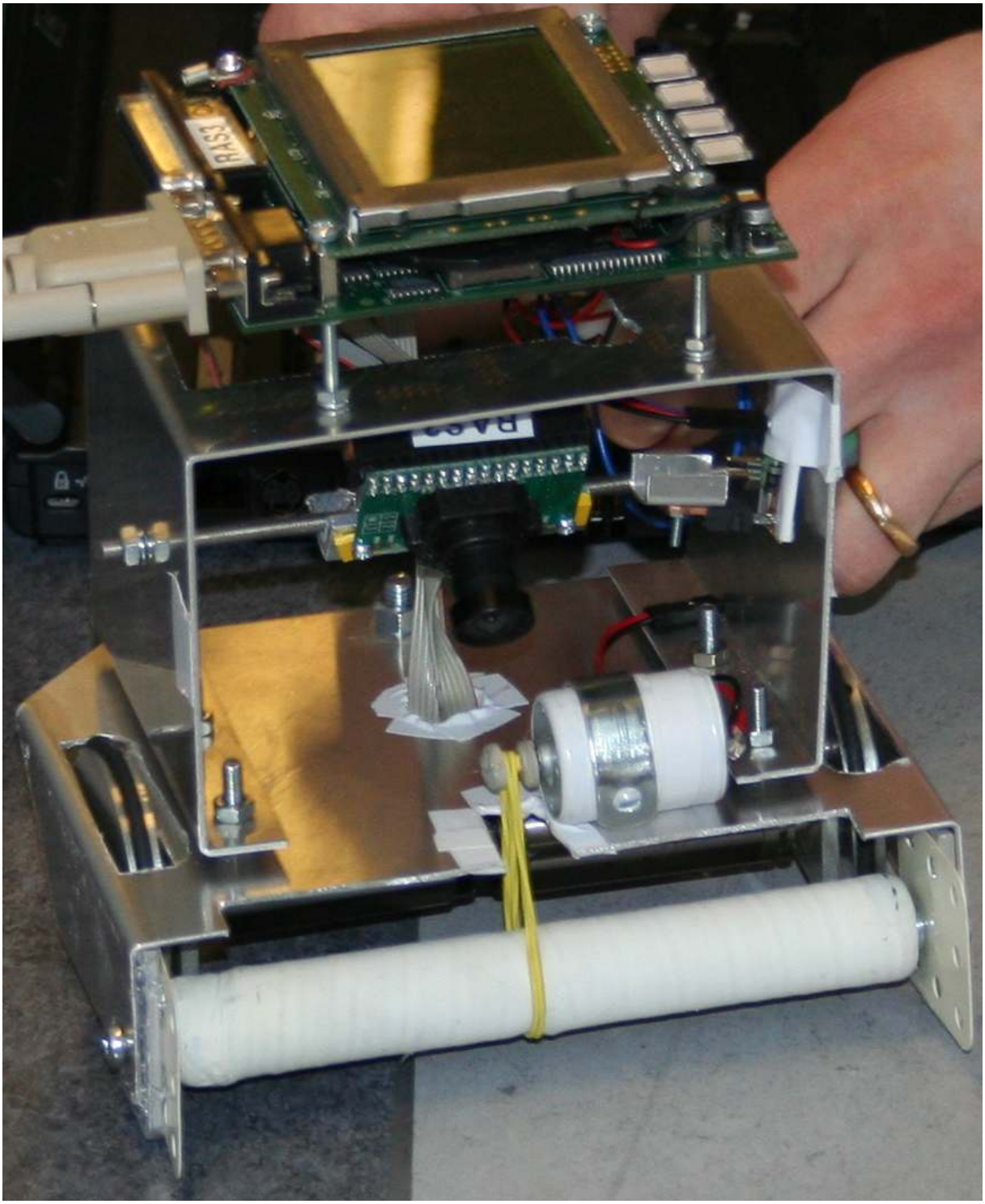
balloon was the best option and it proved very good in the test runs. The roller was driven by a micro motor with a rubber band as a belt.

Roller-controller

The roller-speed is controlled via a circuit which uses a servo-signal pulse to determine for how long the roller-motor will be active. In theory we can set the roller-speed from zero to hundred percent since the only thing we change is the length of the servo pulse. In our final version this solution is used to be able to turn on and off the roller to conserve battery-power.

Electronics tower

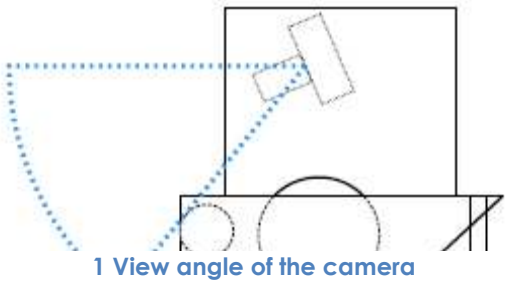
For the camera and Eyebot-computer we build a metal box with only two sides. We mounted the camera inside this box, protecting it from other robots, and the Eyebot-computer at the top. The whole construction gave us a low point of gravity and made the robot very stable. The box was put upon the base and inside we also put the motor for the roller and the batteries for the robot which were held down by a rubber band.
One major concern with the hardware was the batteries which didn't work properly due to their holders, after changing them several times and modifying them we got them to work almost all the time but it wasn't very fun to come back to the room after have left the batteries on charge for several hours just to find out that it hadn't connected and was uncharged.



A close up of the robot showing the construction details.

Camera

Eyecam-3 can return pictures of size 176x144 pixels with a color-depth of 24 bits. The camera is positioned in the middle of the robot with a downward angle. This makes it possible for the camera to see the ball if it is situated directly in front of the roller, while the upper



1 View angle of the camera

border of the picture is low enough to exclude everything outside the soccer-field since we don't want any clothes or other colorful items to be able to disturb our classifier.
The camera returns three values for each pixel in the picture, each of them are eight bits long. These values represent red, green and blue color (RGB).
There is an auto brightness-function which adjusts the brightness in the picture. This function depends on the light-condition when the picture is taken and has the side-effect that a series of pictures can differ even though the robot is standing still. We cannot turn off this feature since some hardware-changes are needed and the problem will have to be taken in consideration in future calculations.

Image processing

When the main-program is executed it starts an initiation-phase. In this phase the program generates a lookup-table for all different classes. This table is used to speed up the image processing since no conversions or calculations will be executed when we classify different pixels. If we wanted to store the entire table for all possible combinations of the incoming RGB-values we had to store $(2^8)^3 = 16777216$ bits. This is impossible since we are limited to a 2 Mb storage memory. Instead of storing all eight bits in all three values we choose to store the five most significant which leaves us with a table of size $(2^5)^3 = 32768$ bits. We will have a marginal loss in precision when we round the three least significant bits since the RGB-values provided aren't that stable. When the lookup-table is initialized we calculate hue, saturation, intensity and the value for all RGB-values since we use that color-space when we classify pixels in a picture. The table contains four different classes:

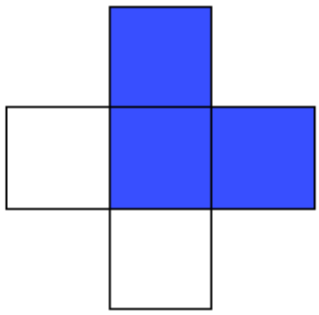
- Ball = 1
- Blue goal = 2
- Yellow goal = 3
- Opponent = 4
- None = 0

The classification-algorithm is based on color-segmentation where we search for the four classes mentioned above. All classes are defined by different intervals in the RGB and HSIV spaces. To classify a pixel correct, all requirements in form of values have to be met.
The robot searches for all classes in every picture that is taken. Since it takes a lot of time to classify every pixel in one picture (176x144 = 25344 pixels) we want to exclude parts of the picture where certain classes never will appear. When searching for the ball we process all pixels in the picture, but when we are looking for a goal we only need to examine the upper part of the image. This way we save a lot of processing-time.

2 Pictures from the robot

All pixels in the limited area of the picture are processed in search of correct class-pixels. Each pixels RGB-values are sent to the lookup-table to retrieve its class. Since the camera has

a limited resolution and color-reproduction, noise can occur. Each class has a simple and fast filtering to reduce noise in form of single pixels.



3 Simple noise-reduction

For every processed pixel its neighboring pixels are examined and classified in the same way. For the ball-class we are satisfied if one neighboring pixel matches the one being processed. But for the goals, which often cover larger areas of the picture, we want more than two neighboring pixels to be of the same class.
For each class the upper-left corner, height and width of the found pixel-cluster are returned. The amount of pixels of each class is also returned and is used to determine if we have found a class or noise. The center-point for each class is calculated and used to determine what the robot should do next.

Programming and AI

We decided to use a very shallow state machine in Nummer 1's AI. Which state it is in is decided only by what the camera is seeing at a given moment, and what the previous state was.

Motor Control

Because we from the beginning didn't have any working encoders in the motors, we started to implement a motor controller which would not need them. We simply decided on a set of movements, which we then created using different motor speeds and different times they should be given.

When we later on got working encoders we decided to stick to the original plan, anything else should have made most of the previous work unnecessary. Using encoder though, we could improve some movement using the measurements from them. The big problem about this solution was that the robot got really sensitive on different forces acting on it. A heavier load, weaker battery or a ball spinning against the roller made a lot of difference on the robot's movement. A PID controller might have been able to resolve this problem, but we really didn't think it would be necessary. In hindsight we should have made a PID a higher priority.

Using the camera

During a match the program alternates between two functions; first it examines the camera image to see what state it is in and if any interesting objects can be seen. Lastly it makes a decision based on that information.

In most cases the robot is standing still when it examines the camera image, because it gives the pictures a higher quality and also because the computations takes a little bit too much time for the old instructions to be valid during the duration of the execution. When the robot sees and goes towards a certain object, we let it continue forwards because it is almost always just the angle that needs to be corrected a bit between the updates.

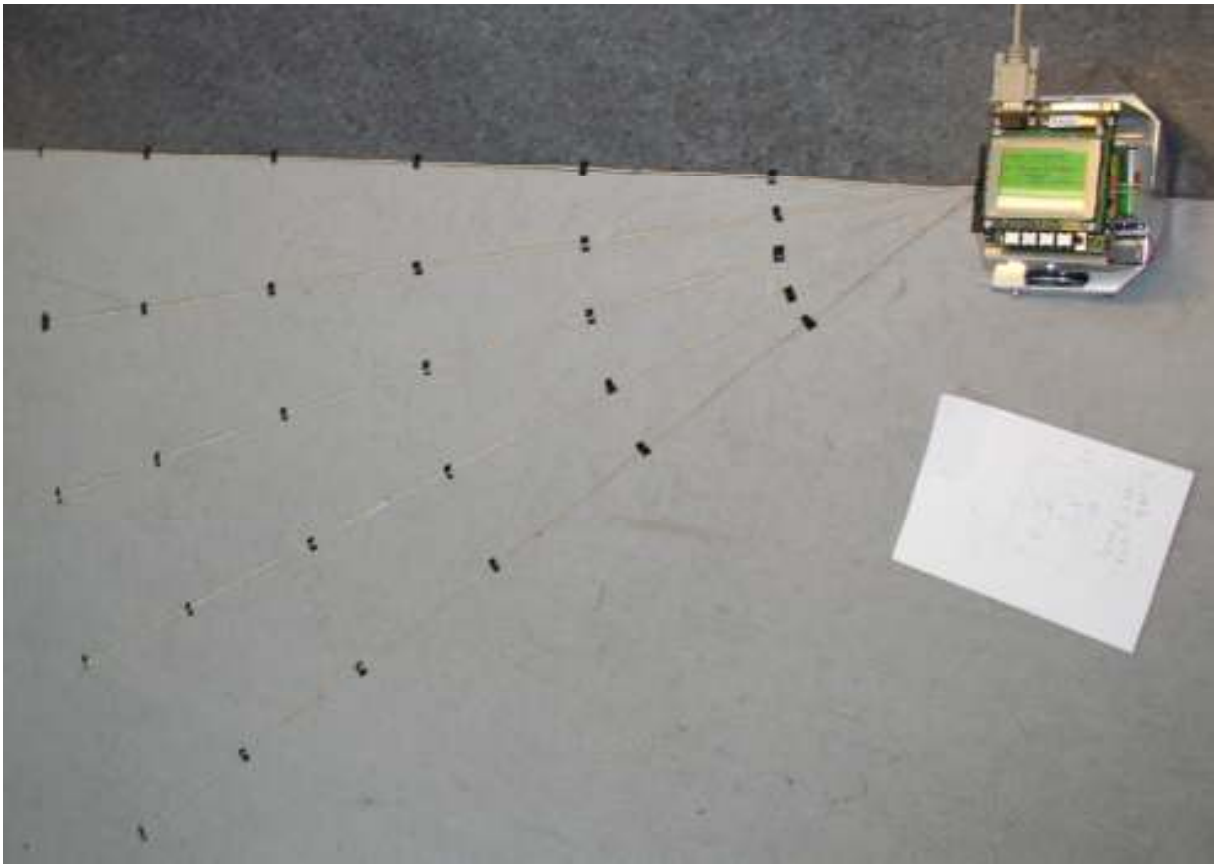
State machine

When the match starts, the robot quickly drives forward about 30 cm to come closer to the ball. If it does not see the ball it quickly rotates 45° and tries again. These rotations are made very quickly because of the stopping mechanism, which gives the engines a reverse thrust for a small period of time. We thought of introducing functionality to make the robot being able to go away at a random direction if it had rotated a whole 360° without finding the ball, but the time wasn't enough and it almost always saw the ball. When the robot has spotted the ball it drives straight towards it and is continuously adjusting it's angle to make the ball get right in front of it. When it has come close enough, it quickly stops, not to push the ball away.

After Nummer 1 has caught the ball it rotates with the ball, which is hold in place by the counter spin of the ball created by the roller. When it sees the opponents goal it drives towards it, now again continuously adjusting to reach the centre of the goal.

If the robot loses the ball, Nummer 1 continues on it's previous path a small period of time as nothing has happened, in case it was just a glitch in the camera not recognising the ball. If the ball still is lost, the robot returns to its original behaviour of finding the ball.

Regardless of what state the robot is in it examines the odometers to see if it hadn't been able to move, and we then assumes it is stuck against a wall. To move away, it backs the wheel at the side which should not be facing the wall. This almost always clears it from the wall. If the robot is stuck two times in five seconds, we assume that it has been caught in a loop, and lets it rotate quickly and drive away a bit not to get stuck again. It then tries to find the ball again. We thought of trying to push hard forward a single time, trying to push the opponent robot backwards if it was that that made the robot get stuck, but that was never implemented.



The method used for initializing the distance measurement table.

Conclusion

Building and programming the Nummer 1-robot was really satisfying (and sometimes frustrating) for everyone involved. We have learnt a lot about robotics and the difficulties constructing even a simple robot like this. The construction and programming would have been a lot easier if we had known a couple of things in advance:

- The best material for giving the roller friction against the ball is balloon rubber.
- The shape of the robot should be as circular as possible, making it easy to steer away from almost any obstacle.
- When building a roller controller, you cannot use a low current transistor. The transistor must be able to handle close to 1A, otherwise the performance will be unstable.
- Be very careful when using the serial cable extension cord. People *will* trip over it, which will lead to a disaster.
- Be sure to be lucky when being assigned robot parts. The shiny new motors will give you a headache when it appears the encoders are not working.
- If not using vw-drive, implement a PID controller. This makes dribbling using a roller easier to program.
- Mount the camera high enough to be able to see the purple marker of the opposition.

The idea of quick and dirty construction was first “forced” on us, but it later became obvious that the initial hardware problems weren’t as big as a handicap as we thought it would be. Some things could of course have been done differently, but the many successful trial runs at the football field showed us the correctness of the general idea behind the construction and programming.

```c
#include <math.h>
#include "globals.h"
#include "action.h"

void run_camera(void);
void run_action(void);

/* Global variables: */
MotorHandle mr, ml;
QuadHandle qr, ql;
Worldview wv, oldwv;
Goal target_goal;
int has_seen_ball;
ServoHandle roller;

/* Local functions: */
int main_menu(void);
void print_goal(void);
void print_state(void);

int main ()
{
  /* Variable initiation: */
  target_goal = GOAL_BLUE;

  /* Camera initiation: */
  RAS3_InitAll();

  /* Motor Initiation: */
  mr = MOTORInit(MOTOR_RIGHT);
  ml = MOTORInit(MOTOR_LEFT);
  stop();

  /* Quad Initiation: */
  qr = QUADInit(QUAD_RIGHT);
  ql = QUADInit(QUAD_LEFT);

  /* Roller Initiation */
  roller = SERVOInit(SERVO8);
  SERVOSet(roller,0);

  while (TRUE) {
    LCDClear();
    int retval = main_menu();
    if (retval != 0) break;
    LCDClear();
    LCDMenu(" ", " ", " ", "MENU");

    wv.state = START;
    wv.goalw = 0;
    wv.myw = 0;
    has_seen_ball = 0;
    oldwv = wv;

    while (KEYRead() != KEY4) {
      run_action();
      run_camera();
```

```
      LCDSetPos(2, 0);
      LCDPrintf("x:%.2f,y:%.2f\n", wv.ballx, wv.bally);
      LCDSetPos(6, 0);
      LCDPrintf("b:%d,g:%d\n", wv.ballw, wv.goalw);
      //if (wv.state != oldwv.state)
        print_state();
    }
    roller_stop();
    stop();
  }

  /* Release all resources: */
  CAMRelease();
  MOTORRelease(mr | ml);
  QUADRelease(qr | ql);
  SERVORelease(roller);
  return 0;
}

int main_menu(void) {
  LCDMenu("GOAL","CHG","RUN","QUIT");
  print_goal();
  while (TRUE) {
    int key = KEYGet();
    switch (key) {
    case KEY1:
      target_goal = 1-target_goal;
      print_goal();
      break;
    case KEY2: break;
    case KEY3: return 0; break;
    case KEY4: return 1; break;
    }
  }
}

void print_goal(void) {
  if (target_goal == GOAL_BLUE) LCDPrintf("Blue");
  else LCDPrintf("Yellow");
  LCDPrintf(" is goal.\n");
}

void print_state(void) {
  switch (wv.state) {
  case FIND_BALL:
    LCDSetString(0, 0, "FIND_BALL");
    break;
  case SCORE:
    LCDSetString(0, 0, "SCORE");
    break;
  case FIND_GOAL:
    LCDSetString(0, 0, "FIND_GOAL");
    break;
  case GET_BALL:
    LCDSetString(0, 0, "GET_BALL");
    break;
  case HAS_SCORED:
```

```
    LCDSetString(0, 0, "HAS_SCORED");
    break;
  case START:
    LCDSetString(0, 0, "START");
    break;
  default:
    LCDSetString(0, 0, "UNKNOWN_STATE");
  }
}

void change_angle(int *deg, int diff) {
  *deg += diff;
  if (*deg > 180) *deg -= 360;
  else if (*deg <= -180) *deg += 360;
}
```

```
#ifndef __GLOBALS_H
#define __GLOBALS_H

#include "eyebot.h"

#define RAD2DEG (57.29577951)

typedef enum {START = 0, GET_BALL, FIND_GOAL, SCORE, SCORE_YELLOW, FIND_BALL, HAS_SCORED,
SPIN_WAIT} State;

typedef int degrees;

typedef struct {
  /* Relative positions: x = right, y = forward, w = ccw from y-axis */
  meter ballx; //relativ bollposition
  meter bally; //relativ bollposition
  degrees ballw;
  meter oppx;  //relativ motståndarposition
  meter oppy;  //relativ motståndarposition
  meter goalx;
  meter goaly;
  degrees goalw; //relativ vinkel till målet
  meter wally; //avstånd till närmaste vägg frammåt
  degrees myw;
  meter odor,odol;

  int timestamp; //set to OSGetCount() after you update.

  State state;
} Worldview;
extern Worldview oldwv, wv; //copy to oldwv before you update.

typedef enum {GOAL_BLUE = 0, GOAL_YELLOW} Goal;
extern Goal target_goal;

extern MotorHandle mr, ml;

extern QuadHandle qr, ql;

extern ServoHandle roller;

extern int has_seen_ball;

void change_angle(int *deg, int diff) ;

#endif //__GLOBALS_H
```

action.h

```
#ifndef __ACTION_H
#define __ACTION_H

void update_drive(void);
void rotate(int, int);
void fast_stop(void);
int drive_straight(float, int);
int backup(float, int);
void fast_rotate(int dir, float distgoal, int);
void roller_run();
void roller_stop();
inline void rot180l(void);
inline void rot180r(void);
inline void rot90l(void);
inline void rot90r(void);
inline void rot45l(void);
inline void rot45r(void);
void stop(void);

#define LEFT 1
#define RIGHT -1

#define ROLLER_SPEED 255
#define REGULAR_SPEED 24
#define ROLLERING_SPEED 22
#define BALL_SPEED 26
#define FIND_GOAL_ROTATION 9
#define FIND_GOAL_SPEED 12
#define MIN_ROTATION 10
#define MAX_ROTATION 20
#define ROTATION_STEP 100
#define LOST_DELAY 50 // s/100
#define STOP_DELAY 50
#define FIX_DELAY 500

#endif
```

```c
#include <math.h>
#include "globals.h"
#include "action.h"

int v = 0, w = 0;
float wf = 0;

int lost_time;
meter lost_x;

int stop_time;
int fix_time = 0;

int rollering = 0;

const float ROTATION_DIST_LEFT_180 = 0.160;
const float ROTATION_DIST_RIGHT_180 = 0.190;
const float ROTATION_DIST_LEFT_90 = 0.088;
const float ROTATION_DIST_RIGHT_90 = 0.108;
const float ROTATION_DIST_LEFT_45 = 0.042;
const float ROTATION_DIST_RIGHT_45 = 0.053;

void run_action(void) {
  LCDSetPos(4, 0);
  LCDPrintf("v: %d, w: %d\n", v, w);
  int dt = wv.timestamp - oldwv.timestamp;
  wv.odol = QUADODORead(ql);
  wv.odor = QUADODORead(qr);
  int iiii, freq = 440;

  switch (wv.state) {

  case START:
    drive_straight(0.3, 500);
    break;

  case GET_BALL:
    if (wv.ballx < -0.05) {
      fast_rotate(LEFT, (ROTATION_DIST_LEFT_45*(-wv.ballw))/90.0, 0);
      change_angle(&(wv.myw), wv.ballw);
    }
    else if (wv.ballx > 0.05) {
      fast_rotate(RIGHT, (ROTATION_DIST_RIGHT_45*(wv.ballw))/90.0, 0);
      change_angle(&(wv.myw), wv.ballw);
    }
    if (!has_seen_ball) {
      has_seen_ball = 1;
    }
    if (wv.bally <= 0.21) {
      roller_run();
      if (oldwv.bally > 0.21) {
        fast_stop();
      }
      v = ROLLERING_SPEED;
    }
    else {
      roller_stop();
```

```
      v = REGULAR_SPEED;
    }
    break;

  case SCORE:
    roller_run();
    if (wv.goalx < 0.05) {
      fast_rotate(LEFT, (ROTATION_DIST_LEFT_45*(-wv.goalw))/120.0, 0);
      change_angle(&(wv.myw), wv.goalw);
    }
    else if (wv.goalx > 0.05) {
      fast_rotate(RIGHT, (ROTATION_DIST_RIGHT_45*(wv.goalw))/120.0, 0);
      change_angle(&(wv.myw), wv.goalw);
    }
    v = BALL_SPEED;
    break;

  case SCORE_YELLOW:
    AUTone(440, 100);
    OSWait(5);
    AUTone(784, 100);
    OSWait(5);
    drive_straight(0.05, 100);
    break;

  case FIND_BALL:

    switch (oldwv.state) {

    case GET_BALL:
      lost_time = wv.timestamp;
      lost_x = oldwv.ballx;
      break;

    case SCORE:
    case FIND_GOAL:
      lost_time = wv.timestamp;
      lost_x = oldwv.ballx;
      break;

    case FIND_BALL:
      if (wv.timestamp - lost_time > LOST_DELAY) {
        roller_stop();
        if (lost_x < 0)
        rot45l();
        else
        rot45r();
        stop();
      }
      break;

    default:
      lost_time = wv.timestamp;
      lost_x = oldwv.ballx;
    }
    break;
```

```
  case FIND_GOAL:
    roller_run();
    if (oldwv.state != FIND_GOAL) fast_stop();
    v = 0;
    if (wv.ballx < 0) {
      v = FIND_GOAL_SPEED;
      w = -FIND_GOAL_ROTATION;
      wv.myw -= 3;
    }
    else {
      v = FIND_GOAL_SPEED;
      w = FIND_GOAL_ROTATION;
      wv.myw += 3;
    }
    update_drive();
    OSWait(25);
    stop();
    break;

  case HAS_SCORED:
    roller_stop();

    AUTone(440, 100);
    OSWait(5);
    AUTone(494, 100);
    OSWait(5);
    AUTone(554, 100);
    OSWait(5);
    AUTone(622, 100);
    OSWait(5);
    AUTone(698, 100);
    OSWait(5);
    AUTone(784, 100);
    OSWait(5);
    backup(0.3, 100);
    fast_stop();
    OSWait(10);
    rot180l();
    drive_straight(1.3, 500);
    fast_stop();
    OSWait(10);
    rot180r();
    stop();
    break;
  }

  update_drive();

  if (wv.state != SPIN_WAIT && !(wv.odol - oldwv.odol > 0.02 || wv.odol - oldwv.odol < -0.02 ||
        wv.odor - oldwv.odor > 0.02 || wv.odor - oldwv.odor < -0.02)) {
    if (stop_time != 0) {
      if (wv.timestamp > stop_time+STOP_DELAY) {
        switch (wv.state) {
        case GET_BALL: case FIND_GOAL:
          if (wv.ballx < 0) MOTORDrive(ml, -REGULAR_SPEED);
          else MOTORDrive(mr, REGULAR_SPEED);
          OSWait(20);
```

```
          break;
        default:
          MOTORDrive(mr | ml, -REGULAR_SPEED);
          OSWait(50);
        }
        stop();
        stop_time = 0;
        if (wv.timestamp < fix_time+FIX_DELAY) {
          switch (wv.state) {
          default:
            rot45r();
            MOTORDrive(mr | ml, -REGULAR_SPEED);
            OSWait(100);
          }
          stop();
        }
        fix_time = wv.timestamp;
      }
    } else stop_time = wv.timestamp;
  }
  else stop_time = 0;
}

void roller_run() {
  SERVOSet (roller,ROLLER_SPEED);
  rollering = 1;
}

void roller_stop() {
  SERVOSet(roller,0);
  rollering = 0;
}

void fast_stop(void) {
  int ticksr = QUADRead(qr);
  int ticksl = QUADRead(ql);
  OSWait(5);
  int dirr = (ticksr < QUADRead(qr))?1:-1;
  int dirl = (ticksl < QUADRead(ql))?1:-1;

  MOTORDrive(mr, -dirr*40);
  MOTORDrive(ml, -dirl*40);
  OSWait(3);
  stop();
}

void stop(void) {
  v = 0;
  w = 0;
  wf = 0;
  MOTORDrive(mr | ml, 0);
}

void rotate(int dir, int dt) {
  if (wf*dir < MIN_ROTATION)
  wf = MIN_ROTATION*dir;
  else
```

```
  wf += dir*ROTATION_STEP/dt;
  w = (int)wf;
  if (dir*w > MAX_ROTATION) {
    w = dir*MAX_ROTATION;
    wf = w;
  }
}

void fast_rotate(int dir, float distgoal, int fast_stop) {
  MOTORDrive(mr, 30*dir);
  MOTORDrive(ml, -30*dir);

  QuadHandle *q = &qr;
  if (dir == -1) {
    q = &ql;
  }
  float diststart = QUADODORead(*q);
  float dist;
  int timeout = OSGetCount() + 100;
  while((dist = (QUADODORead(*q)-diststart)) < distgoal) {
    if (OSGetCount() > timeout) {
      v = -20;
      w = 0;
      update_drive();
      OSWait(20);
      stop();
      break;
    }
  }
  if (fast_stop) {
    MOTORDrive(mr, -60*dir);
    MOTORDrive(ml, 60*dir);
    OSWait(3);
  }
  stop();
}

inline void rot180l() {
  fast_rotate(LEFT, ROTATION_DIST_LEFT_180, 1);
  change_angle(&(wv.myw), -180);
}
inline void rot180r() {
  fast_rotate(RIGHT, ROTATION_DIST_RIGHT_180, 1);
  change_angle(&(wv.myw), 180);
}
inline void rot90l() {
  fast_rotate(LEFT, ROTATION_DIST_LEFT_90, 1);
  change_angle(&(wv.myw), -90);
}
inline void rot90r() {
  fast_rotate(RIGHT, ROTATION_DIST_RIGHT_90, 1);
  change_angle(&(wv.myw), 90);
}
inline void rot45l() {
  fast_rotate(LEFT, ROTATION_DIST_LEFT_45, 1);
  change_angle(&(wv.myw), -45);
}
```

```
inline void rot45r() {
  fast_rotate(RIGHT, ROTATION_DIST_RIGHT_45, 1);
  change_angle(&(wv.myw), 45);
}

int drive_straight(float m, int timeout) {
  MOTORDrive(mr | ml, 30);
  float diststart = QUADODORead(ql);
  float dist;
  timeout += OSGetCount();
  while((dist = (QUADODORead(ql)-diststart)) < m) {
    if (OSGetCount() > timeout) return 0;
  }
  MOTORDrive(mr | ml, -30);
  OSWait(3);
  stop();
  return 1;
}

int backup(float m, int timeout) {
  MOTORDrive(mr | ml, -30);
  float diststart = QUADODORead(ql);
  float dist;
  timeout += OSGetCount();
  while((dist = (QUADODORead(ql)-diststart)) > -m) {
    if (OSGetCount() > timeout) return 0;
  }
  MOTORDrive(mr | ml, 30);
  OSWait(3);
  stop();
  return 1;
}

void update_drive() {
  int vr = v+w;
  int vl = v-w;
  if (vr > 100) {
    vl -= vr-100;
    vr = 100;
  }
  if (vl > 100) {
    vr -= vl-100;
    vl = 100;
  }
  if (vr < -100) {
    vl -= vr+100;
    vr = -100;
  }
  if (vl < -100) {
    vr -= vl+100;
    vl = -100;
  }
  if (vr > 100) vr = 100;
  if (vl > 100) vl = 100;
  MOTORDrive(mr, vr);
  MOTORDrive(ml, vl);
}
```

```
#include "globals.h"
#include "eyelib.h"
#include <math.h>

BYTE ColPic[HEIGHT][WIDTH][3];

const double TAN_50 = 1.191753593;

void run_camera(void) {
  int x1,x2,x3,x4,y1,y2,y3,y4;
  int xc1,xc2,xc3,xc4,yc1,yc2,yc3,yc4;
  int x11,x12,x13,x14,y11,y12,y13,y14;
  int w1,w2,w3,w4,h1,h2,h3,h4;
  int N1,N2,N3,N4;

  oldwv = wv;

  CAMGetFrameRGB(ColPic[0][0]);

  RAS3_FindClasses2(ColPic, &x1, &y1, &x11, &y11, &x2, &y2, &x12, &y12, &x3, &y3, &x13, &y13,
&x4, &y4, &x14, &y14, &N1, &N2, &N3, &N4);

  if (w3 < 25 && h3 < 25) {
    int old_x1, old_y1;
    old_x1 = x1;
    old_y1 = y1;
    x1 = x1<x3?x1:x3;
    y1 = y1<y3?y1:y3;
    x11 = x11>x13?x11:x13;
    y11 = y11>y13?y11:y13;
    w1 = x11-x1;
    h1 = x12-x2;
  }

  xc1 = (x1+x11)/2;
  xc2 = (x2+x12)/2;
  xc3 = (x3+x13)/2;
  xc4 = (x4+x14)/2;
  yc1 = (y1+y11)/2;
  yc2 = (y1+y12)/2;
  yc3 = (y3+y13)/2;
  yc4 = (y4+y14)/2;

  if (N1 > 2) {
    wv.state = GET_BALL;
    if (yc1 > 100) {
      if (oldwv.state == SPIN_WAIT || oldwv.state == FIND_GOAL)
      wv.state = FIND_GOAL;
      else wv.state = SPIN_WAIT;
      wv.bally = 0;
    }
    else if (yc1 > 55) {
      wv.bally = 0.1;
    }
    else if (yc1 > 42) {
      wv.bally = 0.2;
    }
```

```
    else if (yc1 > 39) {
      wv.bally = 0.3;
    }
    else if (yc1 > 37) {
      wv.bally = 0.4;
    }
    else if (yc1 > 32) {
      wv.bally = 0.5;
    }
    else if (yc1 > 28) {
      wv.bally = 0.6;
    }
    else if (yc1 > 27) {
      wv.bally = 0.7;
    }
    else if (yc1 > 25) {
      wv.bally = 0.8;
    }
    else {
      wv.bally = 1;
    }
    wv.ballx = xc1-87;
    wv.ballx *= wv.bally/87.0;

    wv.ballw = atan2(wv.ballx,wv.bally)*RAD2DEG;
  }
  else
  wv.state = FIND_BALL;

  int *goalN, *goalx, *goaly;
  if (target_goal == GOAL_BLUE) {
    goalN = &N2;
    goalx = &xc2;
    goaly = &yc2;
  }
  else {
    goalN = &N3;
    goalx = &xc3;
    goaly = &yc3;
  }

  if (*goalN > 150) {
    if (wv.state == FIND_GOAL)
    wv.state = SCORE;
    if (*goalN > 10000) {
      wv.goaly = 0;
      if (wv.state == SCORE) wv.state = HAS_SCORED;
    }
    else if (*goaly > 85) {
      wv.goaly = 0.1;
    }
    else if (*goaly > 53) {
      wv.goaly = 0.2;
    }
    else if (*goaly > 43) {
      wv.goaly = 0.3;
    }
```

```
    else if (*goaly > 37) {
      wv.goaly = 0.4;
    }
    else if (*goaly > 32) {
      wv.goaly = 0.5;
    }
    else if (*goaly > 28) {
      wv.goaly = 0.6;
    }
    else if (*goaly > 27) {
      wv.goaly = 0.7;
    }
    else if (*goaly > 25) {
      wv.goaly = 0.8;
    }
    else {
      wv.goaly = 1;
    }

    wv.goalx = *goalx-87;
    wv.goalx *= wv.goaly/87.0;

    wv.goalw = atan2(wv.goalx,wv.goaly)*RAD2DEG;
    LCDSetPos(7, 0);
    LCDPrintf("gy:%d,gN:%d\n", *goaly, *goalN);

  }

  wv.timestamp = OSGetCount();
}
```

```
#ifndef eyelib_h
#define eyelib_h

#include <eyebot.h>

#define HEIGHT 144
#define WIDTH 176

#define RAS3_NO_CAMERA 255
#define RAS3_MSB 5
#define RAS3_LSB (8 - RAS3_MSB)
#define RAS3_VALUE 1<<RAS3_MSB
#define RAS3_NOHUE 255

#define CAMERA_ALPHA 2
#define CAMERA_HEIGHT 3
#define CAMERA_LENGTH 0
#define CAMERA_ANGLE 4.2
#define CAMERA_BETA 2.3
#define BALL_RADIUS 21

#define MEASURED_ANGLES 4
#define MEASURED_DISTANCES 4

enum Class {None = 0, Ball, Blue, Yellow, Opponent, class_count};
enum Value {H = 0, S, I, V, value_count};
enum Boundary {Lower = 0, Upper};

typedef BYTE RAS3_Table[RAS3_VALUE][RAS3_VALUE][RAS3_VALUE];

int RAS3_InitAll(void);

BYTE RAS3_CheckPixel(BYTE R, BYTE G, BYTE B);
BYTE RAS3_GetHue(BYTE,BYTE,BYTE);
int RAS3_InitCam(int);
void RAS3_InitThresholds();
void RAS3_InitRGBTable();
BYTE RAS3_RGB2Hue(BYTE, BYTE, BYTE);
BYTE RAS3_RGB2Hue2(BYTE, BYTE, BYTE);
BYTE RAS3_RGB2Sat(BYTE, BYTE, BYTE);
BYTE RAS3_RGB2Int(BYTE, BYTE, BYTE);
BYTE RAS3_RGB2Value(BYTE, BYTE, BYTE);
BYTE RAS3_IsClass(BYTE, BYTE, BYTE, enum Class);
enum Class RAS3_GetClass(BYTE, BYTE, BYTE);
void RAS3_DisplayError(char[16], int, char[16]);

void RAS3_NormRGB(BYTE *, BYTE *, BYTE *);

void RAS3_FindClasses(BYTE Pic[HEIGHT][WIDTH][3], int *, int *, int *, int *, int *, int *,int
 *, int *, int *, int *, int *, int *);

void RAS3_FindClasses2(BYTE Pic[HEIGHT][WIDTH][3], int *, int *, int *, int *, int *, int *,
int *, int *, int *, int *, int *, int *, int *, int *, int *, int *, int *, int *, int *, int
 *);

#endif
```

```c
#include "eyelib.h"
#include <math.h>
#include "distance.h"

static BYTE RAS3_IsBall(BYTE, BYTE, BYTE);
static BYTE RAS3_IsYellowGoal(BYTE, BYTE, BYTE);
static BYTE RAS3_IsBlueGoal(BYTE, BYTE, BYTE);
static BYTE RAS3_IsOpponent(BYTE,BYTE,BYTE);
static void RAS3_InitProgressBar();
static void RAS3_ProgressBar(BYTE);

BYTE thresholds[class_count][value_count][2];

#define MIN(a,b) (a<b?a:b)
#define MAX(a,b) (a>b?a:b)

RAS3_Table RGBTable;

BYTE RAS3_DistTable[WIDTH][HEIGHT];
BYTE RAS3_AngleTable[WIDTH][HEIGHT];

int pbarpos=-1;

int RAS3_InitAll(void)
{
  RAS3_InitCam(NORMAL);
  RAS3_InitThresholds();
  RAS3_InitRGBTable();
}

BYTE RAS3_CheckPixel(BYTE R, BYTE G, BYTE B)
{
  return RGBTable[R>>RAS3_LSB][G>>RAS3_LSB][B>>RAS3_LSB];
}

int RAS3_InitCam(int mode)
{
  int camversion = 0;
  int n = 0;

  LCDClear();
  LCDSetPrintf(1,1,"InitCam");
  camversion = CAMInit(mode);
  if(camversion == NOCAM || camversion == INITERROR)
  RAS3_DisplayError("InitCam",camversion,"");
  CAMMode(NOAUTOBRIGHTNESS);
  return camversion;
}

void RAS3_InitThresholds() {
  thresholds[Ball][H][Lower] = 0;
  thresholds[Ball][H][Upper] = 20;
  thresholds[Ball][S][Lower] = 75;
  thresholds[Ball][S][Upper] = 95;
  thresholds[Ball][I][Lower] = 95;
  thresholds[Ball][I][Upper] = 130;
  thresholds[Ball][V][Lower] = 235;
```

```
  thresholds[Ball][V][Upper] = 254;

  thresholds[Blue][H][Lower] = 120;
  thresholds[Blue][H][Upper] = 170;
  thresholds[Blue][S][Lower] = 0;
  thresholds[Blue][S][Upper] = 60;
  thresholds[Blue][I][Lower] = 0;
  thresholds[Blue][I][Upper] = 254;
  thresholds[Blue][V][Lower] = 0;
  thresholds[Blue][V][Upper] = 254;

  thresholds[Yellow][H][Lower] = 19;
  thresholds[Yellow][H][Upper] = 44;
  thresholds[Yellow][S][Lower] = 78;
  thresholds[Yellow][S][Upper] = 95;
  thresholds[Yellow][I][Lower] = 102;
  thresholds[Yellow][I][Upper] = 166;
  thresholds[Yellow][V][Lower] = 150;
  thresholds[Yellow][V][Upper] = 254;

  thresholds[Opponent][H][Lower] = 0;
  thresholds[Opponent][H][Upper] = 10;
  thresholds[Opponent][S][Lower] = 45;
  thresholds[Opponent][S][Upper] = 75;
  thresholds[Opponent][I][Lower] = 0;
  thresholds[Opponent][I][Upper] = 254;
  thresholds[Opponent][V][Lower] = 0;
  thresholds[Opponent][V][Upper] = 254;
}

BYTE RAS3_IsClass(BYTE R, BYTE G, BYTE B, enum Class c) {
  BYTE h = RAS3_RGB2Hue(R,G,B);
  BYTE s = RAS3_RGB2Sat(R,G,B);
  BYTE i = RAS3_RGB2Int(R,G,B);
  BYTE v = RAS3_RGB2Value(R,G,B);

  return (h > thresholds[c][H][Lower] && h < thresholds[c][H][Upper] &&
  s > thresholds[c][S][Lower] && s < thresholds[c][S][Upper] &&
  i > thresholds[c][I][Lower] && i < thresholds[c][I][Upper] &&
  v > thresholds[c][V][Lower] && v < thresholds[c][V][Upper]);
}

static void RAS3_InitProgressBar() {
  pbarpos = -1;
}
static void RAS3_ProgressBar(BYTE i) {
  if(i % (RAS3_VALUE>>4) ==0) {
    ++pbarpos;
    LCDSetPrintf(2,pbarpos,".");
  }
}
BYTE RAS3_RGB2Hue(BYTE R, BYTE G, BYTE B) {
  int hue = 0;
  BYTE max = MAX(R,MAX(G,B));
  BYTE min = MIN(R,MIN(G,B));
  BYTE delta = max - min;
```

```
  if(R == max)
  hue = (delta != 0 ? (42*(G-B)/delta):(42*(G-B)));
  else if(G == max)
  hue = (delta != 0 ? (84 + 42*(B-R)/delta):((84 + 42*(B - R))));
  else if(B == max)
  hue = (delta != 0 ? (168 + 42*(R-G)/delta):(168 + 42*(R - G)));

  if(hue < 0)
  hue += 255;
  return (BYTE)hue;
}
BYTE RAS3_RGB2Sat(BYTE R, BYTE G, BYTE B) {
  BYTE min = MIN(R,MIN(G,B));
  BYTE max = MAX(R,MAX(G,B));
  BYTE delta = max - min;
  BYTE sat;

  sat = (delta != 0 ? (100*((int)delta)/(int)max):(0));
  return sat;
}
BYTE RAS3_RGB2Int(BYTE R, BYTE G, BYTE B) {
  return ((BYTE)(((int)R+(int)G+(int)B)/3));
}
BYTE RAS3_RGB2Value(BYTE R, BYTE G, BYTE B) {
  return MAX(R, MAX(G,B));
}

void RAS3_InitRGBTable()
{
  LCDClear();
  LCDSetPrintf(1,1,"InitRGBTable");
  BYTE r,g,b;
  RAS3_InitProgressBar();
  for(r = 0; r < RAS3_VALUE; ++r)
  {
    for(g = 0; g < RAS3_VALUE; ++g)
    {
      for(b = 0; b < RAS3_VALUE; ++b)
      {
        RGBTable[r][g][b] = (BYTE)RAS3_GetClass(r<<RAS3_LSB,g<<RAS3_LSB,b<<RAS3_LSB);
      }
    }
    RAS3_ProgressBar(r);
  }
}

static BYTE RAS3_IsBall(BYTE R,BYTE G,BYTE B) {
  return RAS3_IsClass(R,G,B,Ball);
}
static BYTE RAS3_IsBlueGoal(BYTE R,BYTE G,BYTE B) {
  return RAS3_IsClass(R,G,B,Blue);
}
static BYTE RAS3_IsYellowGoal(BYTE R,BYTE G,BYTE B) {
  return RAS3_IsClass(R,G,B,Yellow);
}
static BYTE RAS3_IsOpponent(BYTE R,BYTE G,BYTE B) {
  return RAS3_IsClass(R,G,B,Opponent);
```

```
}
void RAS3_NormRGB(BYTE *R, BYTE *G, BYTE *B) {
  unsigned r,g,b;
  r = *R; g = *G; b = *B;
  unsigned L = r+b+g;
  if(L == 0)
  L = 1;
  (*R) = (BYTE)((100*(r)) / L);
  (*G) = (BYTE)((100*(g)) / L);
  (*B) = (BYTE)((100*(b)) / L);
}

void RAS3_FindClasses(BYTE Pic[HEIGHT][WIDTH][3], int *xc1, int *yc1, int *xc2, int *yc2, int
*xc3, int *yc3, int *xc4, int *yc4, int *N1, int *N2, int *N3, int *N4) {
  int x1, x2, x3, x4, y1, y2, y3, y4, w1, w2, w3, w4, h1, h2, h3, h4;
  RAS3_FindClasses2(Pic, &x1, &y1, &w1, &h1, &x2, &y2, &w2, &h2, &x3, &y3, &w3, &h3, &x4, &y4,
 &w4, &h4, N1, N2, N3, N4);

  if (N1 != 0) {
    *xc1 = x1+w1/2;
    *yc1 = y1+h1/2;
  }
  if (N2 != 0) {
    *xc2 = x2+w2/2;
    *yc2 = y2+h2/2;
  }
  if (N3 != 0) {
    *xc3 = x3+w3/2;
    *yc3 = y3+h3/2;
  }
  if (N4 != 0) {
    *xc4 = x4+w4/2;
    *yc4 = y4+h4/2;
  }
}

void RAS3_FindClasses2(BYTE Pic[HEIGHT][WIDTH][3], int *x11, int *y11, int *y12, int *x12, int
 *x21, int *y21, int *x22, int *y22, int *x31, int *y31, int *x32, int *y32, int *x41, int *
y41, int *x42, int *y42, int *N1, int *N2, int *N3, int *N4)
{
  BYTE c;
  int i,j;

  (*x11)=(*y11)=(*x21)=(*y21)=(*x31)=(*y31)=(*x41)=(*y41)=255;
  (*x12)=(*y12)=(*x22)=(*y22)=(*x32)=(*y32)=(*x42)=(*y42)=0;
  (*N1)=0;(*N2)=0;(*N3)=0;(*N4)=0;


  for(i=1;i<HEIGHT-1;i+=1) {
    for(j=1;j<(WIDTH-1);j+=1) {
      c = RGBTable[(Pic[i][j][0])>>RAS3_LSB][(Pic[i][j][1])>>RAS3_LSB][(Pic[i][j][2])>>
RAS3_LSB];
      if(c == 1) {
        if ((RAS3_CheckPixel(Pic[i][j-1][0],Pic[i][j-1][1], Pic[i][j-1][2]) == 1) + (
RAS3_CheckPixel(Pic[i][j+1][0],Pic[i][j+1][1], Pic[i][j+1][2]) == 1) + (RAS3_CheckPixel(Pic[i-
1][j][0],Pic[i-1][j][1], Pic[i-1][j][2]) == 1) + (RAS3_CheckPixel(Pic[i+1][j][0],Pic[i+1][j][1
], Pic[i+1][j][2]) == 1) > 0) {
```

```
          if (*x11 > j)
          *x11 = j;
          if (*x12 < j)
          *x12 = j;
          if (*y11 > i)
          *y11 = i;
          if (*y12 < i)
          *y12 = i;
          ++(*N1);
        }
      }
      else if(c == 2 && i < 70) {
        if ((RAS3_CheckPixel(Pic[i][j-1][0],Pic[i][j-1][1], Pic[i][j-1][2]) == 2) + (
RAS3_CheckPixel(Pic[i][j+1][0],Pic[i][j+1][1], Pic[i][j+1][2]) == 2) + (RAS3_CheckPixel(Pic[i-
1][j][0],Pic[i-1][j][1], Pic[i-1][j][2]) == 2) + (RAS3_CheckPixel(Pic[i+1][j][0],Pic[i+1][j][1
], Pic[i+1][j][2]) == 2) > 2) {
          if (*x21 > j)
          *x21 = j;
          if (*x22 < j)
          *x22 = j;
          if (*y21 > i)
          *y21 = i;
          if (*y22 < i)
          *y22 = i;
          ++(*N2);
        }
      }
      else if(c == 3 && i < 70) {
        if ((RAS3_CheckPixel(Pic[i][j-1][0],Pic[i][j-1][1], Pic[i][j-1][2]) == 3) + (
RAS3_CheckPixel(Pic[i][j+1][0],Pic[i][j+1][1], Pic[i][j+1][2]) == 3) + (RAS3_CheckPixel(Pic[i-
1][j][0],Pic[i-1][j][1], Pic[i-1][j][2]) == 3) + (RAS3_CheckPixel(Pic[i+1][j][0],Pic[i+1][j][1
], Pic[i+1][j][2]) == 3) > 2) {
          if (*x31 > j)
          *x31 = j;
          if (*x32 < j)
          *x32 = j;
          if (*y31 > i)
          *y31 = i;
          if (*y32 < i)
          *y32 = i;
          ++(*N3);
        }
      }
    }
  }
}

void RAS3_DisplayError(char title[16], int error, char msg[16])
{
  LCDClear();
  LCDSetPrintf(0,0,title);
  LCDSetPrintf(2,0,"Errorcode: %d", error);
  LCDSetPrintf(3,0,"Message:");
  LCDSetPrintf(4,0,msg);
  LCDMenu("", "", "", "Cont");
  KEYWait(KEY4);
}
```

```
BYTE pre_angle[MEASURED_ANGLES] = {0, 23, 43, 54};
BYTE pre_dist[MEASURED_DISTANCES] = {0, 32, 54, 75};
BYTE pre_x[MEASURED_ANGLES][MEASURED_DISTANCES] = {{0, 0, 175, 175},{0, 0, 175, 175},{0, 0,
175, 175},{0, 0, 175, 175}};
BYTE pre_y[MEASURED_ANGLES][MEASURED_DISTANCES] = {{0, 0, 175, 175},{0, 0, 175, 175},{0, 0,
175, 175},{0, 0, 175, 175}};

void RAS3_InitDistTable(void)
{
  LCDClear();
  LCDSetPrintf(1,1,"InitDistTable");
  RAS3_InitProgressBar();
  int i,j,a,d;
  int best_a, best_d;
  int x1, x2, x3, x4;
  int y1, y2, y3, y4;
  int d1, d2, d3, d4;
  for(i = 0; i < WIDTH; ++i) {
    for(j = 0; j < HEIGHT; ++j) {
      best_a = 0; best_d = 0;
      for (a = 0; a < MEASURED_ANGLES; ++a) {
        for (d = 0; d < MEASURED_DISTANCES; ++d) {
          if (pre_angle[a]+pre_dist[d] < pre_angle[best_a]+pre_dist[best_d]) {
            best_a = a;
            best_d = d;
          }
        }
      }
      RAS3_DistTable[i][j] = pre_dist[best_d];
      RAS3_AngleTable[i][j] = pre_angle[best_a];
    }
  }
}
```

thresholds.c

```
#include "eyelib.h"

enum Class RAS3_GetClass(BYTE R, BYTE G, BYTE B) {
  BYTE h = RAS3_RGB2Hue(R,G,B);
  BYTE s = RAS3_RGB2Sat(R,G,B);
  BYTE i = RAS3_RGB2Int(R,G,B);
  BYTE v = RAS3_RGB2Value(R,G,B);

  if (R >= 112 && R <= 240 &&
    G >= 48 && G <= 240 &&
    B >= 16 && B <= 24 &&
    h >= 10 && h <= 42 &&
    s >= 85 && s <= 93 &&
    i >= 64 && i <= 165 &&
    v >= 112 && v <= 240)
      return Yellow;

  if (R >= 128 && R <= 240 &&
    G >= 16 && G <= 144 &&
    B >= 16 && B <= 16 &&
    h >= 0 && h <= 24 &&
    s >= 87 && s <= 93 &&
    i >= 58 && i <= 133 &&
    v >= 128 && v <= 240)
      return Ball;

  if (R >= 56 && R <= 152 &&
    G >= 64 && G <= 184 &&
    B >= 80 && B <= 192 &&
    h >= 91 && h <= 189 &&
    s >= 7 && s <= 52 &&
    i >= 66 && i <= 170 &&
    v >= 80 && v <= 192)
      return Blue;

  return None;
}
```

Makefile

```
CC := gcc68

all : $(patsubst %.c,%.hex,$(wildcard *.c))

%.hex : %.o
  $(CC) -o $@ $^ -lm

.PHONY : clean all

clean :
  rm -f *.o *.hex

# The above assumes that every .c-file should be compiled into a separate .hex-file
# This behaviour can be modified by adding one or more of the rules below.

# foo should be linked with bar and baz.
# foo.hex : bar.o baz.o

# bar.hex and baz.hex should not be made.
# bar.hex baz.hex : ;

# foo.c and bar.c both include bar.h.
# foo.o bar.o : bar.h

main.hex : camera.o action.o eyelib.o thresholds.o

camera.hex action.hex eyelib.hex thresholds.hex : ;

main.o camera.o action.o : globals.h

camera.o : eyelib.h

main.o action.o : action.h
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Point;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedInputStream;
import java.io.OutputStreamWriter;
import java.util.Arrays;

import javax.swing.Box;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.event.MouseInputListener;

public class ThresholdFinder extends JFrame {
  private static final long serialVersionUID = 1L;
    static int columns = 5;
    static int rows = 4;
  static final int w = 176;
  static final int h = 144;
    static int scale = 1;
    static final int bitmask = 0xFF-7;

  static final String[] classNames = {"None", "Ball", "Blue", "Yellow", "Opponent"};
  static final int[] classPrio = {3, 1, 2, 4};

  static final String outputfile = "thresholds.c";

  JLabel classInfo;
  JLabel colorInfo;
  JLabel invertInfo;

  Thresholds thresholds;

  int currentClass;
  boolean invert = true;

  public static void main(String[] args) {
      if(args.length >= 1) {
    ThresholdFinder.scale = Integer.parseInt(args[0]);
    ThresholdFinder.columns = 1;
    ThresholdFinder.rows = 1;
```

```
    }
    if(args.length == 3) {
  ThresholdFinder.columns = Integer.parseInt(args[1]);
  ThresholdFinder.rows = Integer.parseInt(args[2]);
    }
  new ThresholdFinder();
}

public ThresholdFinder() {
  super("ThresholdFinder");
  thresholds = new Thresholds();

  JPanel imagesPanel = new JPanel();
  add(imagesPanel, BorderLayout.CENTER);
  imagesPanel.setLayout(new GridLayout(rows, columns, 2, 2));
  for (int i = 0; i < rows*columns; ++i) {
    imagesPanel.add(new ThresholdPanel());
  }

  Box infoPanel = Box.createHorizontalBox();
  add(infoPanel, BorderLayout.SOUTH);
  invertInfo = new JLabel();
  infoPanel.add(invertInfo);
  classInfo = new JLabel();
  infoPanel.add(classInfo);
  infoPanel.add(Box.createHorizontalGlue());
  colorInfo = new JLabel();
  infoPanel.add(colorInfo);

  setClass(1);
  toggleInvert();


  addKeyListener(new FinderListener());
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  pack();
  setResizable(false);
  setVisible(true);
}

void setClass(int c) {
  currentClass = c;
  classInfo.setText(classNames[c]);
  repaint();
}

void toggleInvert() {
  invert = !invert;
  if (invert)
    invertInfo.setText("Remove From ");
  else
    invertInfo.setText("Add to ");
}

void exportThresholds() {
  try {
    BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(
```

```
outputfile)));
      out.write("#include \"eyelib.h\"\n\n");
      out.write("enum Class RAS3_GetClass(BYTE R, BYTE G, BYTE B) {\n");
      out.write("  BYTE h = RAS3_RGB2Hue(R,G,B);\n");
      out.write("  BYTE s = RAS3_RGB2Sat(R,G,B);\n");
      out.write("  BYTE i = RAS3_RGB2Int(R,G,B);\n");
      out.write("  BYTE v = RAS3_RGB2Value(R,G,B);\n\n");
      for (int i = 0; i < classPrio.length; ++i) {
        for (int j = 0; j < thresholds.classes[classPrio[i]].length; ++j) {
          if (thresholds.classes[classPrio[i]][j].upper[0] == 0 &&
              thresholds.classes[classPrio[i]][j].lower[0] == 0) continue;
          out.write("  if (R >= "+thresholds.classes[classPrio[i]][j].lower[0]+" && ");
          out.write("R <= "+thresholds.classes[classPrio[i]][j].upper[0]+" &&\n");
          out.write("    G >= "+thresholds.classes[classPrio[i]][j].lower[1]+" && ");
          out.write("G <= "+thresholds.classes[classPrio[i]][j].upper[1]+" &&\n");
          out.write("    B >= "+thresholds.classes[classPrio[i]][j].lower[2]+" && ");
          out.write("B <= "+thresholds.classes[classPrio[i]][j].upper[2]+" &&\n");
          out.write("    h >= "+thresholds.classes[classPrio[i]][j].lower[3]+" && ");
          out.write("h <= "+thresholds.classes[classPrio[i]][j].upper[3]+" &&\n");
          out.write("    s >= "+thresholds.classes[classPrio[i]][j].lower[4]+" && ");
          out.write("s <= "+thresholds.classes[classPrio[i]][j].upper[4]+" &&\n");
          out.write("    i >= "+thresholds.classes[classPrio[i]][j].lower[5]+" && ");
          out.write("i <= "+thresholds.classes[classPrio[i]][j].upper[5]+" &&\n");
          out.write("    v >= "+thresholds.classes[classPrio[i]][j].lower[6]+" && ");
          out.write("v <= "+thresholds.classes[classPrio[i]][j].upper[6]+")\n");
          out.write("      return "+classNames[classPrio[i]]+";\n\n");
        }
      }
      out.write("  return None;\n");
      out.write("}\n");
      out.close();
    } catch (IOException exc) {
      System.out.println(exc);
    }
  }

  void exportTable() {
    try {
      setTitle("ThresholdFinder - saving");
      int[] v = null;
      BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(
outputfile)));
      out.write("static const RAS3_Table RGBTable = {\n");
      for (int r = 0; r < 256; ++r) {
        out.write("{\n");
        for (int g = 0; g < 256; ++g) {
          out.write("{");
          colorloop: for (int b = 0; b < 256; ++b) {
            int rgb = r | (g << 8) | (b << 16);
            v = getValues(rgb, v);
            for (int c = 0; c < classPrio.length; ++c) {
              if (thresholds.isInClass(v, classPrio[c])) {
                out.write((classPrio[c]+'0')+",");
                continue colorloop;
              }
            }
            out.write("0,");
```

```
        }
        out.write("},\n");
      }
      out.write("},");
    }
    out.write("}\n");
  } catch (IOException exc) {
    System.err.println(exc);
  }
  setTitle("ThresholdFinder");
}

int[] getValues(int rgb, int[] v) {
  int r = (rgb >> 16) & bitmask;
  int g = (rgb >> 8) & bitmask;
  int b = rgb & bitmask;
  if (v == null) v = new int[7];
  v[0] = r;
  v[1] = g;
  v[2] = b;
  v[3] = h(r,g,b);
  v[4] = s(r,g,b);
  v[5] = i(r,g,b);
  v[6] = v(r,g,b);
  return v;
}

int h(int r, int g, int b) {
  int hue = 0;
  int max = Math.max(r, Math.max(g, b));
  int min = Math.min(r, Math.min(g, b));
  int delta = max - min;

  if (r == max)
    hue = (delta != 0 ? (42 * (g - b) / delta) : (42 * (g - b)));
  else if (g == max)
    hue = (delta != 0 ? (84 + 42 * (b - r) / delta)
        : ((84 + 42 * (b - r))));
  else if (b == max)
    hue = (delta != 0 ? (168 + 42 * (r - g) / delta)
        : (168 + 42 * (r - g)));

  if (hue < 0)
    hue += 255;
  return hue;
}

int s(int r, int g, int b) {
  int max = Math.max(r, Math.max(g, b));
  int min = Math.min(r, Math.min(g, b));
  int delta = max - min;
  return (delta != 0 ? (100*delta/max):(0));
}

int i(int r, int g, int b) {
  return (r+g+b)/3;
}
```

```
int v(int r, int g, int b) {
  return Math.max(r, Math.max(g, b));
}

class FinderListener implements KeyListener {
  public void keyTyped(KeyEvent e) {
    char c = e.getKeyChar();
    if (Character.isDigit(c) && c-'0' > 0 && c-'0' < classNames.length) {
      setClass(c-'0');
    } else if (c == ' ') {
      toggleInvert();
    }
  }

  public void keyPressed(KeyEvent e) {
    int c = e.getKeyCode();
    if (e.isControlDown()) {
      switch (c) {
      case KeyEvent.VK_Z:
        thresholds.undo();
        break;
      case KeyEvent.VK_S:
        exportThresholds();
        break;
      }
    }
  }
  public void keyReleased(KeyEvent e) {}
}

class ThresholdPanel extends JPanel {
  private static final long serialVersionUID = 1L;
  BufferedImage image;
  Graphics imageGraphics;
  Point dragStart;

  Rectangle selection;

  public ThresholdPanel() {
    image = new BufferedImage(w, h, BufferedImage.TYPE_3BYTE_BGR);
    imageGraphics = image.getGraphics();
    selection = new Rectangle();
    ThresholdPanelListener listen = new ThresholdPanelListener(this);
    addMouseListener(listen);
    addMouseMotionListener(listen);
    setPreferredSize(new Dimension(w*scale, h*scale-1));
  }

    void loadImage(File file) {
  try {
  BufferedInputStream in = new BufferedInputStream(new FileInputStream(file));
  for (int i = 0; i < h; ++i) {
      for (int j = 0; j < w; ++j) {
    int rgb = (in.read() << 16) | (in.read() << 8) | in.read();
    image.setRGB(j, i, rgb);
      }
```

```
    }
    } catch (IOException exc) {
        System.err.println(exc);
    }
      }

    void loadImage2(File file) {
      try {
        BufferedReader in = new BufferedReader(new InputStreamReader(new FileInputStream(file
)));
        String line;
        String[] words;
        for (int i = 0; i < h; ++i) {
          for (int j = 0; j < w; ++j) {
            line = in.readLine();
            if (line == null) {
              System.out.println("Corrupted file");
              return;
            }
            words = line.split(" ");
            if (words.length != 3) {
              System.out.println("Corrupted file");
              return;
            }
            imageGraphics.setColor(new Color(Integer.parseInt(words[0]), Integer.parseInt(
words[1]), Integer.parseInt(words[2])));
            imageGraphics.drawLine(j, i, j, i);
          }
        }
      } catch (IOException exc) {
        System.err.println(exc);
      }
    }

    @Override
    protected void paintComponent(Graphics g) {
        if (scale == 1)
      g.drawImage(image, 0, 0, this);
        else
      g.drawImage(image.getScaledInstance(w*scale, h*scale, BufferedImage.SCALE_REPLICATE), 0,
 0, this);
      if (dragStart != null) {
        g.setColor(new Color(1f, 1f, 1f, 0.5f));
        g.fillRect(selection.x, selection.y, selection.width, selection.height);
      }
      g.setColor(Color.PINK);
      for (int y = 0; y < image.getHeight(); ++y) {
        for (int x = 0; x < image.getWidth(); ++x) {
          int rgb = image.getRGB(x, y);
          if (thresholds.isInClass(getValues(rgb, null), currentClass)) {
            g.drawLine(x, y, x, y);
          }
        }
      }
    }

    class ThresholdPanelListener implements MouseInputListener {
```

```
  ThresholdPanel panel;

  public ThresholdPanelListener(ThresholdPanel panel) {
    this.panel = panel;
  }

  public void mouseClicked(MouseEvent e) {
    if (e.getButton() == MouseEvent.BUTTON3) {
      JFileChooser fc = new JFileChooser(".");
      if (fc.showOpenDialog(panel) == JFileChooser.APPROVE_OPTION)
        panel.loadImage(fc.getSelectedFile());
    }
  }

  public void mousePressed(MouseEvent e) {
    if (e.getButton() == MouseEvent.BUTTON1) {
      dragStart = e.getPoint();
      selection.setBounds(e.getX(), e.getY(), 0, 0);
    }
  }

  public void mouseReleased(MouseEvent e) {
    if (e.getButton() == MouseEvent.BUTTON1) {
      dragStart = null;
      e.getComponent().repaint();
      thresholds.update(image, selection);
    }
  }

  public void mouseEntered(MouseEvent e) {
    if ((e.getModifiersEx() & MouseEvent.BUTTON1_DOWN_MASK) == 0)
      dragStart = null;
  }

  public void mouseExited(MouseEvent e) {}

  public void mouseDragged(MouseEvent e) {
    if ((e.getModifiersEx() & MouseEvent.BUTTON1_DOWN_MASK) != 0) {
      int x1 = Math.max(0, Math.min(dragStart.x, e.getX()));
      int y1 = Math.max(0, Math.min(dragStart.y, e.getY()));
      int x2 = Math.min(w-1, Math.max(dragStart.x, e.getX()));
      int y2 = Math.min(h-1, Math.max(dragStart.y, e.getY()));
      selection.setBounds(x1, y1, x2-x1, y2-y1);
      e.getComponent().repaint();
    }
    mouseMoved(e);
  }

  public void mouseMoved(MouseEvent e) {
      int x = e.getX()/scale;
      int y = e.getY()/scale;
    if (new Rectangle(0, 0, w-1, h-1).contains(x,y)) {
        int rgb = image.getRGB(x, y);
      int r = (rgb >> 16) & bitmask;
      int g = (rgb >> 8) & bitmask;
      int b = rgb & bitmask;
```

```
        int c = 0;
        for (int i = 0; i < classPrio.length; ++i) {
            if (thresholds.isInClass(getValues(rgb, null), classPrio[i])) {
          c = classPrio[i];
          break;
            }
          }
        colorInfo.setText(x+","+y+": "+classNames[c]+" ("+r+","+g+","+b+")");
      }
    }
  }
}

class Thresholds {
  Class[][] classes;

  int undoIndex;
  Class[] undoData;

  public Thresholds() {
    classes = new Class[classNames.length][];
    classes[1] = new Class[1];
    classes[2] = new Class[1];
    classes[3] = new Class[1];
    classes[4] = new Class[1];
    for (int i = 1; i < classes.length; ++i) {
      for (int j = 0; j < classes[i].length; ++j) {
        classes[i][j] = new Class();
      }
    }

    undoData = null;
  }

  void update(BufferedImage image, Rectangle selection) {
    int[] v = null;
    backup();
    for (int y = 0; y < selection.height; ++y) {
      for (int x = 0; x < selection.width; ++x) {
        int rgb = image.getRGB(x+selection.x, y+selection.y);
        v = getValues(rgb, v);
        if (invert) {
          for (int i = 0; i < classes[currentClass].length; ++i) {
            if (classes[currentClass][i].contains(v)) {
              classes[currentClass][i].retract(v);
              break;
            }
          }
        } else {
          int shortest = Integer.MAX_VALUE;
          int shortestI = 0;
          for (int i = 0; i < classes[currentClass].length; ++i) {
            int d = classes[currentClass][i].distance(v);
            if (d < shortest) {
              shortest = d;
              shortestI = i;
            }
```

```
            }
            if (classes[currentClass].length == 2 && classes[currentClass][0].distance(classes
[currentClass][1]) < shortest) {
              classes[currentClass][0].merge(classes[currentClass][1]);
              classes[currentClass][1].clear();
              classes[currentClass][1].expand(v);
            } else if (classes[currentClass].length > 2) {
              System.err.println("Add support for more than 2 classes first!");
              System.exit(0);
            } else {
              classes[currentClass][shortestI].expand(v);
            }
          }
        }
      }
      repaint();
    }

    boolean isInClass(int[] v, int c) {
      for (int i = 0; i < classes[c].length; ++i) {
        if (classes[c][i].contains(v))
          return true;
      }
      return false;
    }

    void backup() {
      undoIndex = currentClass;
      undoData = new Class[classes[undoIndex].length];
      for (int i = 0; i < undoData.length; ++i) {
        undoData[i] = new Class(classes[undoIndex][i]);
      }
    }

    void undo() {
      if (undoData != null) {
        Class[] temp = classes[undoIndex];
        classes[undoIndex] = undoData;
        undoData = temp;
      }
      repaint();
    }
  }

  class Class {
    int[] upper;
    int[] lower;

    public Class() {
      upper = new int[7];
      lower = new int[7];
      Arrays.fill(upper, 0);
      Arrays.fill(lower, 0);
    }

    public Class(Class c) {
      upper = new int[7];
```

```
      lower = new int[7];
      System.arraycopy(c.upper, 0, upper, 0, upper.length);
      System.arraycopy(c.lower, 0, lower, 0, lower.length);
    }

    boolean contains(int[] v) {
      for (int j = 0; j < v.length; ++j) {
        if (lower[j] > v[j] || upper[j] < v[j]
            || (upper[j] == 0 && lower[j] == 0)) {
          return false;
        }
      }
      return true;
    }

    int distance(Class c) {
      int d = 0;
      for (int i = 1; i < upper.length; ++i) {
        if (upper[i] != 0 || lower[i] != 0) {
          if (upper[i] < c.lower[i]) d += c.lower[i]-upper[i];
          if (lower[i] > c.upper[i]) d += lower[i]-c.upper[i];
        }
      }
      return d;
    }

    void merge(Class c) {
      for (int i = 1; i < upper.length; ++i) {
        if (upper[i] != 0 || lower[i] != 0) {
          lower[i] = Math.min(lower[i], c.lower[i]);
          upper[i] = Math.max(upper[i], c.upper[i]);
        } else if (c.upper[i] != 0 || c.lower[i] != 0) {
          lower[i] = c.lower[i];
          upper[i] = c.upper[i];
        }
      }
    }

    int expand(int[] v) {
      int d = 0;
      for (int i = 0; i < v.length; ++i) {
        if (upper[i] == 0 && lower[i] == 0) {
          upper[i] = v[i];
          lower[i] = v[i];
        } else {
          if (v[i] < lower[i]) {
            d += lower[i]-v[i];
            lower[i] = v[i];
          }
          if (v[i] > upper[i]) {
            d += v[i]-upper[i];
            upper[i] = v[i];
          }
        }
      }
      return d;
    }
```

```
    void retract(int[] v) {
      int shortest = Integer.MAX_VALUE;
      int shortestI = 0;
      boolean shortestUpper = true;
      for (int i = 0; i < v.length; ++i) {
        if (upper[i] - v[i] < shortest) {
          shortest = upper[i] - v[i];
          shortestI = i;
          shortestUpper = true;
        }
        if (v[i] - lower[i] < shortest) {
          shortest = v[i] - lower[i];
          shortestI = i;
          shortestUpper = false;
        }
      }
      if (shortestUpper)
        upper[shortestI] = v[shortestI]-1;
      else
        lower[shortestI] = v[shortestI]+1;
    }

    int distance(int[] v) {
      int d = 0;
      for (int i = 1; i < v.length; ++i) {
        if (upper[i] != 0 || lower[i] != 0) {
          if (v[i] < lower[i]) {
            d += lower[i]-v[i];
          }
          if (v[i] > upper[i]) {
            d += v[i]-upper[i];
          }
        }
      }
      return d;
    }

    void clear() {
      Arrays.fill(upper, 0);
      Arrays.fill(lower, 0);
    }

    @Override
    public String toString() {
      return Arrays.toString(upper) + " " + Arrays.toString(lower);
    }
  }
}
```