KUNGLIGA TEKNISKA HÖGSKOLAN **15 juni 2007**
CSC
Project Report

Robotics and Autonomous Systems

Boten Anna Project Report

Martin Andersson	830110-7176	martin3@kth.se
Christian Ziethén	810812-3350	ziethen@kth.se
Felix Hammarstrand	840913-0195	felix41@gmail.com
Carl-Fredrik Ullberg	860128-0053	cful@kth.se

Abstract

Boten Anna, an autonomous soccer playing robot designed in the course 2D1426 at Nada, KTH. This report covers the hardware and software implementations of Boten Anna and shows the results.

Contents

1 Background

In the course Robotic and Autonomous Systems at the Royal institute of technology we have got the task to construct and design a football robot. We were divided in teams of four persons in each. The first part of the course consisted of lectures and the robot slowly growing from ideas and sketches to construction and implementation. The latter part of the course was a race against time to get the robot finished for the competition

1.1 Project Goals

The goal of the project was to get knowledge about the basic concepts and technologies in the broad and interdisciplinary field of robotics and to get experience of building and programming an autonomous robot. One other goal was to have a general idea about the possibilities and limitations for robot technology of today.

1.2 Rules of the game

There were many rules but the general idea is described below

- The robot must fit in a 180 mm diameter vertical cylinder at all times
- The robot should be autonomously, meaning no help from the outside
- It is not allowed to hold the ball; no degrees of freedom of the ball should be removed
- Everything on the playing field is colour coded: the goals are blue respective yellow and the ball is an orange golf ball. The opponent has a purple stripe around it. This makes feature based recognition a lot easier.

1.3 Materials and tools

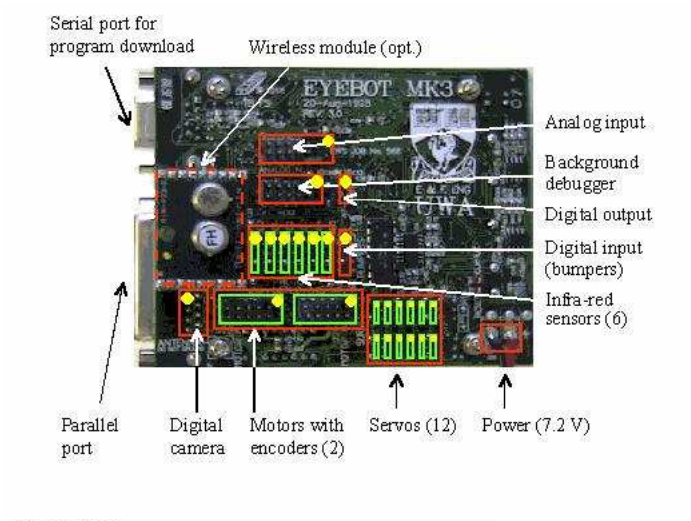
All groups were provided with these things at the start of the project

- Eyebot and a corresponding camera
- Two motors with encoders and wheels,
- 3 sheets of aluminium
- A RC servo

We also had access to a small workshop where we could do some drilling, bending and cutting of our aluminium sheets.

1.4 Eyebot

The eyebot (version M4) is the core of the robot and has a powerful 25Mhz 32bit Motorola processor (see Figure 1) . It acts as a development board for the robot and has the following main features:



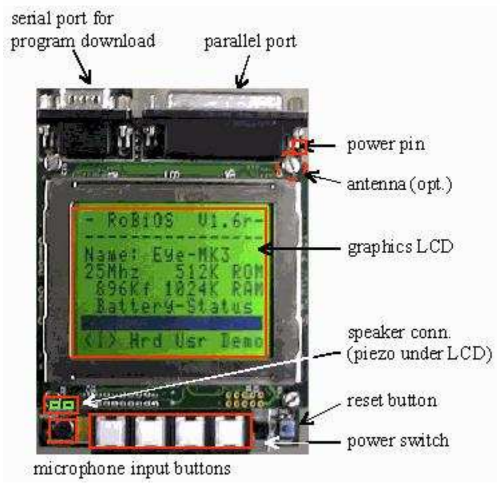


Figure 1: Eyebot

- LCD (128x64 pixels) which is useful for debugging.
- Two integrated drivers and encoders for the motors
- 16 programmable timing processor I/Os where you can connect for example the RC servo
- 4 input and 8 output pins for general I/O
- 4 buttons, mic and a speaker
- Serial connector which you use to program your robot

The Eyebot also has an operating system, the RoBIOS. The interface between the hardware and the software has a basic structure described in the Hardware Description Table (HDT). If you connect a supported hardware device and describe it in the HDT file, you will have access to a test function in the RoBIOS when you start the eyebot. The RoBIOS also has a large library of functions which mostly are a higher level interface to the hardware instead of accessing it directly. For example the omega drive function gives you a velocity and position feedback control for the movement of the robot.

2 General Approach

Initially we set out to build a state machine with no mapping or memory of the surrounding, the camera was used to detect whether we had control of the ball or not. A lot of effort was focused on making the roller perform well, so that the robot would be able to turn freely without worrying about losing the ball. There were issues with image processing while running the $V\omega$-drive, so the robot took a snapshot and then issued a command and didn't take another picture until the movement command was completed. This approach generated two problems, we couldn't see far enough due to the camera being tilted downwards and if we ran into a wall we would keep moving into it since the wheels wouldn't spin enough due to being stalled by the wall. By adding an IR-sensor the robot was able to detect whether it had the ball while maintaining the camera in a position that allowed it to spot the ball at roughly a 1m distance. In order to solve the second problem we added four whiskers to the robot which once configured properly made wall detection trivial. This also lead us to change our general tactic from simply looking for the goal once we had the ball, to actively following the wall on the robots right side.

3 Hardware

3.1 Camera

The camera is a 24 bit digital colour camera with 80x60 resolution.

3.2 Bumpers

Bumpers, or Whiskers, are made to make the eyebot aware of close-by objects such as walls and opponents. In this case the bumpers were made out of a small metallic cylinder and a wire sticking out of the cylinder, not touching its edges. Whenever a bumper were pushed against a wall the wire got in connection with the metallic cylinder and the circuit were closed. With only a little help from a Pull-Up resistor ($10k\Omega$ the circuit were complete (see Figure 2). The robot used 4 of these, one at every corner of the robot.

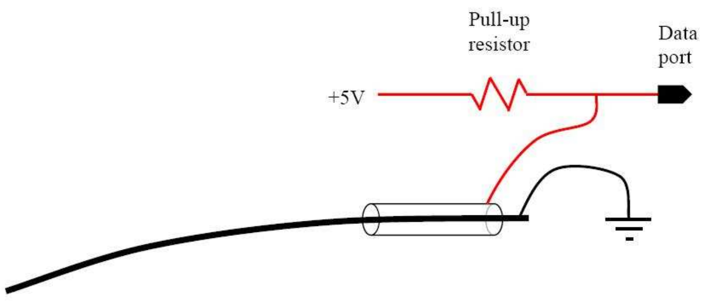


Figure 2: Bumper Circuit

3.3 Roller

To be able to move the robot without loosing the ball once we had captured it we needed some kind of ball control device. Therefore we placed a spinning roller in the front of the robot, giving the ball backspin and a backward motion towards the robot.

To construct a good backspin roller appeared to be much more troublesome than we first had anticipated.

The problem lay in having an even surface with enough friction to suck up a ball in motion and make it stay and not having it bouncing off again because of a bump or that the roller was not fixed along its axis.

Our first try was a roller in wood made out of a broom shaft. The idea was to lathe the roller so that mid-radius got smaller than the edge radius and accordingly push the rolling ball towards the centre of the robot. This roller design failed when we could not get access to a lathe machine. But we also saw problems with using a wooden core for the roller, it was very hard to fix the roller along its axis with good enough precision to make it spin without wobbling.

For our second try we decided to give up the idea of varying radius and go for a cylinder shaped roller. For this we used a plastic cable pipe with a radius of 16 mm. We used a 5 mm steal beam as axle and wrapped paper around it to fill out the gap. This solution proved very hard to carry out. The problem was that it was very hard to get an even distribution of the paper and create a roller that does not wobble. We used this type of roller in our final robot but this problem was never really solved. The wobbling also created a lot of noise and vibrations.

3.3.1 Coverings

When we tested this roller we promptly found that the smooth plastic surface of the plastic pipe did not create enough friction to make the ball spin. We needed to cover our pipe with something more sticky.

We tried many different coverings. Some of the more interesting types were:

Velcro-tape: Did not create any better grip than the original plastic surface. Useless.

Rubber bands: We tried to cover our roller with lots of different rubber bands. Rubber can have an astonishing good capacity to absorb and catch a ball. But it varied very much from type to type of the rubber bands. In fact our best roller was made out of rubber bands. Unfortunate the bands were coloured yellow which made them impossible to use in the competition. We tried to spray-paint it black but the paint destroyed the rubber surface. An attempt to do a second rubber band roller with brown bands was made but the brown rubber bands did not create the same grip the yellow ones did. Other problems with rubber bands were that the winding of the bands created an uneven surface and plenty of joint which created sore points where the roller could be damaged if it collided with an edge on the field or an enemy robot.

Textile glue: Textile glue can create a rubber like surface and fairly easy to spread out and create an even surface. Our glue-roller failed to reach our demands, probably because we did not put enough glue on. This is a concept which might very well do the trick if it is carried out the right way. But there are even better alternatives than textile glue, e.g. liquid latex.

Party balloons: Classic inflatable rubber party balloons can be slipped on to the pipe. And they come in all possible imaginable colours! They have nearly as good grip as rubber bands but are not flawed by the seams. We used a white balloon for our roller. When stretched out over the roller the balloon got a light shade of yellow. This was solved by putting a blue balloon under the white one, counteracting the yellow shade, resulting in a white roller. Balloons proved to be a very good cover and were our choice for our final roller. It worked so well that two other groups received spare balloons from us that they used on their own rollers.

3.3.2 Motor

The roller was driven by a permanent magnetised DC-motor and we set the speed of the roller by adjusting the voltage over the motor. The motor was recycled from an earlier soccer-robot which made it impossible for us to find any specifications for the motor. But through experiment we found that 4 * 1.2 V rechargeable batteries resulted in an angular velocity adequate for our needs. We also tried to power the motor through the eyebot but the motor made the batteries drop in voltage very quick and powering it through the eyebot made it harder to turn the motor on when needed. Therefore we used a secondary battery chunk on our robot, exclusively of the motor. The motor together with the wobbling roller created enough vibrations to make screw nuts fall of and we were worried that the vibrations would cause the camera to take poor photos. But our design and strategy was built on having an excellent roller which made these drawbacks a fair trade. Screw nuts were fixed with hot melt glue and camera vibrations taken care of in the calibration of the camera software.

3.3.3 Belt

To connect the motor to the roller we used a rubber band as a belt. The motor and roller were placed very close to each other and we did not find a rubber band small enough to use without folding it once, making the band go twice between them. This caused a lot of stress and wear on the rubber and the band had to be changed often or it could break. This solution is not recommended for future robots.

3.3.4 Suspension

We fixed the steel axle between two metal bars sticking out from either side of the front of the robot. These bars also served as stop blocks at the sides making the ball stay in front of the robot and not slip away from the roller when the robot turns. We placed the rollers axle above the centre of the ball to make the force directed towards the robot and not just down towards the field as illustrated below.

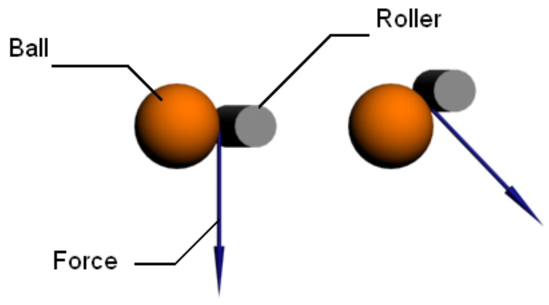


Figure 3: Suspension

3.4 Detection of the ball

One problem to solve was for the robot to know if it had the ball or not. The first approach to this problem was using the information we got from the camera, and if the position of the ball was close enough to the robot, we would deduct we had the ball. This solution had some problems. First, because the camera has a small angle of sight, we had to turn the camera downward to make sure we saw the ball. When doing this we lost the ability to see far away. Secondly, the probability of actually having the ball or not was not great because the lower threshold parameter, when deciding if we had the ball or not, was hard to calibrate.

A more robust and exact solution is to make the ball cut a beam of Infrared-light in front of the robot. The construction is made with a IR-diode and a IR-sensor. Almost everything that is warm emits IR-light so to distinguish the light from all the others sources the robot sends a 38Khz modulated IR-light, see figure 4.

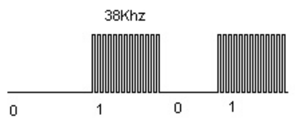


Figure 4: 38khz modulated signal

The implementation is described below: The IR-diode used is a IR/HIR 333 from Elfa and the circuit schema is seen in figure 5. The diode is connected to a TPU channel on the eyebot. By doing this you could easily get an 38Khz singal from the eyebot. The software implementation is done in the HDT file.

The IR-sensor used is a ELIRM8601S from Elfa consist first of a IR-diode (sensor), amplifier, band-pass filter and a comparator. The sensor only reacts to IR-light with the wavelength at 980nm modulated at 38khz.

It is TTL compatible which makes it easy to implement with the eyebot. The

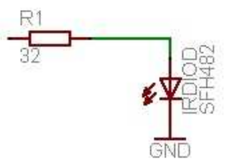


Figure 5: IR diode schematics

circuit schema is seen in figure 6. The sensor's output pin is connected to a digital in on the robot. Because we already used the 4 available input pins for the bumpers, we found out that the PSD-Infrared interface on the eyebot had a free input pin.

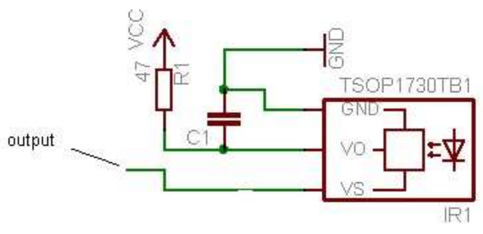


Figure 6: IR sensor schematics

3.5 Construction

The two motors as well as an upside down screw were our three contact points with the surface. Both motors were fastened directly on the main platforms side wings which were bent downwards, initially this lead to a problem with the wheels being tilted so a threaded rod was attached in order to hold both the wings together and keep the wheels straight. The main platform was intentionally made to be rather small in order to make room for a wide roller, at first the camera was located on a servo situated directly on the platform, but with the camera tilted downwards this gave poor visibility while scanning the surroundings for the ball or the enemy goal. After failing to configure this setup properly a tower was added in order to position the camera higher for better long range visibility. The motors were positioned below the main platform in order to raise the platform so that the roller would get a higher position, initial tests had shown that a roller slightly above the ball was much better than a roller situated directly behind the ball.

4 Software

4.1 State diagram

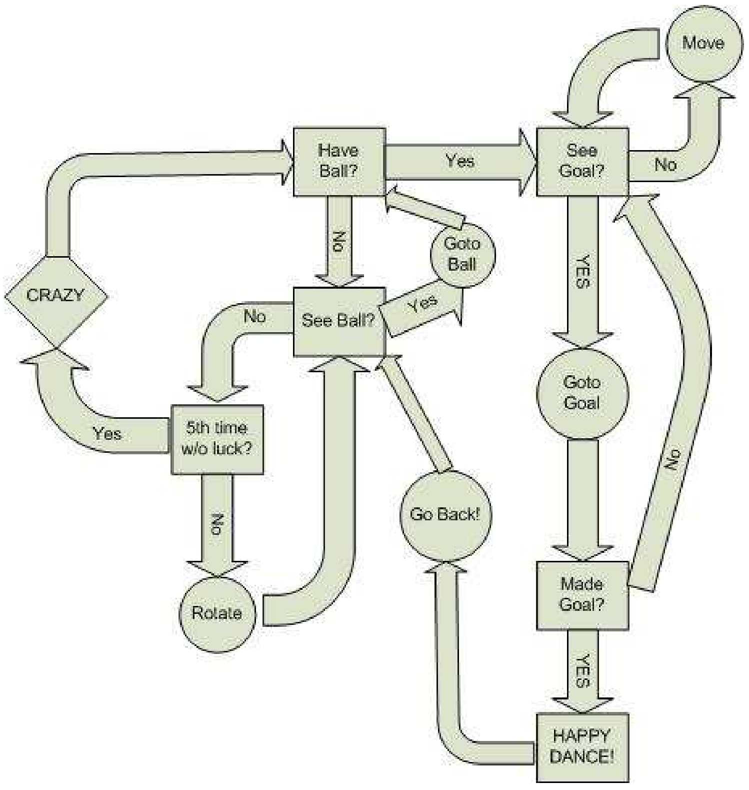


Figure 7: State-Diagram

4.2 Imageprocessing

For feature detection in the pictures that the robot took, we used the given code in the eyedemo program. The code checks the hue of the pixel it is examining to see if it corresponds to the feature we are trying to detect, if it does we do an additional check with RGB to ensure that we found what we are really looking for (i.e. the grass and the blue goal have similar hue, but a goal pixel has a higher blue than green value).

4.3 VW-Interface

The $V\omega$-interface is a high level interface for the motors, rather than having to issue instructions to both motors simultaneously the $V\Omega$-interface takes angles and distances as parameters and translates it into motor instructions. The robot rigorously uses the VWDriveCurve function in order to position itself correctly after getting a bumper signal, which is interpreted as running into a wall. A PI-regulator is used by the $V\Omega$-interface in order to interface with the motors.

5 Results

We had some bad luck because right before the competition our eyebot started to malfunction and crash when we tried to start it. This resulted in that we couldn't compete in the competition but after the competition was finished we borrowed another team's eyebot and we could demonstrate our robot. The robot successfully detected the ball from several initial ball positions. It drove to the ball and when the ball came in contact with the roller it gave it a backspin which made the ball stuck to the robot. After that it tried to locate the goal. Because of the limited range sight of the robot we had a different solution to find the goal when the goal wasn't in range. It was by clinging to the wall and eventually coming to one of the goal sides.

6 Conclusions

One experience we learned is don't over think your designs. You always find some small miss in the design when you actually build the robot. We ended up not using almost any of the aluminium sheets we got. It would have been better if we have designed the robot in iterations and gradually coming to the final design. If you don't you will always make a lot of last minute changes in the end which can not be properly tested and tuned.

Another experience is we have made us depend on one particular solution to get the robot to score goals. This was the roller, which if it didn't work properly we couldn't test robot dribbling with the ball. It would have helped if we had more then one solution to the problem because then you always will have a backup solution if one doesn't work. We once had a really good roller which we could turn and move the robot in any particular way. Unfortunately the roller wasn't robust enough so when it degraded in performance the robot's ability to hold the ball was none existence. This made it hard for the robot to score a goal.

7 References

http://www.elfa.se/pdf/75/07530389.pdf IR-sensor: IRM-8601S
http://www.elfa.se/pdf/75/07522519.pdf IR-diode: IR/HIR 333

http://www.csc.kth.se/utbildning/kth/kurser/DD2426/robot07/ Robotics and autonomous systems 2D1426

http://www.nada.kth.se/kurser/kth/2D1426/reports2006/Robinho.pdf

8 Appendix

```c
/**

 * Tina.c
 * Our Main-program

*/



#include "eyebot.h"
#include "trackdemo.h"
#include "imageproc.h"
#include "help.h"
#include <math.h>

#define SERVO_CAM SERVO11
#define SERVO_IR SERVO12


extern float wht_mult_red;
extern float wht_mult_green;
extern float wht_mult_blue;

void stallTest(BOOL);
BOOL haveBall(void);

VWHandle vw;
int direction;
int timesStalled;
int enemyGoalCol = 0;
colimage img;
int friendlyGoalCol = 0;
int main(void){
  /** Variabel-deklaration */
  int ball_pixelX = 0;
  int ball_pixelY = 0;
  int enemyGoal_pixelX = 0;
  int enemyGoal_pixelY = 0;
  int mainkey = 0;
  int trackkey = 0;
  int ball_error, ball_size, enemyGoal_size, enemyGoal_error;
  int val;
  float turnvelocity = 1;
  float velocity = 0.5;
  int enemyGoal = 0;
  PositionType start;
  int turns = 5; //Antal gånger han kommer leta innan han ger upp och antar att han inte hittar bollen..
  /** bilden att hantera */
  /** bilden till LCDn */
  image greyimg;

  image dithgreyimg;
  float t = 0.25*pi;
  SpeedType s;

  /** Slut på Variabel-deklaration*/

  /** Initieringsfasen*/
```

```
s.v = 0.2;
s.w = t;
timesStalled = 0;
direction = 1;
vw = VWInit(VW_DRIVE,1);
if(vw == 0)
  {
    LCDPutString("VWInit Error!\n");
    OSWait(200); return 1;
  }
VWSetPosition(vw,0,0,0);
VWGetPosition(vw,&start);
VWStartControl(vw,2,0.5,7,0.07);

LCDMode(SCROLLING|NOCURSOR);   /* init lcd */
LCDClear();
Init_Cam();              /* init camera */
CAMSet (FPS1_875, 0, 0); //om du har omega drive samtidigt, gör det här
ServoHandle cam_servo= SERVOInit(SERVO_CAM); // init servo

compute_rgb2hueLUT();
LCDClear();
ServoHandle ir_servo = SERVOInit(SERVO_IR);
ir_servo = ir_servo;
/** Slut på Initieringsfasen*/


while (mainkey != KEY4){
  LCDMenu("TRK","GOAL","SET","STP");
  mainkey = KEYRead();
  LCDSetPos(0, 10);
  LCDPrintf("%s\n", OSMachineName());
  /* print current colour values for ball and goal */

  CAMGetColFrame(&img, FALSE);
  IPColor2Grey(&img, &greyimg);
  LCDPutGraphic(&greyimg);

 OSWriteOutLatch(0,255, 0);

  switch(mainkey) {
  case KEY1:
      /* Startar Trackingprogrammet */
    LCDClear();
    while(trackkey != KEY4){

  //get_subsampled_colimage(&img);

  //LCDPrintf("(%d)", rounds++);
  LCDClear();
  BYTE latch0 = OSReadInLatch(0);
  LCDPrintf("latch0 %d ",latch0);

  latch0 = latch0 & 0x01;

  LCDPrintf("mask0 %d ",latch0);
```

```
// Bumper-program
/*
OSWait(10);
BYTE r = OSReadInLatch(0);
//LCDPrintf("%d", r);
LCDSetPos(1,1);
LCDPrintf("FV%d\n", (r & 0x10));
LCDPrintf("FH%d\n", (r & 0x40));
LCDPrintf("BV%d\n", (r & 0x80));
LCDPrintf("BH%d\n", (r & 0x20));
// Slut Bumper-program
*/
trackkey = KEYRead();

}//while
  break;

case KEY2:
  /*Här startar målgörar-läget*/
  LCDClear();
  LCDMenu("GOB","GOB","GOY","STP");
  val = KEYRead();
  while(val != KEY4){
val = KEYRead();
if(val == KEY2){

  enemyGoal = 1;
  enemyGoalCol = 200;
  friendlyGoalCol = 90;
  break;
}
else if(val == KEY1){

  enemyGoal = 1;
  enemyGoalCol = 200;
  friendlyGoalCol = 90;
  break;
}
else if(val == KEY3){

  enemyGoal = 2;
  enemyGoalCol = 90;
  friendlyGoalCol = 200;
  break;
}
  }
  LCDClear();
  LCDMenu(" "," "," ","STP");
  int counter = 0;
  int cykles_lost = 0;
  VWDriveStraight(vw, 0.8, velocity);
  stallTest(TRUE);
  while(KEYRead() != KEY4){
counter++;
LCDClear();
LCDSetPos(1,10);
LCDPrintf("C: %d", counter);
```

```
    get_subsampled_colimage(&img);
    float distans;
    float vridning;
    if((find_object(&ball_pixelY, &ball_pixelX, &ball_size, &ball_error, get_ball_col(), img)
 && ball_size_check(&ball_size) && find_ball(&ball_pixelY, &ball_pixelX, get_ball_col(), img
)) || (haveBall())){ //Letar boll
      if(KEYRead() == KEY4){
        break;
      }
      cykles_lost = 0;
      LCDSetPos(2,10);
      LCDPrintf("Hittat boll vid X%d Y%d\n", ball_pixelX, ball_pixelY);
      LCDPrintf("Size: %d\n", ball_size);
      goToCoordinate(ball_pixelX, ball_pixelY,  &distans, &vridning); //Beräknar dist och vrid
      VWDriveTurn(vw, vridning, turnvelocity);
      stallTest(TRUE);
      LCDSetPos(7,1);
      LCDPrintf("d:%f\n", distans);
      LCDPrintf("v:%f", vridning);
      VWDriveStraight(vw, distans, velocity);
        direction = 1;
        stallTest(TRUE);

      LCDSetPos(1,1);
      LCDPrintf("Har akt");

      get_subsampled_colimage(&img);
     // while(TRUE){
      while(haveBall()){ //Har boll med IR
        get_subsampled_colimage(&img);
        LCDClear();
        LCDSetPos(1,1);
        LCDPrintf("Har boll");
        cykles_lost = 0;
        if(find_object(&enemyGoal_pixelY, &enemyGoal_pixelX, &enemyGoal_size, &
enemyGoal_error, friendlyGoalCol, img)&& find_goal(&enemyGoal_pixelY, &enemyGoal_pixelX,
friendlyGoalCol, img) && goal_size_check(&enemyGoal_size) ){ //Leta mål
          VWDriveTurn(vw, pi, turnvelocity);
           stallTest(TRUE);
        }
        else if(find_object(&enemyGoal_pixelY, &enemyGoal_pixelX, &enemyGoal_size, &
enemyGoal_error, enemyGoalCol, img) && find_goal(&enemyGoal_pixelY, &enemyGoal_pixelX,
enemyGoalCol, img)){
          LCDSetPos(1,1);
          LCDPrintf("Hittat maal vid X%d Y%d\nStorlek: %d", enemyGoal_pixelX,
enemyGoal_pixelY, &enemyGoal_size);
          goToCoordinate(enemyGoal_pixelX, enemyGoal_pixelY, &distans, &vridning); //Gör mål,
          VWDriveTurn(vw, vridning,  turnvelocity);
            stallTest(TRUE);
          LCDSetPos(1,1);
          LCDPrintf("Go maal");
          cykles_lost = 0;

          LCDSetPos(7,1);
          LCDPrintf("d:%f\n", distans);
          LCDPrintf("v:%f", vridning);
          VWDriveStraight(vw, distans, velocity); //Kör framåt till målet
            direction = 1;
          stallTest(TRUE);

          if(Goal(&img, enemyGoalCol) && haveBall()){ //Mål!!!
                LCDSetPos(1,1);
                LCDPrintf("MAAAAAAL");
                playFriendlyTone(); //Målgest
                VWDriveStraight(vw, -1.5, velocity);
                direction = -1;
                OSWait(200);
                stallTest(TRUE);
          }
        }//Letar mål
        else{
          if (cykles_lost < 5*turns && cykles_lost%turns == 0){
        VWDriveCurve(vw, 0.215, -0.16, 0.20);
        direction = 1;
        //VWDriveTurn(vw, 3/2, turnvelocity/2);
        stallTest(TRUE);
          }
          else if(cykles_lost > 5*turns)
        {
           cykles_lost = 0;
           LCDPrintf("KAOS");
           AUTone(659, 500);
           OSWait(500/10);
           VWDriveStraight(vw, -0.8, velocity);
            direction = -1;
           stallTest(TRUE);
           VWDriveTurn(vw, 1, turnvelocity);
           stallTest(TRUE);
           VWDriveStraight(vw, -0.8, velocity);
            direction = -1;
           stallTest(TRUE);
           VWDriveTurn(vw, 2,  turnvelocity);
           stallTest(TRUE);
        }
        }//else Letar mål
      } //Fångar boll
      LCDPrintf("Tappat bort maal");
    } //Letar boll
    else{    //Om den inte hittar boll
      cykles_lost++;
      if (cykles_lost < 9*turns && cykles_lost%turns == 0){
        VWDriveTurn(vw, pi/4, turnvelocity);
        stallTest(TRUE);
      }
      else if(cykles_lost >= 9*turns)
      {
          cykles_lost = 0;
          AUTone(659, 500);
          OSWait(500/10);
          LCDPrintf("KAOS");
          VWDriveStraight(vw, 0.5, 0.3);
            direction = 1;
          stallTest(TRUE);
```

```
                /* VWDriveTurn(vw, 1, 0.1);
                   stallTest(TRUE);
          VWDriveStraight(vw, -0.3, 0.3);
                   direction = -1;
          stallTest(TRUE);
          VWDriveTurn(vw, 3-1,  0.1);
           stallTest(TRUE);
          */
          }
          else{
                 stallTest(TRUE);
          }
          //Snurra
          //och om det inte går... Kaos!
       }
          }  //målgörarläget
          break;

       case KEY3:  /** Konfigurera Världsbilden */
          if(CalibrateCamera(&img, img, &greyimg, &dithgreyimg) == 1){
       LCDPrintf("Kameran färdigkalibrerad\n");
          }else{
       LCDPrintf("Kameran ej kalibrerad\n");
          }
          mainkey = 0;
          break;

       case KEY4:
          //frigoer
          VWRelease(vw);

          SERVORelease(cam_servo);
          CAMRelease();
          break;

       default:
          break;
       }//switch

    }//while

    return 0;
}


/**
 * Vår egen hemmabyggda VWDriveWait(vw)
 * Om en av bumprarna trycks in medan den försöker nå
 * destinationen så kommer den avbryta och göra en undvikande manöver
*/

void stallTest(BOOL neverTested)
{
       BYTE bumpers;
       BYTE bumpers_last;
       bumpers_last=0;
       int time = 0;
```

```c
    while(!VWDriveDone(vw)){
    //FV = 0x10
    //FH = 0x40
    //BV = 0x80
    //BH = 0x20
    //IR = 0x01
    LCDSetPos(5,5);
    //LCDPrintf("S:%d\n", timesStalled);
    time++;
    //LCDPrintf("T:%d", time);
    bumpers = OSReadInLatch(0);
    //bumpers |= bumpers_last;

    if(VWStalled(vw) == 1 && !neverTested){
        LCDPrintf("Stalled");
        VWDriveStraight(vw, -0.3*direction, 0.7);

        //VWSetSpeed(vw, 0, 0);
        direction = -1*direction;
        timesStalled++;

    }
    if((bumpers & 0x10) == 0 && (bumpers & 0x40) == 0){
      LCDPrintf("skrap framat!\n");
      bumpers_last =1;
      AUBeep();
        get_subsampled_colimage(&img);
      if(Goal(&img, enemyGoalCol)&& FALSE)
      {
        playFriendlyTone();
        VWDriveStraight(vw, -1.5, 1.0);
            direction = -1;
            OSWait(200);
      }
      else{
        VWDriveCurve(vw, -0.35, -0.55, 0.15);
            direction = -1;
            neverTested = FALSE;
        }
       //stallTest(FALSE);
    }
    else if((bumpers & 0x10) == 0 && (bumpers & 0x80) == 0){
        bumpers_last =2;
      LCDPrintf("Skrap vanster!\n");
      AUBeep();
      VWDriveCurve(vw, 0.2, -2.0, 0.15);
        direction = 1;
        neverTested = FALSE;
        //stallTest(FALSE);
    }
    else if((bumpers & 0x80) == 0 && (bumpers & 0x20) == 0){
       bumpers_last =3;
      LCDPrintf("Skrap bakat!\n");

      AUBeep();
      VWDriveCurve(vw, +0.2, -0.5, 0.15);
        direction = 1;
        neverTested = FALSE;
      //stallTest(FALSE);
    }
    else if((bumpers & 0x40) == 0 && (bumpers & 0x20) == 0){
       bumpers_last =4;
      LCDPrintf("Skrap hoger!\n");
        AUBeep();
        VWDriveCurve(vw, 0.2, 0.4, 0.15);
        direction = -1;
        neverTested = FALSE;
      //stallTest(FALSE);
    }
    else if((bumpers & 0x10) == 0){
      LCDPrintf("Skrap FV!%d\n", VWStalled(vw));
      AUBeep();
       bumpers_last =5;

        get_subsampled_colimage(&img);
      if(Goal(&img, enemyGoalCol)&& FALSE)
      {
        playFriendlyTone();
        VWDriveStraight(vw, -1.5, 1.0);
            direction = -1;
            OSWait(200);
      }
      else{
          VWDriveCurve(vw, -0.2, -0.8, 0.15);
            direction = -1;
            neverTested = FALSE;
        }
      //stallTest(FALSE);
    }
    else if((bumpers & 0x40) == 0){
      LCDPrintf("Skrap FH!%d\n", VWStalled(vw));
        bumpers_last =6;
      AUBeep();
       bumpers_last =0;
      bumpers_last = 0x40;
        get_subsampled_colimage(&img);
      if(Goal(&img, enemyGoalCol) && FALSE)
      {
        playFriendlyTone();
        VWDriveStraight(vw, -1.5, 1.0);
            direction = -1;
            OSWait(200);
      }
      else{
        VWDriveCurve(vw, -0.11, -0.9, 0.15);
            direction = -1;
            neverTested = FALSE;
        }
      //stallTest(FALSE);
    }
    else if((bumpers & 0x80) == 0){
        bumpers_last =7;

      LCDPrintf("Skrap BV!%d\n", VWStalled(vw));
      AUBeep();
      VWDriveCurve(vw, +0.2, 2.0, 0.15);
        direction = 1;
        neverTested = FALSE;
      //stallTest(FALSE);
    }
    else if((bumpers & 0x20) == 0){
        bumpers_last =6;

      LCDPrintf("Skrap BH!%d\n", VWStalled(vw));
      AUBeep();
      VWDriveCurve(vw, 0.2, 0.3, 0.15);
        direction = 1;
        neverTested = FALSE;
      //stallTest(FALSE );
    }
  }
}


/**
 * Kollar om vi har bollen.
 * kollar om våran IR-stråle har brutits have något objekt
 * Nackdel: Kan inte avgöra om det är en motståndare eller en boll.
*/
BOOL haveBall(){
    int de = 0;
    for(de = 0; de<10;de++){
        if((OSReadInLatch(0) & 0x1) == 1)
            return TRUE;
    }
    return FALSE;
}

/***************************************************************************/
/** imageproc.h
   Contains image processing headers.

   author Birgit Graf,  UWA, 1998, modified 2000 (Mk3/4)
   author Thomas Braunl, UWA, 1998 (HSV)

   modified for RAS by Mattias Bratt
       modified more for Ras4 - Boten Anna by Rasa4 2007
*/
/***************************************************************************/

#include "eyebot.h"

#define NO_HUE 255

/* -----------= colours =-------------- */
#define BLACK   0
#define WHITE 15


/**
  Init camera.
```

```c
 */
void Init_Cam();

/** Change camera parameters.
   Changes brightness, hue and saturation in case default values
   aren't good enough (e.g. different light conditions). Function as
   in RoBiOS-Setup, but without camera-initialisation, works only for
   colour-camera.
 */
void set_cam_parameters();

/** Change parameters.
   Changes thresholds which are used to select ball/goal coloured
   pixels from others and size of ball/goal.
 */
void set_img_parameters();

/** Set whitepoint.
   Set  colour detected in middle of picture
   (mean value of 5x5 area around middle of picture).
   to white by defining multipliers for R, G and B
   to make their adjusted values equal at this point.
*/

/**************************************************************************/
/** Change RBG to HSV -- use hue only.
   Thomas Braunl, UWA 1998.

   param r,g,b rgb value of single pixel
   return hue for single RGB value
*/
/**************************************************************************/

int RGBtoHue(BYTE r, BYTE g, BYTE b);



void set_white_point(colimage img);

void reset_white_point();

/** correct colour using white point determined in set_white_point() **/

void correct_colour(int *r,int *g,int *b);


/** Compute values for the hue lookup table using
 ** values from RGBtuHUE, which compensates for
 ** chosen white point
 **/

void compute_rgb2hueLUT();

/** Set Ball colour.
   Set ball colour to colour detected in middle of picture
   (mean value of 5x5 area around middle of picture).
*/
```

```c
void set_ball_colour(colimage img);

void set_closeBallColour(colimage img);

int get_ball_col();

int get_closeBall_col();

/** Mark object.
   Convert colour picture into greyscale (for LCD output),
   mark goal position by drawing a vertical and horizontal white
   line through it. Ball and it's size is displayed by black lines.
 */

int get_enemyGoal_col();

void set_enemyGoalColour(colimage img);

void mark_object(image greyimg, int x_middle, int y_middle, int object_size);

/** Search for object.
   Search for reagions in rows, with consecutive matching pixels.
   Hue error must be below thresh at the endpoints, but between them
   max_lowq_seq consecutive pixels are allowed to match less strictly (thresh2)
   or have undefined hue (NO_HUE).
   max_bad_in_seq pixels in this sequence that do not match at all are
   also allowed.
  */

int find_object(int *row_middle, int *col_middle, int *object_size, int *mean_error,
        int colour, colimage img);

int find_objectRGB(int *row_middle, int *col_middle, int *object_size, int *mean_error,
        int colour, colimage img, int typ);

int find_goal(int *row_middle, int *col_middle, int colour, colimage img);

int find_ball(int *row_ball, int *col_ball, int colour, colimage img);

int ball_size_check(int *ball_size);

int goal_size_check(int *goal_size);

/* Get an image of 82x62 size subsampled from the full 176x144 image */
void get_subsampled_colimage(colimage *img);

/* Send colour image to PC over RS232
  do
  $ stty -F/dev/ttyS0 38400
  $ cat /dev/ttyS0 >bild.ppm
  on PC before sending
  view with
  $ gimp bild.ppm &
  (Try modifying the code for higher speed and see if that works)
*/
void sendRGBData(colimage img);
```

```
/* Send greyscale image to PC over RS232
  do
  $ stty -F/dev/ttyS0 38400
  $ cat /dev/ttyS0 >bild.ppm
  on PC before sending
  view with
  $ gimp bild.ppm &
*/
void sendGreyData(image img);


/***************************************************************************/
/** imageproc.c
   Contains image processing routines.

   author Birgit Graf,   UWA, 1998, modified 2000 (Mk3/4)
   author Thomas Braunl, UWA, 1998 (HSV)

   modified for RAS by Mattias Bratt
        modified more for Ras4 - Boten Anna by Rasa4 2007
*/
/***************************************************************************/

#include "imageproc.h"
#include "help.h"
#include "eyebot.h"
#include <stdio.h>

/** thresholds for comparison of colours to ballcolour */
int thresh = 20;  //5;
int thresh2 = 40;  //7;

/** white point multipliers for red gren blue**/
float wht_mult_red=1.119205;
float wht_mult_green=1.119205;
float wht_mult_blue=4.694444;

/** colour space lookup table from 3x6bit RGB to hue **/
// Size in memory is 2^(3x5)=32K=LUTres
#define LUTres 32
BYTE rgb2hueLUT[LUTres][LUTres][LUTres];
int LUTinitialised =FALSE;




/** ball colour */
int ballColourR = 48;//42; //red is default
int ballColourG = 48;//42; //red is default
int ballColourB = 48;//42; //red is default
int closeBallColourR = 72;
int closeBallColourG = 72;
int closeBallColourB = 72;
int enemyGoalColourR = 0;  //"odefinierad" från början
int enemyGoalColourG = 0;  //"odefinierad" från början
int enemyGoalColourB = 0;  //"odefinierad" från början
int BallColour = 45;
int yellow_Goal = 90;  //yellow is default
int blue_Goal = 200;  //blue is default
/***************************************************************************/
/** Init camera.


   Called by main().
*/
/***************************************************************************/

void Init_Cam()
{
   int camera;
   /*int bright, hue, sat;*/
/*  int i,x,y; */

   camera = CAMInit(NORMAL);
   LCDSetPos(4,4);
   LCDPrintf("c=%d", camera);
   if (camera < COLCAM)
     error("No colour camera!");
   else
   {
     if (camera == NOCAM)
       error("No camera!\n");
     else
       if (camera == INITERROR)
         error("CAMInit!\n");
   }
}

/***************************************************************************/
/** Change camera parameters.
   Changes brightness, hue and saturation in case default values
   aren't good enough (e.g. different light conditions). Function as
   in RoBiOS-Setup, but without camera-initialisation, works only for
   colour-camera.
*/
/***************************************************************************/

void set_cam_parameters ()
{
   image greyimg;
   colimage img;
   int fps,w,h;
   int end_proc = FALSE;

   LCDClear();
   LCDMenu("AUT","NAUT","","END");
   LCDPrintf("Camera param.:\n");

   while (!end_proc)
   {
     CAMGet(&fps, &w, &h);

     LCDSetPos(2, 0);
     LCDPrintf("FPS: %d\n", fps);
     LCDPrintf("FullWidth: %d\n", w);
```

```c
    LCDPrintf("FullHeight: %d\n", h);

    CAMGetColFrame(&img, FALSE);
    IPColor2Grey(&img, &greyimg);

    LCDPutGraphic (&greyimg);

    switch (KEYRead())
    {
    case KEY1:
      CAMMode(AUTOBRIGHTNESS);
       break;

    case KEY2:
      CAMMode(NOAUTOBRIGHTNESS);
      break;

    case KEY4:
      LCDClear(); end_proc = TRUE; break;

    default: break;
    }
  }
}


/***************************************************************************/
/** Change parameters for image processing.
  Changes thresholds which are used to select ball/goal coloured
  pixels from others and size of ball/goal.

*/
/***************************************************************************/

void set_img_parameters()
{
  int ind = 0;
  int end_proc = FALSE;

  LCDClear();
  LCDMenu("CHG", "NXT", " ", "END");

  while (!end_proc)
  {
    LCDSetPos(0, 0);
    LCDPrintf("Image parameters\n");

    LCDSetPos(2, 0);
    LCDPrintf("Hue Thresh\n");
    LCDSetPos(3, 0);
    LCDPrintf("Hue Thresh2\n");

    LCDSetChar(2 + ind, 15, '*');

    switch (KEYGet())
    {
    case KEY1:
      switch(ind)
      {
      case 0:
        thresh = set_iparam("Hue Thresh", 0, thresh, 100, 1);
        break;
      case 1:
        thresh2 = set_iparam("Thresh ball", thresh, thresh2, 100, 1);
        break;
       default: break;
      }

      LCDMenu("CHG", "NXT", " ", "END");
      break;

    case KEY2:
      LCDSetChar(2 + ind, 15, ' ');
      ind ++;

      if (ind > 1) ind = 0;

      break;

    case KEY4:
      LCDClear(); end_proc = TRUE; break;

    default: break;
    }
  }
}

//forward declaration
void correct_colourB(BYTE *r,BYTE *g,BYTE *b);

/****************************************************************************/
/** Change RBG to HSV -- use hue only.
   Thomas Braunl, UWA 1998.

   param r,g,b rgb value of single pixel
   return hue for single RGB value
*/
/****************************************************************************/

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

int RGBtoHue(BYTE r, BYTE g, BYTE b)
{
  BYTE hue /*,sat, val*/, delta, max, min;

  correct_colourB(&r,&g,&b);

  max   = MAX(r, MAX(g,b));
  min   = MIN(r, MIN(g,b));
  delta = max - min;
  hue =0;
  /* initialise hue*/
  /* val   = max;
  if (max != 0) sat = delta / max; else sat = 0;
  if (sat == 0) hue = NO_HUE;
 */
  if (2 * delta <= max) hue = NO_HUE;
  else {
    if (r == max) hue =  42 + 42*(g-b) / delta;  /* 1*42 */
    else if  (g == max) hue = 126 + 42*(b-r) / delta;  /* 3*42 */
    else if (b == max) hue = 210 + 42*(r-g) / delta;  /* 5*42 */
    /* now: hue is in range [0..252] */
  }
  return hue;
}



/***********************************************************************/
/** Set target colour.
  Set target colour to colour detected in middle of picture
  (mean value of 5x5 area around middle of picture).

  param img colour picture
  return mean colour of image middle and surroundings
*/
/***********************************************************************/

int set_colour(colimage img)
{
  int row, column;
  int count = 0;
  int my_hue, hue;

  my_hue = 0;

  for (row = imagerows/2 - 2; row < (imagerows/2 + 3); row ++)
    for (column = imagecolumns/2 - 2; column < imagecolumns/2 + 3; column ++)
    {
      hue = RGBtoHue(img[row][column][0], img[row][column][1],
          img[row][column][2]);
      if (hue != NO_HUE)
      {
        my_hue += hue;
        count ++;
      }
    }

  if (count != 0)
    my_hue /= count;

  LCDClear();
  LCDMenu("", "", "", "OK");
  LCDSetPos(7,0);
  LCDPrintf("HUE: %3d\n", my_hue);

  KEYWait(KEY4); LCDClear();

  return my_hue;
}
```

```c
/****************************************************************************/
/** Set whitepoint.
   Set  colour detected in middle of picture
   (mean value of 5x5 area around middle of picture).
   to white by defining multipliers for R, G and B
   to make their adjusted values equal at this point.

*/
/****************************************************************************/

void set_white_point(colimage img)
{
  int row, column;
  int count = 0;
  int r=0,g=0,b=0;

  for (row = imagerows/2 - 2; row < (imagerows/2 + 3); row ++)
    for (column = imagecolumns/2 - 2; column < imagecolumns/2 + 3; column ++)
    {
      r+=img[row][column][0];
      g+=img[row][column][1];
      b+=img[row][column][2];
      count ++;
    }

  if (count != 0){
    r /= count;
    g /= count;
    b /= count;
  }
  int max = MAX(r, MAX(g,b));

  wht_mult_red=((float)max)/r;
  wht_mult_green=((float)max)/g;
  wht_mult_blue=((float)max)/b;

  LCDClear();
  LCDMenu("", "", "", "OK");
  LCDSetPos(2,0);
  LCDPrintf("r: %d\n", r);
  LCDSetPos(3,0);
  LCDPrintf("g: %d\n", g);
  LCDSetPos(4,0);
  LCDPrintf("b: %d\n", b);

  KEYWait(KEY4); LCDClear();
}


void reset_white_point()
{
  wht_mult_red=wht_mult_green=wht_mult_blue=1.0;
  LCDClear();
  LCDMenu("", "", "", "OK");
  LCDSetPos(2,0);
  LCDPrintf("White point");
  LCDSetPos(3,0);
  LCDPrintf("reset");
  KEYWait(KEY4); LCDClear();
}

/** correct colour using white point determined in set_white_point() **/
void correct_colour(int *r,int *g,int *b){
  int cr=wht_mult_red*(*r);
  int cg=wht_mult_green*(*g);
  int cb=wht_mult_blue*(*b);
  *r=*g=*b=255;
  if(cr<255) *r=cr;
  if(cg<255) *g=cg;
  if(cb<255) *b=cb;
}
void correct_colourB(BYTE *r,BYTE *g,BYTE *b){
  int cr=wht_mult_red*(*r);
  int cg=wht_mult_green*(*g);
  int cb=wht_mult_blue*(*b);
  *r=*g=*b=255;
  if(cr<255) *r=cr;
  if(cg<255) *g=cg;
  if(cb<255) *b=cb;
}

/** Compute values for the hue lookup table using
 ** values from RGBtuHUE, which compensates for
 ** chosen white point so that the LUT will also
 ** do so.
 **/
void compute_rgb2hueLUT(){
  BYTE r,g,b;
  LCDClear();
  LCDMenu("", "", "", "ABRT");
  LCDSetPos(2,0);
  LCDPrintf("Computing colour");
  LCDSetPos(3,0);
  LCDPrintf("lookup table");
  LCDSetPos(5,0);
  LCDPrintf("   of %d rows",LUTres);
  LCDSetPos(6,0);
  LCDPrintf("completed");

  for(r=0;r<LUTres;r++) {
    LCDSetPos(5,0);
    LCDPrintf("%2d",r);
    if(KEYRead()==KEY4) return;
    for(g=0;g<LUTres;g++){
      for(b=0;b<LUTres;b++){
    rgb2hueLUT[r][g][b]=RGBtoHue(r*(256/LUTres),g*(256/LUTres),b*(256/LUTres));
      }
    }
  }
  LUTinitialised=TRUE;
}
```

```
/** Use fast lookup table method to find the hue
 ** from rgb values
 **/
int rgb2hue(int r, int g, int b){
  if(LUTinitialised)
    return rgb2hueLUT[r/(256/LUTres)][g/(256/LUTres)][b/(256/LUTres)];
  else
    return RGBtoHue(r,g,b);
}


/***********************************************************************/
/** Set ??? colour.
     Sets the ???'s colour values
*/
/***********************************************************************/
void set_ball_colour(colimage img)
{
  BallColour = set_colour(img);
}

void set_enemyGoalColour(colimage img)
{
  int row, column;
  int count = 0;
  int my_hue;
  int r = 0, g = 0, b = 0;
  my_hue = 0;

  for (row = 0; row < 5; row ++)
    for (column = imagecolumns/2-2; column < imagecolumns/2 + 3; column ++)
    {
      r += img[row][column][0];
      g += img[row][column][1];
      b += img[row][column][2];
      count++;
    }

  if (count != 0){
    r /= count;
    g /= count;
    b /= count;
  }

  LCDMenu("", "", "", "OK");
  LCDPrintf("%d/%d/%d\n", r, g, b);

  KEYWait(KEY4);
  enemyGoalColourR = r;
  enemyGoalColourG = g;
  enemyGoalColourB = g;
}

void set_closeBallColour(colimage img){
  int row, column;
  int count = 0;
  int my_hue;
```

```c
  int r = 0, g = 0, b = 0;
  my_hue = 0;

  for (row = imagerows - 5; row < imagerows; row ++)
    for (column = imagecolumns/2 - 2; column < imagecolumns/2 + 3; column ++)
    {
      r += img[row][column][0];
      g += img[row][column][1];
      b += img[row][column][2];
      count++;
    }

  if (count != 0){
    r /= count;
    g /= count;
    b /= count;
  }

  LCDMenu("", "", "", "OK");
  LCDPrintf("%d/%d/%d\n", r, g, b);

  KEYWait(KEY4);
  closeBallColourR = r;
  closeBallColourG = g;
  closeBallColourB = g;
}

/***************************************************************************/
/** Mark object.
   Convert colour picture into greyscale (for LCD output),
   mark ball/goal position by drawing a vertical and horizontal white
   line through it. Ball and it's size is displayed by black lines.

   Called by find_object().

   param row_middle, col_middle pixel-coordinates of object
   param object_size width of marking
*/
/***************************************************************************/

void mark_object(image greyimg, int row_middle, int col_middle, int object_size)
{
  int row, column;
  if (row_middle<1 || row_middle>=imagerows-1 || col_middle<1 || col_middle>=imagecolumns-1)
    error("mark_object: ");
  /* mark object position: WHITE */
  for (row = 1; row < imagerows; row ++)
    greyimg[row][col_middle] = WHITE;

  for (column = 1; column < imagecolumns; column ++)
    greyimg[row_middle][column] = WHITE;


  /* mark object size: BLACK */
  int col1=col_middle - object_size/2;
  int col2=col_middle + object_size/2;
  for (row = row_middle - 7; row < row_middle + 8; row ++)
  {
    if(row<1||row>=imagerows-1) continue;
    if(col1>0 && col1<imagecolumns-1) greyimg[row][col1] = BLACK;
    if(col2>0 && col2<imagecolumns-1) greyimg[row][col2] = BLACK;
  }

  for (column = col_middle - object_size / 2 + 1; column < col_middle +object_size / 2;
column ++)
    if(column>0 && column<imagecolumns-1)
      greyimg[row_middle][column] = BLACK;
}




/***************************************************************************/
/** Search for coloured object.
   Search for reagions in rows, with consecutive matching pixels.
   Hue error must be below thresh at the endpoints, but between them
   max_lowq_seq consecutive pixels are allowed to match less strictly (thresh2)
   or have undefined hue (NO_HUE).
   max_bad_in_seq pixels in this sequence that do not match at all are
   also allowed.

   param *row_middle, *column_middle pointers to pixel-coordinates of middle
   of object.
   param *object_size pointer to object's size in number of columns
   param *mean_error mean of the absolute hue error over the sequence
   param colour desired colour of searched object
   global thresh threshold for colour-comparison
   global thresh2 less strict (larger) threshold for colour-comparison
   return TRUE or FALSE if ball/goal was found or not
   return changed ball/goal coordinates, error  and size in referenced
   variables
*/
/***************************************************************************/



#define max_lowq_seq 4 //number of low quality (thresh2) matching pixels allowed in sequence between good matches
#define max_bad_in_seq 2 //number of unmatching pixels allowed between good matches
#define min_seq_length 3 //minimum length of sequence == minimum size of object tha can be detected

int find_object(int *row_middle, int *col_middle, int *object_size, int *mean_error,
    int colour, colimage img) {
  // use short (16 bits) and BYTE (8bits unsigned) instead of int to save time
  image obj;
  int testi = 0, testj = 0;
  for(testi = 1;testi<61;testi++){
    for(testj=1;testj<81;testj++){
      obj[testi][testj] = (BYTE)0;
    }
  }

  BYTE row, column;
  BYTE count=0;                /* no.of pixels with valid hue */
  BYTE best_row=0, best_column=0, best_count=0, best_mean_diff=255;//best region found
  BYTE huediff;
  short sumdiff=0;
  short sumlowqdiff=0;
  short meandiff;
  BYTE hue;
  BYTE lowq_count=0;  /*number of low quality matches since last good match*/
  BYTE bad_count=0;  /*number of pixels not matching since last good match*/
  BYTE sequence_ended=FALSE;

  for (row = 1; row < imagerows-1; row ++) {
    for (column=1;column < imagecolumns-1;column++){
      hue=rgb2hue(img[row][column][0],img[row][column][1],img[row][column][2]);
      if(hue!=NO_HUE){
    huediff=colour-hue; if(huediff>127) huediff=-huediff; // abs(colour-hue)
        //if the diff is larger than 126 than the other
        //way round is shorter because max hue value is 253 (e.g. hue 2 is close to 252)
    if (huediff > 126) huediff = 253 - huediff;
      }
      else huediff=NO_HUE;

      if(huediff<=thresh) {
    obj[row][column] = (BYTE)255;
    count++;
    sumdiff+=huediff;
    sumlowqdiff=0;
    lowq_count=0;
    bad_count=0;
      }
      else if(count>0) {
    if(lowq_count<max_lowq_seq) {
      if(huediff<= thresh2) {
        obj[row][column] = (BYTE)128;
        count++;
        lowq_count++;
        sumdiff+=huediff;
        sumlowqdiff+=huediff;
      }
      else if(huediff==NO_HUE) {
        count++;
        lowq_count++;
        sumdiff+=thresh2;
        sumlowqdiff+=thresh2;
      }
      else if(bad_count<max_bad_in_seq){
        count++;
        lowq_count++;
        bad_count++;
        sumdiff+=huediff;
        sumlowqdiff+=huediff;
      }
      else sequence_ended=TRUE;
    }
    else  sequence_ended=TRUE;
    if(sequence_ended) {
      sequence_ended=FALSE;
      count-=lowq_count;
      meandiff=(sumdiff-sumlowqdiff)/count;
```

```
      if(count>min_seq_length&&meandiff<best_mean_diff) {
        best_row=row;
        best_column=column-count-lowq_count;
        best_count=count;
        best_mean_diff=meandiff;
      }
      count=0;
      sumdiff=0;
    }
      }
    }
    if(count!=0){  // sequence ended at end of row
      count-=lowq_count;
      meandiff=(sumdiff-sumlowqdiff)/count;
      if(count>min_seq_length&&meandiff<best_mean_diff) {
    best_row=row;
    best_column=column-count-lowq_count;
    best_count=count;
    best_mean_diff=meandiff;
      }
      count=0;
      sumdiff=0;
    }
  }
  *row_middle=best_row;
  *col_middle=best_column+best_count/2;
  *object_size=best_count;
  *mean_error=best_mean_diff;

  image dithed;
  IPDither(&obj,&dithed);
  LCDPutGraphic(&dithed);

  if(best_count>min_seq_length) return TRUE;
  else return FALSE;
}

/**
* Match the colors instead of the HUE
*/
BOOL colorMatch(int row, int col, colimage img, int thr, int r, int g, int b){
  if(img[row][col][0] < r+thr &&img[row][col][0] > r-thr  &&img[row][col][1] < g+thr &&img[
row][col][1] > g-thr  &&img[row][col][2] < b+thr &&img[row][col][2] > b-thr){
    return TRUE;
  }
  return FALSE;
}


/**
 * An  advanced find_object that tracks color's instead of HUE
*/
int find_objectRGB(int *row_middle, int *col_middle, int *object_size, int *mean_error,
           int colour, colimage img, int typ) {
  image obj;
  int testi = 0, testj = 0;
  for(testi = 1;testi<61;testi++){
    for(testj=1;testj<81;testj++){
      obj[testi][testj] = (BYTE)0;
    }
  }
  // use short (16 bits) and BYTE (8bits unsigned) instead of int to save time
  int startpix = 1;
  int endpix = imagerows-1;
  int foR=0, foG=0, foB=0;
  switch(typ){
  case 1:
    foR = enemyGoalColourR;
    foG = enemyGoalColourG;
    foB = enemyGoalColourB;
    endpix -= 20;
    break;
  case 2:
    break;
  case 3:
    foR = ballColourR;
    foG = ballColourG;
    foB = ballColourB;
    break;
  case 4:
    break;
  case 13:
    startpix = 48;
    typ = 3;
    foR = closeBallColourR;
    foG = closeBallColourG;
    foB = closeBallColourB;
    break;
  }

  BYTE row, column;
  BYTE count=0;                /* no.of pixels with valid hue */
  BYTE best_row=0, best_column=0, best_count=0;//best region found

  short sumdiff=0;
  short sumlowqdiff=0;
  short meandiff;
  BYTE lowq_count=0; /*number of low quality matches since last good match*/
  BYTE bad_count=0; /*number of pixels not matching since last good match*/
  BYTE sequence_ended=FALSE;

  for (row = startpix; row < endpix; row ++) {
    for (column=1;column < imagecolumns-1;column++){
      if(colorMatch(row, column, img, thresh, foR, foG, foB)) {
    obj[row][column] = (BYTE)255;
    count++;
    sumlowqdiff=0;
    lowq_count=0;
    bad_count=0;
      }
      else if(count>0) {
    if(lowq_count<max_lowq_seq) {
      if(colorMatch(row, column, img, thresh2, foR, foG, foB)) {
        obj[row][column] = (BYTE)128;
        count++;
        lowq_count++;
      }

      else if(bad_count<max_bad_in_seq){
        count++;
        lowq_count++;
        bad_count++;
      }
      else sequence_ended=TRUE;
    }
    else  sequence_ended=TRUE;
    if(sequence_ended) {
      sequence_ended=FALSE;
      count-=lowq_count;
      meandiff=(sumdiff-sumlowqdiff)/count;
      if(count>min_seq_length && count > best_count) {
        best_row=row;
        best_column=column-count-lowq_count;
        best_count=count;
      }
      count=0;
    }
      }
    }
    if(count!=0){ // sequence ended at end of row
      count-=lowq_count;
      if(count>min_seq_length && count > best_count) {
    best_row=row;
    best_column=column-count-lowq_count;
    best_count=count;
      }
      count=0;
    }
  }
  *row_middle=best_row;
  *col_middle=best_column+best_count/2;
  *object_size=best_count;
  *mean_error=0;
  BOOL target_ok = FALSE;
  switch(typ){
  case 1: //Blue goal
    LCDPrintf("%d %d %d Y%d X%d", img[*row_middle][*col_middle][0], img[*row_middle][*
col_middle][1], img[*row_middle][*col_middle][2], *row_middle, *col_middle);
    if(img[*row_middle][*col_middle][0] < img[*row_middle][*col_middle][1] && img[*row_middle
][*col_middle][1] < img[*row_middle][*col_middle][2]){
      target_ok = TRUE;
    }
    break;
  case 2: //Yellow goal
    LCDPrintf("%d %d %d %d", img[*row_middle][*col_middle][0], img[*row_middle][*col_middle][
1], img[*row_middle][*col_middle][2], best_count);
    if(img[*row_middle][*col_middle][0] > img[*row_middle][*col_middle][1] && img[*row_middle
][*col_middle][1] < img[*row_middle][*col_middle][2]){
      target_ok = TRUE;
    }
    break;
  case 3: //Ball
    //if(img[*row_middle][*col_middle][0] > 200 && img[*row_middle][*col_middle][1] < 130 && img[*row_middle][*col_middle][2] < 50){
      target_ok = TRUE;
      //}
    break;
  case 4: //Carpet
    if(img[*row_middle][*col_middle][0] < 100 && img[*row_middle][*col_middle][1] > 200 &&
img[*row_middle][*col_middle][2] < 200){
      target_ok = TRUE;
    }
    break;
  }
  image dithed;
  IPDither(&obj,&dithed);
  LCDPutGraphic(&dithed);
  LCDSetPos(1,10);
  LCDPrintf("Typ: %d", typ);
  if(best_count>min_seq_length && target_ok) {
    LCDPrintf("TRUE %d", best_count);
    return TRUE;
  }
  else {
    LCDPrintf("FALSE %d", best_count);
    return FALSE;
  }
}

/**
 * A final test for deciding if the found object actually is a goal.
*/
int find_goal(int *row_goal, int *col_goal, int colour, colimage img) {
  if(colour>150) {
    if(img[*row_goal][*col_goal][2] > img[*row_goal][*col_goal][1]){
      LCDPrintf("Malet ar blatt");
      return TRUE;
    }
    LCDPrintf("Malet ar inte blatt");
    return FALSE;
  }
  else if(colour < 120){
    if(1){
        return TRUE;
    }
    LCDPrintf("Malet ar inte gult");
    return FALSE;
  }
  LCDPrintf("Malet ar inte i ratt farg");
  return FALSE;
}


/**
 * A final test for deciding if the found object actually is a ball.
*/
int find_ball(int *row_ball, int *col_ball, int colour, colimage img) {
    if(img[*row_ball][*col_ball][0] > 220){
        LCDPrintf("Bollen ar en boll");
        return TRUE;
    }
    LCDPrintf("Bollen ar ett mal");
    return FALSE;
}

int ball_size_check(int *ball_size){
    if(*ball_size < 15){
        return TRUE;
    }
    return FALSE;
}

int goal_size_check(int *goal_size){
    if(*goal_size > 10){
        return TRUE;
    }
    return FALSE;
}
/****************************************************************************/
/** Get ball colour.
   Get current values for ball colour.

   Called by main().
*/
/****************************************************************************/


void get_subsampled_colimage(colimage *img) {
  typedef BYTE bigcolimage[144][176][3];
  bigcolimage big_img;
  CAMGetFrameRGB((BYTE *)big_img);
  BYTE row,col,bigrow,bigcol;
  for(row=0,bigrow=10;row<imagerows;row++,bigrow+=2) {
    for(col=0,bigcol=6;col<imagecolumns;col++,bigcol+=2) {
      (*img)[row][col][0]=big_img[bigrow][bigcol][0];
      (*img)[row][col][1]=big_img[bigrow][bigcol][1];
      (*img)[row][col][2]=big_img[bigrow][bigcol][2];
    }
  }
}

void sendString(char* s )
{
  while (*s)
  {
    OSSendRS232( s, SERIAL1);
    s++;
  }
}

void sendRGBData(colimage img)
{
  int i,j;
  char temp[100];
  OSInitRS232( SER38400, NONE, SERIAL1 );
  sprintf(temp,
      "P3\n"
      "82 62\n"
      "255\n");
  sendString(temp);
  for(i=0, j=0; i < imagerows*imagecolumns; i++, j+=3)
  {
    BYTE r,g,b;
    r=((BYTE *)img)[j];
    g=((BYTE *)img)[j+1];
    b=((BYTE *)img)[j+2];
    correct_colourB(&r,&g,&b);
    sprintf(temp,"%u %u %u \n",r,g,b);
    if ((i % imagecolumns) == 0)
      {
    LCDClear();
    LCDPrintf(" %4d of %4d\n", i, imagerows*imagecolumns );
      }
    sendString(temp);
  }
  /* send RS232 termination character */
  sendString("\x04\n");
}

void sendGreyData(image img)
{
  int i,j;
  char temp[100];
  OSInitRS232( SER38400, NONE, SERIAL1 );

  sprintf(temp,
      "P2\n"
      "82 62\n"
      "15\n");
  sendString(temp);
  for(i=0, j=0; i < imagerows*imagecolumns; i++)
  {
    sprintf(temp,"%u\n",((BYTE *)img)[i]);
    if ((i % imagecolumns) == 0)
      {
    LCDClear();
    LCDPrintf(" %4d of %4d\n", i, imagerows*imagecolumns );
      }
    sendString(temp);
  }
  /* send RS232 termination character */
  sendString("\x04\n");
}

int get_ball_col()
{
    return BallColour;
}

/**

 * help.h
 * Our small acessory-header
```

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "eyebot.h"
#include "imageproc.h"


#define TRUE 1
#define FALSE 0


/** Error message.
  Print error message and end program.
*/
void error(char *str);

/** Calibrates the camera */
int CalibrateCamera(colimage*, colimage, image*, image*);

/** Set Parameter (int) over keyboard */
int set_iparam(char text[], int minp, int start, int maxp, int inc);

float pi;

float PixeltoRadian(int);
int PixeltoDegree(int);
void goToCoordinate(int, int, float*, float*); //Åker till boll
BOOL capturedBall(colimage*); //Greppa boll
BOOL findEnemyGoal(int*, int*); //Leta mål
BOOL Goal(colimage* img, int enemyGoal); //Om den är i mål med boll
void playFriendlyTone(); //Målgest
//void goToFriendlyGoal();//Åk hem och börja om

//void stallTest(VWHandle*, BumpHandle, BumpHandle, BumpHandle, BumpHandle);

/**

 * help.c
 * Our small acessory-file

*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "eyebot.h"
#include "imageproc.h"

float pi = 3.14159265358979;

/**
 * Översätter pixlar till vinkeln i radianer.
*/
float PixeltoRadian(int columns){
```

```c
  float scale;
  scale = -3.14/180;
  return scale*columns;
}

/**
 * Översätter pixlar till vinkeln i grader.
*/
int  PixeltoDegree(int columns){
  float scale;
  scale = 1;
  return (int)scale*columns;
}

/**
 * Ger distansen och vinkeln till pixeln.
*/
void goToCoordinate(int X_togo, int Y_togo, float* dist, float* vrid){
  //Höjd över marken = 100mm
  //Hypotenusan = 300mm
  //Ger: Vinkeln = 19,471220634490691369245999339962 grader
  //             = 0,33983690945412193709639251339176 radianer
  // Med mätningarna för första pixeln i jämförelse med den mellersta,
  //vid linjärt samband, fås:
  // y=kx+m ger k = 6.43 och m = 83.57
  //float k = 6.43;
  float k = 7.4;
  float m = 83.57;
  float d = 60.00; //distans från kameran till rollern.
  if(fabs(X_togo-40) < 30){
    *dist = ((62-Y_togo)*k+m-d)/1000;
  }
  else{
    *dist = 0;
  }
  *vrid = PixeltoRadian(X_togo-40);
  if(fabs(*vrid) < pi/16){
        *vrid = 0;
  }
}

/**
 * En funktion som kollar huruvida vi är i fiendemålet samt har boll med oss
 * returnerar sant då
*/
BOOL Goal(colimage* img, int enemyGoal){ //Om den är i mål med boll
    int pixelX, pixelY, enemygoal_error, enemygoal_size;
    if(find_object(&pixelX, &pixelY, &enemygoal_size, &enemygoal_error, enemyGoal, *img)){
        if(enemygoal_size>40){
            return TRUE;
        }
    }
    return FALSE;
}

/**
 * Vår Vinstmelodi!
```

```
*/
void playFriendlyTone(){ //Målgestnorm
  int norm = 500; //, snabb = 300, XS = 100;
  AUTone(659, norm);
  OSWait(norm/10);
  AUTone(728, norm);
  OSWait(norm/10);
  AUTone(528, norm);
  OSWait(norm/10);
  AUTone(659, norm);
  OSWait(norm/10);
  AUTone(728, norm);
  OSWait(norm/10);
  AUTone(528, norm);
  OSWait(norm/10);
}


/***********************************************************************/
/** Error message.
  Print error message and end program.

  param str string, which should be printed
*/
/***********************************************************************/

void error(char str[])
{
  LCDPrintf("ERROR: %s\n", str);
  OSWait(200);
  OSExit(0);
}



float fsign(float number)
{
  if (number < 0.0)
    return -1.0;
  else
    if (number > 0.0)
      return 1.0;
    else
      return 0.0;
}


/***********************************************************************/
/** Set Parameter over keyboard (float).

  Could be called by main().

  param text[] name of parameter
  param minp,maxp minimum and maximum value of parameter
  param start start value for parameter
  param inc step to increment parameter
*/
```

```
/***************************************************************************/

float set_fparam(char text[], float minp, float start, float maxp, float inc)
{
  float val;
  int   done = FALSE;

  LCDClear();
  LCDPrintf("Set Parameter\n%s\n", text);

  LCDMenu("+", "-", " ","OK");
  val = start;
  do
  {
    LCDSetPos(5, 0);
    LCDPrintf("%f\n", val);
    switch(KEYGet())
    {
    case KEY1:
      if (val < maxp)
    val += inc;
      break;
    case KEY2:
      if (val > minp)
    val -= inc;
      break;
    case KEY4:
      done = TRUE;
    }
  } while (!done);

  LCDClear();
  return val;
}



/***************************************************************************/
/** Set Parameter over keyboard (int).

  Called by main().

  param text[] name of parameter
  param minp,maxp minimum and maximum value of parameter
  param start start value for parameter
  param inc step to increment parameter
*/
/***************************************************************************/

int set_iparam(char text[], int minp, int start, int maxp, int inc)
{
  int val;
  int done = FALSE;

  LCDClear();
  LCDPrintf("Set Parameter\n%s\n", text);
  LCDMenu("+", "-", " ","OK");
  val = start;
  do
  {
    LCDSetPos(5, 0);
    LCDPrintf("%3d\n", val);
    switch(KEYGet())
    {
    case KEY1:
      if (val <maxp)
    val += inc;
      break;
    case KEY2:
      if (val > minp)
    val -= inc;
      break;
    case KEY4:
      done = TRUE;
    }
  } while (!done);

  LCDClear();
  return val;
}



/**
 * Calbrates the camera.
*/
int CalibrateCamera(colimage *img, colimage nybild, image *greyimg, image *dithgreyimg){
  // Lägg till färginställning för motståndarmål
  int exit_settings = FALSE;
  int exit_settings2 = FALSE;
  int exit_colour = FALSE;
  LCDClear();
  do {
    LCDMenu("COL", "", "...", "END");
    CAMGetColFrame(img, FALSE);
    IPColor2Grey(img, greyimg);
    LCDPutGraphic(greyimg);
    int pelle = 0;
    pelle = KEYRead();
    switch(pelle) {
    case KEY1:  /* SET-->COL: set colour values */
      LCDClear();
      do {
    int r,g,b;
    LCDMenu("BALL", "WHT", "RSTW", " END");
    CAMGetColFrame(img, FALSE);
    IPColor2Grey(img,greyimg);
    IPDither(greyimg,dithgreyimg);
    // draw a crosshair over the whole picture
    mark_object(*dithgreyimg, imagerows / 2 - 1,  imagecolumns / 2 - 1, 0);
    LCDPutGraphic(dithgreyimg);
    LCDPrintf("test");
    r=nybild[imagerows/2][imagecolumns/2][0];
    g=nybild[imagerows/2][imagecolumns/2][1];
    b=nybild[imagerows/2][imagecolumns/2][2];
    // print middle point hue and target hue
    // print white point-corrected RGB values
    correct_colour(&r,&g,&b);
    LCDSetPos(4, 10);
    LCDPrintf("r:%3d\n",r);
    LCDSetPos(5, 10);
    LCDPrintf("g:%3d\n",g);
    LCDSetPos(6, 10);
    LCDPrintf("b:%3d\n",b);
    int nyckeln = 0;
    nyckeln = KEYRead();
    switch(nyckeln) {
    case KEY1: // BALL
      LCDClear();
      LCDMenu("BALL", "", "", "");
      while(KEYRead() != KEY1){
        CAMGetColFrame(img, FALSE);
        IPColor2Grey(img, greyimg);
        IPDither(greyimg,dithgreyimg);
        LCDPutGraphic(dithgreyimg);
      }
      set_ball_colour(nybild);
      LCDSetPos(1,10);
      LCDPrintf("Ball:, %d\n", get_ball_col());
      LCDSetPos(2,10);
      LCDMenu("CBC","","","");
      while(KEYRead() != KEY1){
        CAMGetColFrame(img, FALSE);
        IPColor2Grey(img, greyimg);
        IPDither(greyimg,dithgreyimg);
        LCDPutGraphic(dithgreyimg);
      }
      set_closeBallColour(nybild);
      LCDSetPos(3,10);
      LCDMenu("EGB","","","");
      while(KEYRead() != KEY1){
        CAMGetColFrame(img, FALSE);
        IPColor2Grey(img, greyimg);
        IPDither(greyimg,dithgreyimg);
        LCDPutGraphic(dithgreyimg);
      }
      set_enemyGoalColour(nybild);
      break;
    case KEY2: //WHT
      /* Send image to PC over RS232 */
      set_white_point(nybild);
      break;
    case KEY3: //RSTW
      reset_white_point(nybild);
      break;
    case KEY4: //END
      exit_colour = TRUE; // end menu
      // But first offer to calculate new hue lookup table
      LCDClear();
      LCDMenu("NO", "NO", "NO", " YES");
      LCDSetPos(3, 0);
      LCDPrintf("Save in colour");
      LCDSetPos(4, 0);
      LCDPrintf("lookup table?");
      int pressed_key;
      while((pressed_key=KEYRead())==0) continue;
      if(pressed_key==KEY4) {
        compute_rgb2hueLUT();
      }
      break;
    default: break;
    }
      } while (!exit_colour);
      LCDClear();
      exit_colour = FALSE;
      break;

    case KEY3: /* SET-->...: other settings */
      LCDClear();
      do {
    LCDMenu("CAM", "IMG", "", "END");
    CAMGetColFrame(img, FALSE);
    IPColor2Grey(img, greyimg);
    LCDPutGraphic(greyimg);
    switch(KEYRead()) {
    case KEY1: set_cam_parameters();
      break;          /* CAM: set camera values */
    case KEY2: set_img_parameters();
      break;          /* IMG: set image parameters */
    case KEY3:
      break;
    case KEY4: exit_settings2 = TRUE;
      break;          /* END */
    default: break;
    }
      } while (!exit_settings2);

      LCDClear();
      exit_settings2 = FALSE;
      break;

    case KEY4: exit_settings = TRUE; break; /* SET-->END */
    default: break;
    }
  } while (!exit_settings);
  LCDClear();
  exit_settings = FALSE;
  return 1;
}
```