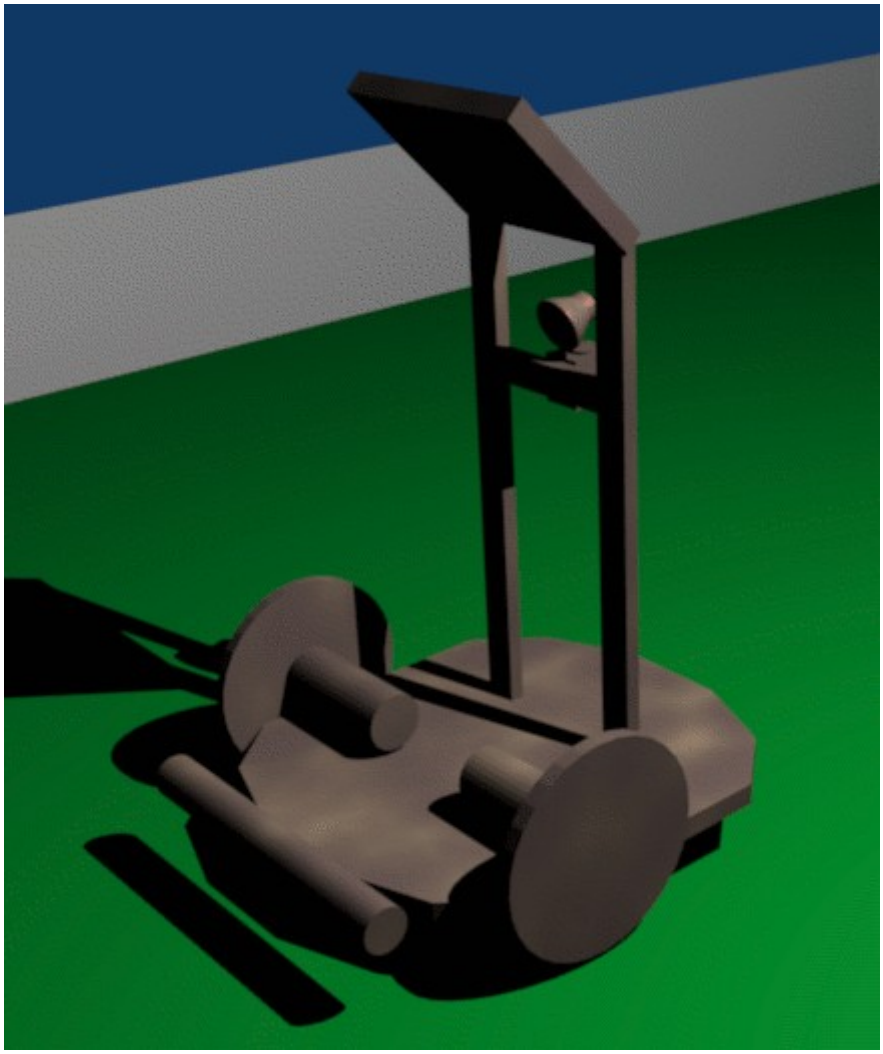
Kungl. Tekniska Högskolan

Eyebot-robot



Datum: 2007-06-15
Författare: Anders Alm
Kursansvarig: Patric Jensfelt
Kurs: robot07

Innehållsförteckning

Abstract

The purpose of this project was to construct an Eyebot robot which can score goals on a soccer field. One of the main problems was to attend to the locomotion issues that appeared when an encoder was broken.

The author experienced problems with the stability of the on-board autobrightness functionality, which introduced some misleading classification results, before this problem was discovered.

This implementation uses a scanner class and two classifiers, based on support vector machines, to classify and locate objects. The classifiers proved to be quite accurate.

Due to the fairly slow processor power and the huge amount of support vectors the chosen SVM implementation produced, only linear kernels were practically suitable.

Inledning

Merparten av arbetet i kursen *Robotik och Autonoma System* går ut på att konstruera en robot som ska försöka göra mål med en härligt rödorange golfboll på en liten fotbollsplan.

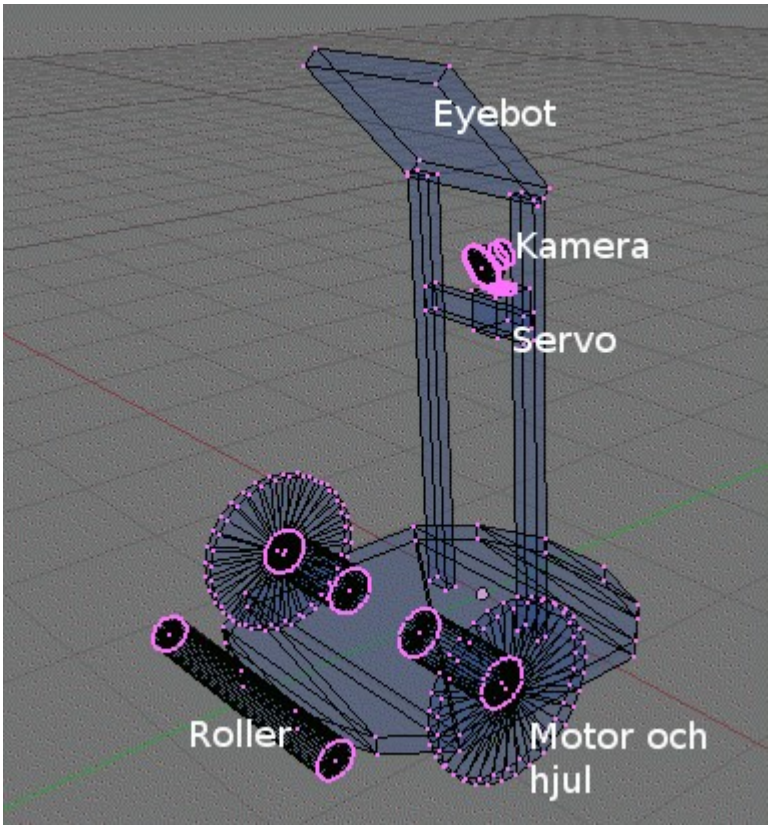
Då arbetet med roboten, av olika anledningar, var långt från färdigt på tävlingsdagen blev den smidigaste lösningen för alla involverade projektmedlemmar att författa individuella rapporter.

Roboten

Hårdvara

Robotens hårdvara[1] består av en Eyebot utrustad med 35 MHz 32-bitars Motorola 68332 processor, 2 MB RAM, en färgkamera med maxupplösningen 176 x 144 bildpunkter, två motorer med tillhörande hjul, samt ett antal aluminium-plåtar och dylika attiraljer som krävs för att konstruera en gedigen fobollsrobot. Datorn saknar flyttalsprocessor.

Konstruktion



Robotens konstruktion var enkel. Inledningsvis användes ett litet hjul som bakre stöd, men då det på ett tidigt stadium visade sig att detta förorsakade en markant manövreringsproblematik, konstruerade kursens labbassistent Anders ett temporärt substitut i form av en fasttejpad sked. Det visade sig att denna nödlösning passade utmärkt. Så några ytterligare modifieringar, som kunde leda till framtida eventuella manövreringsfiaskon, var det givetvis inte tal om.

Kameran är placerad på så sätt att den ska kunna få överblick över hela spelplanen från alla positioner, och i görligaste mån registrera så lite aktivitet som möjligt utanför spelplanen, vilket kan skapa förvirring hos den intellektuellt underutvecklade roboten.

I denna implementation användes inte rollern eller servon.

De regler[2] som definierats inför tävlingen introducerade bland annat en maxdiameter på 18 cm.

Metod

Inför träningen av supportvektormaskinerna insamlades sammanlagt ett 300-tal fotografier av spelplanen. Tyvärr tog överföringen mellan robot och stationär dator ungefär 30 sekunder per fotografi, vilket medförde en total väntan på ungefär 150 minuter, exklusive tidsåtgången för själva fotograferingen. Den varierande kvalitén på fotografierna medförde att detta överflöd av fotografier var nödvändigt för att få en klassificering i högsta klass. Dessvärre visade det sig, något som jag inte var medveten om från början, att kameran behövde ta ett 100-tal fotografier innan den inbyggda autobrightness-funktionen stabiliserats.

Val av mjukvara

Då det visade sig att C++-kompilatorn för Eyebot:en fungerade utmärkt, var detta ett givet alternativ, eftersom det underlättar underhåll av källkoden och kod-enkapsulering minskar möjligheterna att introducera buggar (om man inte trivs vansinnigt bra med C dvs).

Som klassificering användes modeller som extraherats från SVM[1][3]-implementationen TinySVM[4].

Utöver källkod för att driva roboten, utvecklades ett antal hjälpscript/program i GNU Bash[5], C++[6] och Octave[7]. Dessa användes framförallt för att ladda ner kamerafoton från roboten, manipulera bildformat och underlätta träningen av SVM-näten.

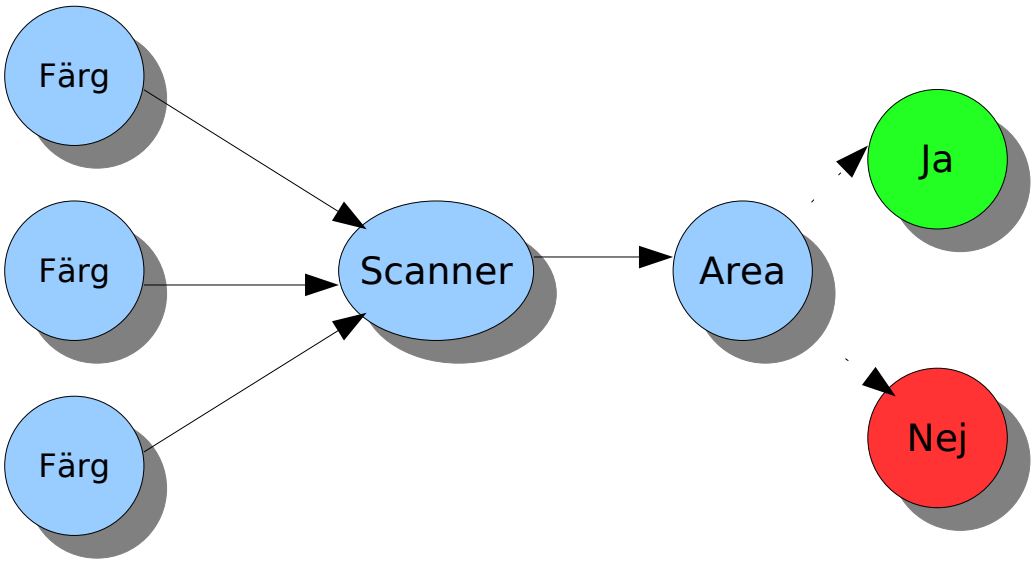
Klassificerare

Ett viktigt problem för att en robot ska kunna hitta objekt på en spelplan är att utveckla bra klassificerare, och i detta projekt bestod dessa av två SVM-varianter och en “area-scanner”. Då det visade sig att modellerna som TinySVM genererade renderade en ofantlig mängd supportvektorer, var den enda möjliga lösningen att använda en linjär kärna. Detta medför att supportvektormaskinen tappar i prestanda vad gäller generaliseringsförmågan, vilket vi också kan konstatera i resultatdelen. Fördelen med en linjär kärna är att tidsåtgången för en klassificering handlar om ett fåtal cpu-cykler, med bibehållen hyfsat god precision.

Eftersom hyperplanen i de olika SVM-modellerna av naturliga skäl är representerade med flyttal, och datorn saknar flyttalsprocessor, var det nödvändigt att konvertera dessa flyttal till heltal med en lämplig skalningsparameter.

Figuren nedan visar stegen för en klassificering. Scannern söker igenom kamerabilden och använder färg-klassificeraren för att detektera ett område av positiva klassificeringar. Areaklassificerarens uppgift är att avgöra om area, bredd, höjd och y-värde kan medföra en positiv klassificering. T.ex. kan bollen lätt förväxlas med ett gult mål, därför är detta steg nödvändigt.

1 Support Vector Machine



...

Klasser

Klassificerarna måste identifiera följande tre klasser:

1. Boll
2. Blått mål
3. Gult mål

Färg

Inparametrar till denna SVM är RGB-värdet samt y-värdet i kamerafotot. Det visade sig att bollen och gult mål lätt förväxlades, men då målet tenderar att ligga högt upp i fotot och bollen närmare mitten gav denna extra y-parameter större precision i klassificeringen.

Scanner

Scannerns uppgift är att detektera ett område i fotot av positiva klassificeringar. Den beräknar också områdets area, position, bredd samt höjd.

I Scanner-klassen kan man välja vilka områden i fotot som ska scannas, samt optimera genom att ställa in på vilken rad den ska gå från att scanna varje, till att scanna var fjärde, pixel.

Area

Den andra SVM:ens syfte är att validera resultatet från scannern, som redan nämnts ovan är detta nödvändigt för att minimera felaktiga klassificeringar.

Styrsystem

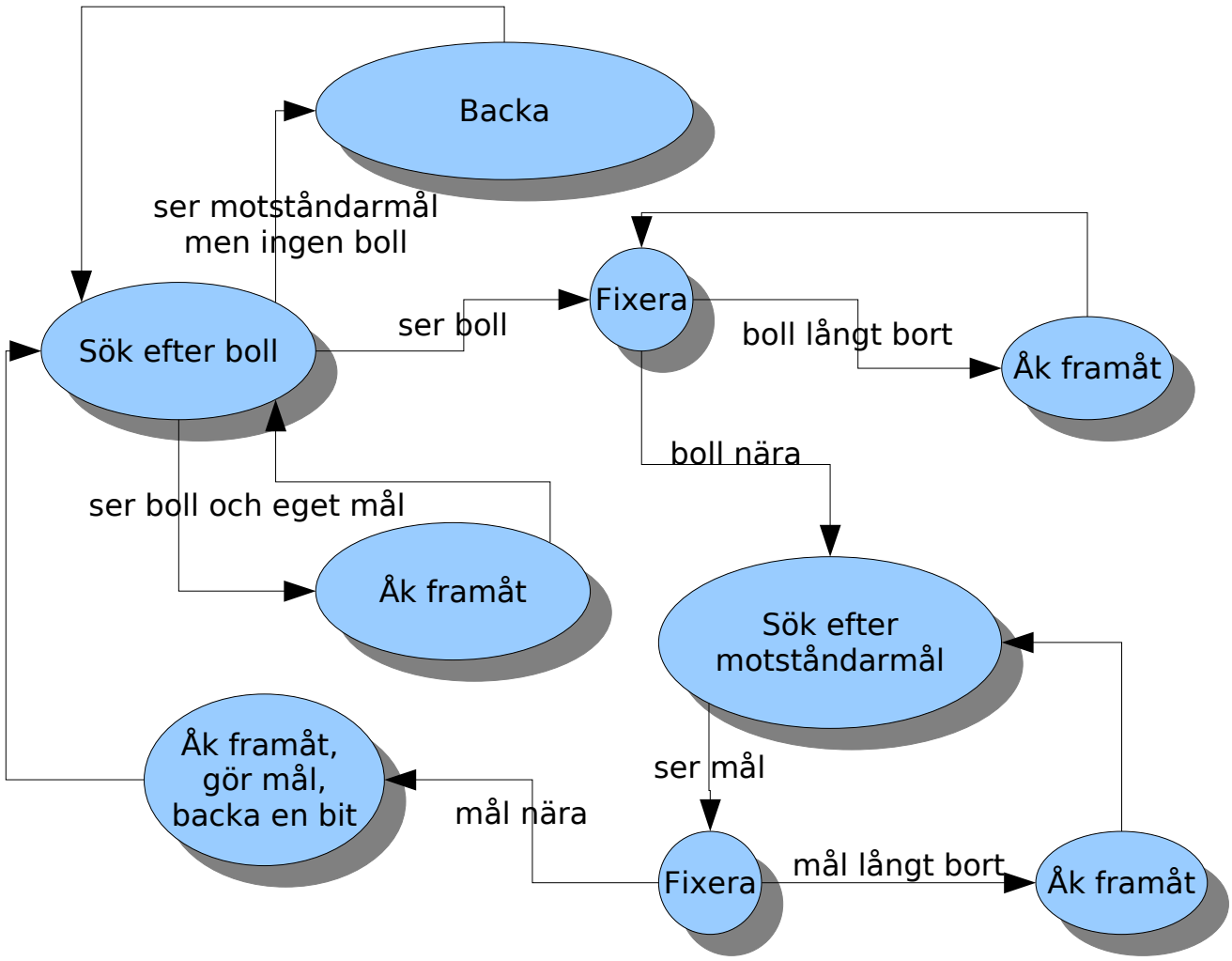
Tyvärr visade det sig att robotens enkoder-system hade gått sönder, därför utvecklades en enklare variant som använde sig av tidsbaserad motorstyrning. Problemet med denna approach är att

roboten svängde olika mycket beroende på hur mycket kräm det fanns i batterierna. Dessutom visade det sig att motorerna roterade olika snabbt med samma spänning.

Gissningsvis bestod ungefär 30-50% (!) av arbetet i robot-labbet åt att kalibrera styrningen, så att roboten åtminstone skulle få en acceptabel manövreringsförmåga.

Robotens taktik

Robotens tillståndskarta[1] kan beskådas i figuren nedan.



Starttillstånd är "Sök efter boll".
"Fixera" innebär att roboten siktar in sig på objektet så det befinner sig rakt framför.

Denna enkla modell fungerade ganska bra för att lokalisera bollen och göra mål, men det fanns en risk att den roterade i all oändlighet och aldrig hittade bollen, detta inträffade dock väldigt sällan. Emellertid hade den överhängande lätt för att fastna i t.ex. ett hörn.

Det var inte nödvändigt att implementera en kontroll för att se om den verkligen behärskade bollen när den försökte göra mål, eftersom den valda taktiken vanligtvis positionerade roboten bakom bollen, och de praktiska resultaten visade att roboten missade bollen endast ett fåtal gånger.

1 För att göra figuren lättöverskådlig (utan för den skull missvisande) har ett fåtal, ej kritiska, optimeringsmoment i tillståndskartan uteslutits.

Resultat

Träningsdata

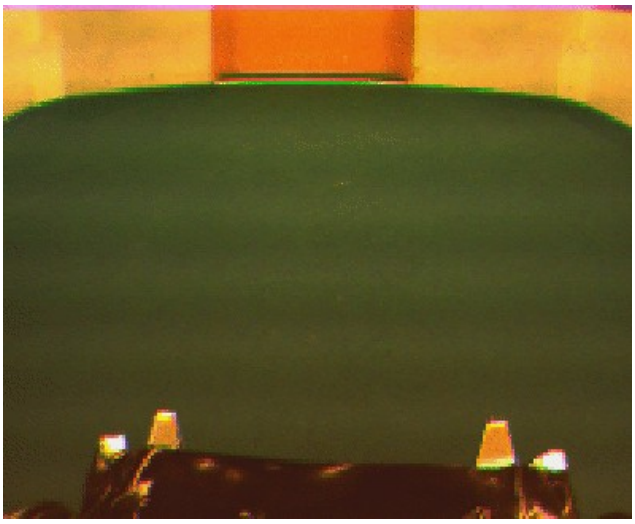
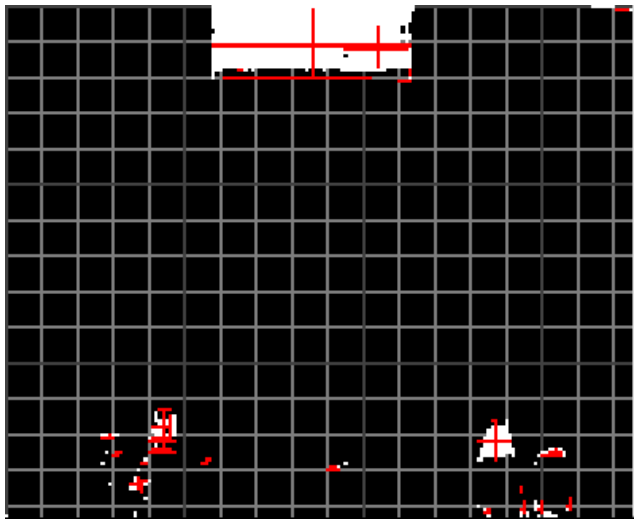
Nedan följer resultat från ett antal utvalda kamerafoton.

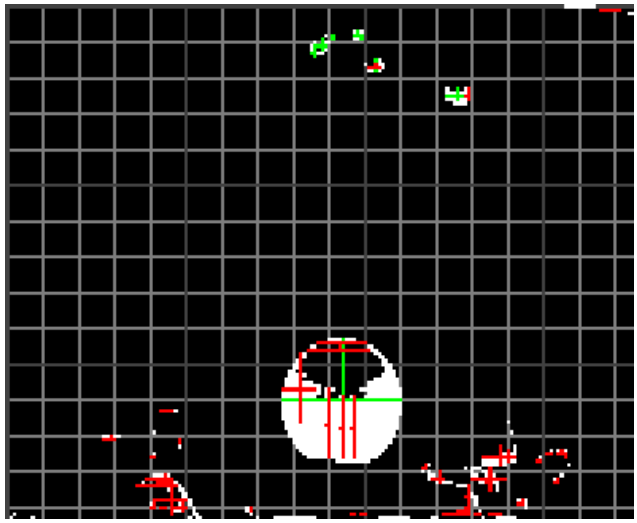
De vita områdena i klassificeringsbilden i högerspalten indikerar att färgklassificeraren klassificerar den pixeln positivt.

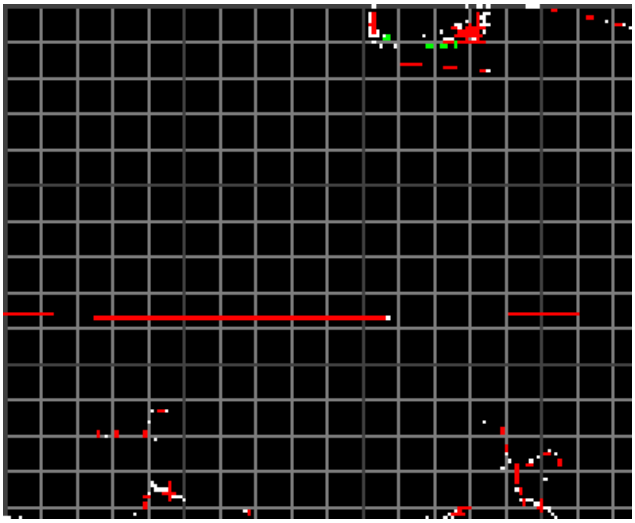
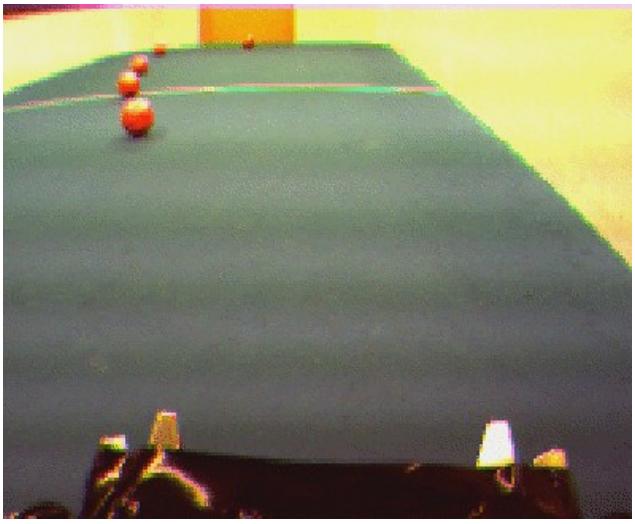
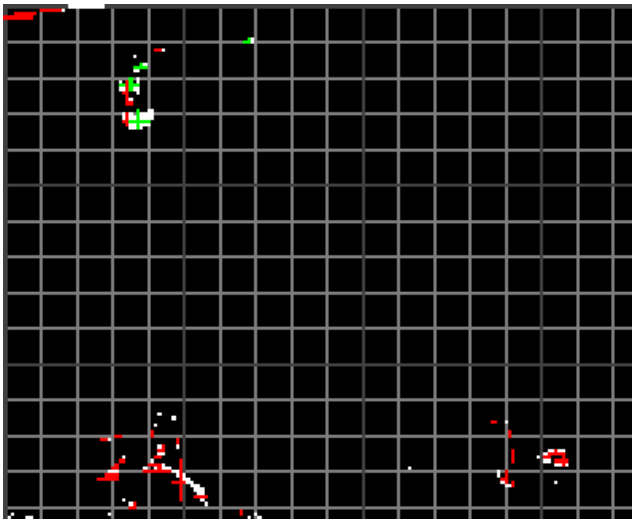
Gröna och röda kors i fotot indikerar ett **positivt** respektive **negativt** klassificerat område.

Varje kvadrat i klassificeringsbilderna är 10x10 bildpunkter.

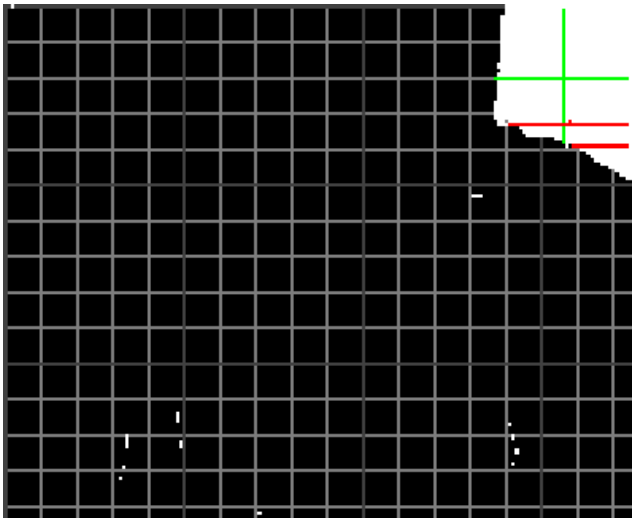
Boll

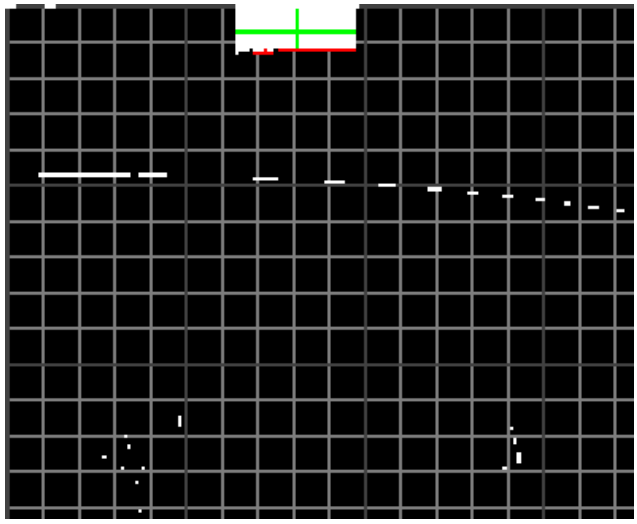



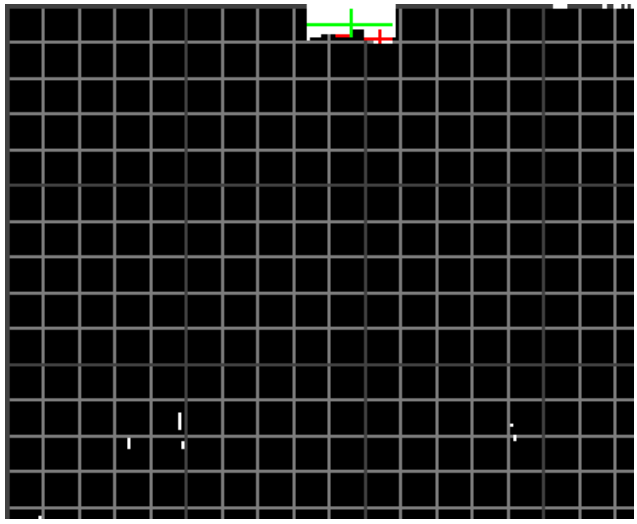
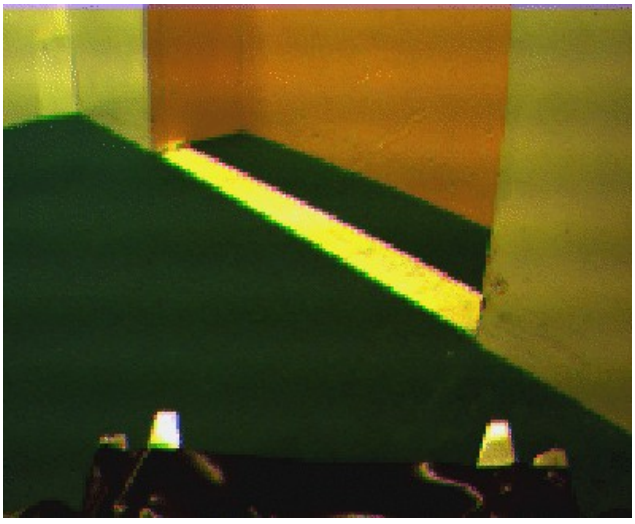
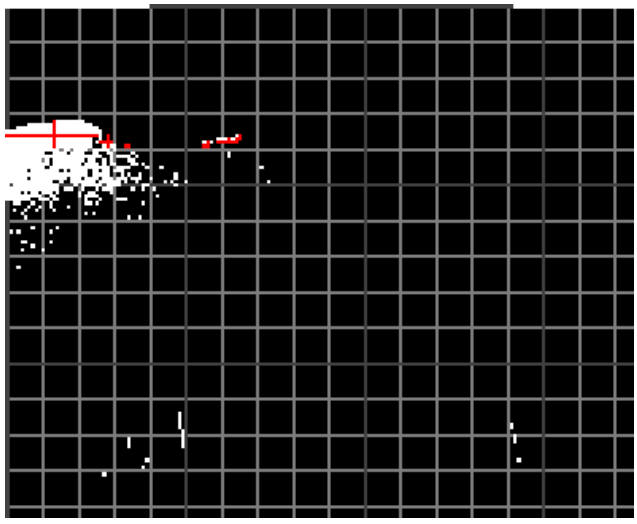


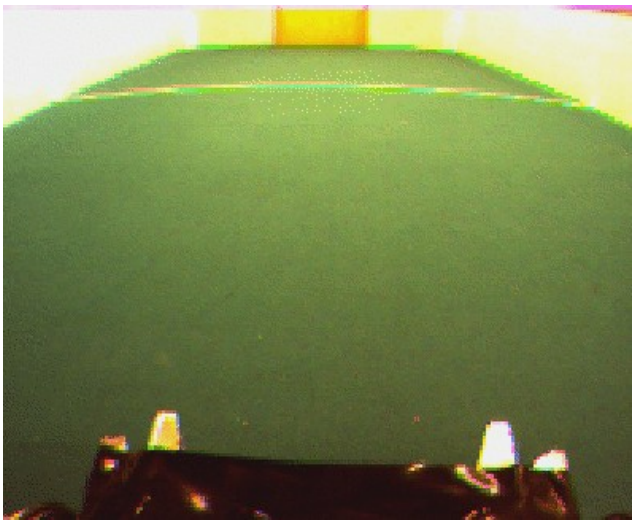
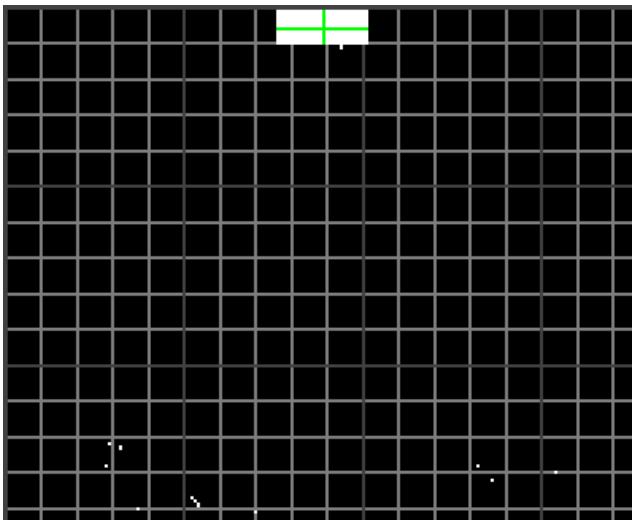
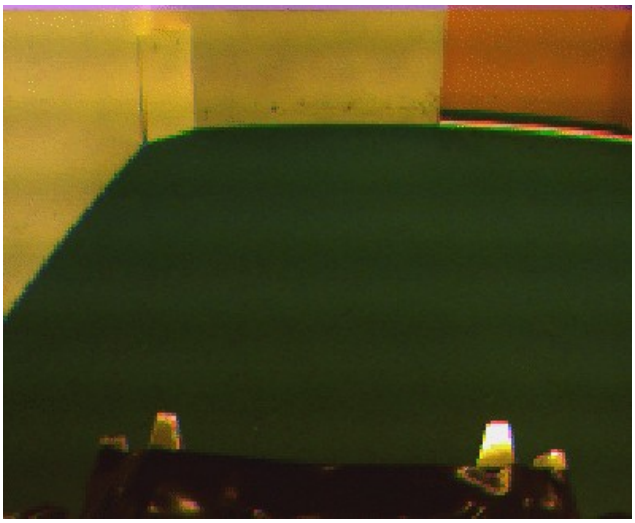
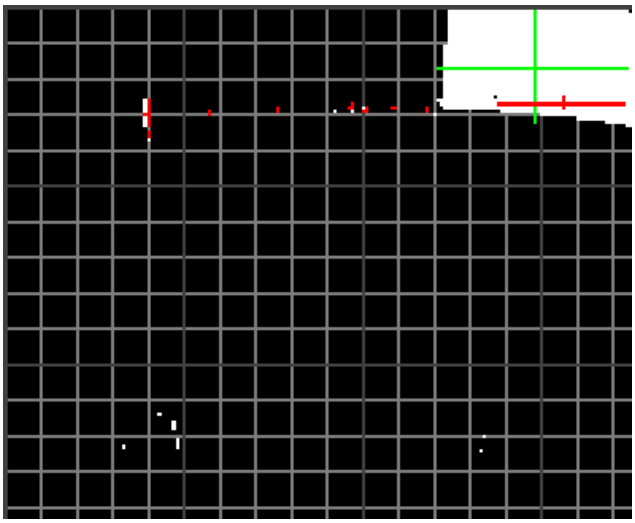
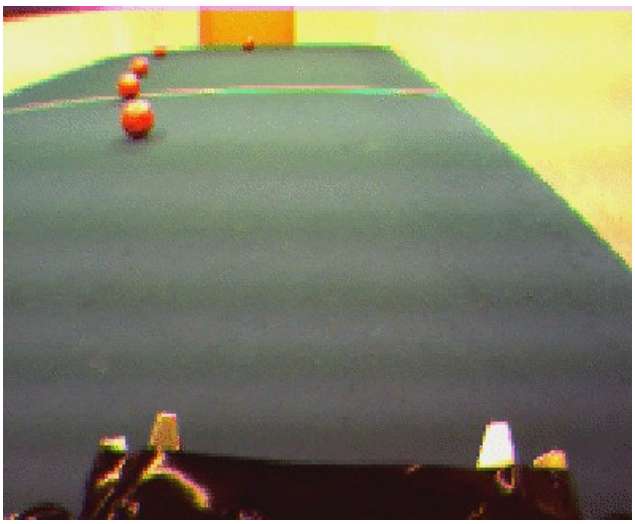
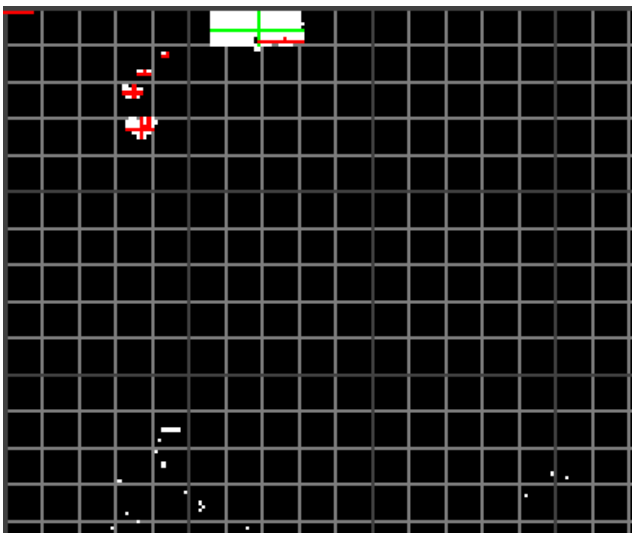
Blått mål

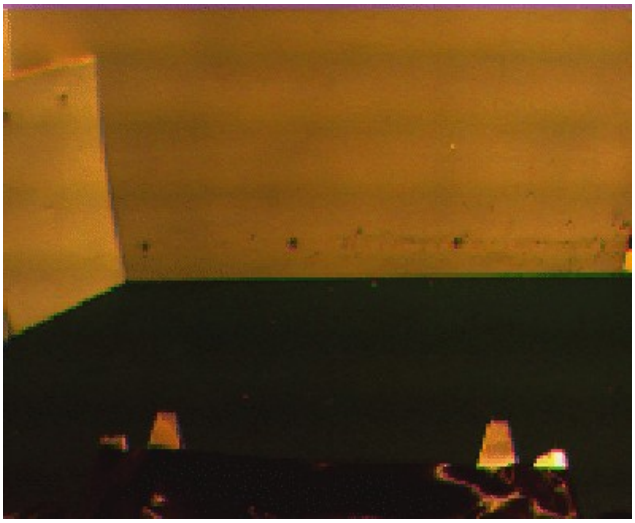



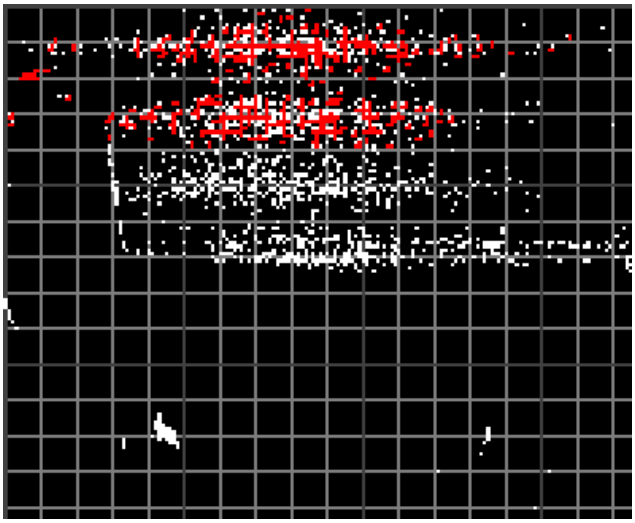



Gult mål





Det sista fotot visar väggen och inte det gula målet. Även om vissa bildpunkter är positivt klassificerade blir den slutgiltiga klassificeringen negativ, eftersom areaklassificeraren förkastar detta foto.

Man kan konstatera att roboten inte hade några problem att identifiera målen, trots att bilderna varierade kraftigt. Men ibland kunde den förväxla boll och gult mål på långa avstånd, vilket också återspeglar sig i de praktiska resultaten.

En jämförelse av SVM-Kärnor

Hade man haft tillgång till lite bättre hårdvara, 35 MHz och 2 MB RAM är inte direkt topnotch, så skulle man kunna ha använt en icke-linjär SVM-kärna. En stor styrka med en supportvektormaskin är att man kan utnyttja en kärna som “translaterar” träningspunkter till en högre dimension[1]. Data som inte är linjärt separerbara i den ursprungliga dimensionen har större sannolikhet att vara det om man kan representera dem i en högre dimension.

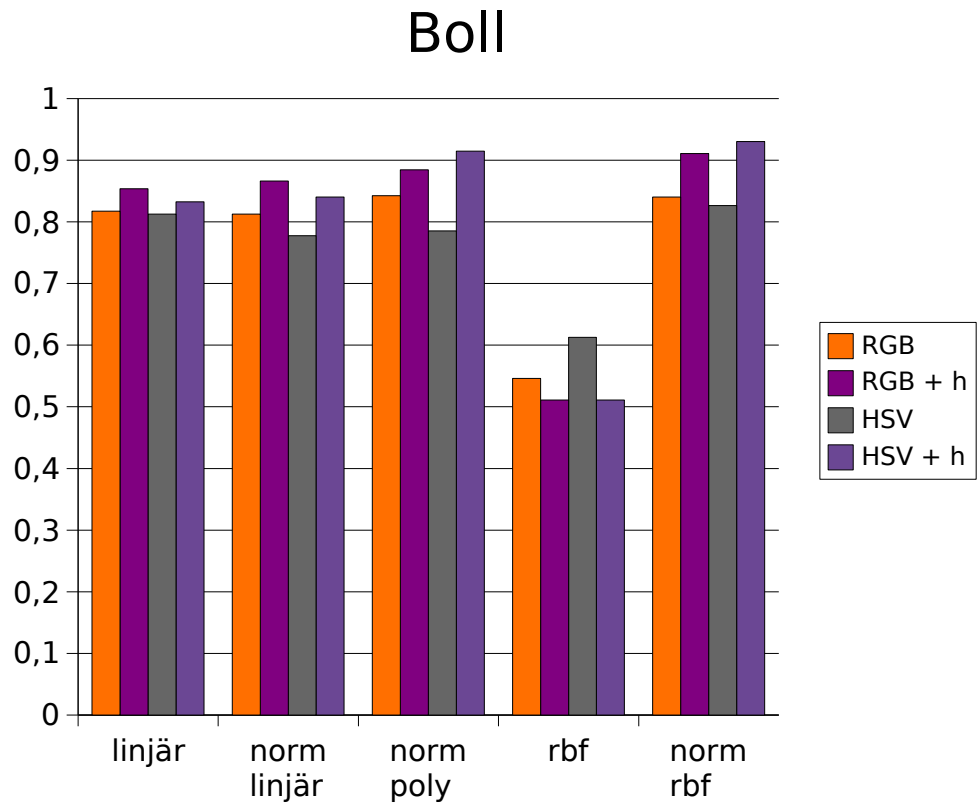
Nedan ser vi resultatet från färgklassificeraren på test-data, med olika kärnor[2] och representationer[3] av en pixel. De båda målens klassificerare presenterade goda resultat redan med en linjär kärna, men precisionen i bollens klassificerare ökade markant med en icke-linjär.

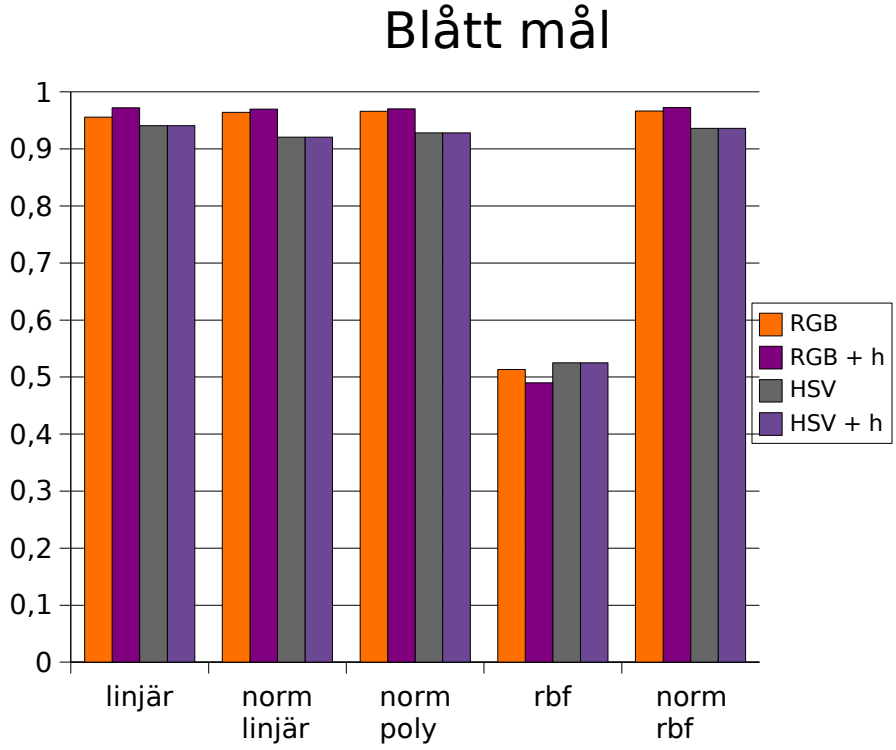
I praktiken användes en linjär kärna med RGB + y-värde som inparamtrerar, där RGB-värdena gick från 0 till 255 och y-värdet från 0 till 143.

1 Begreppet dimension syftar på antal parametrar man använder för en träningspunkt (tex RGB ger en 3-dimensionell träningspunkt och RGB + h ger en 4-dimensionell)

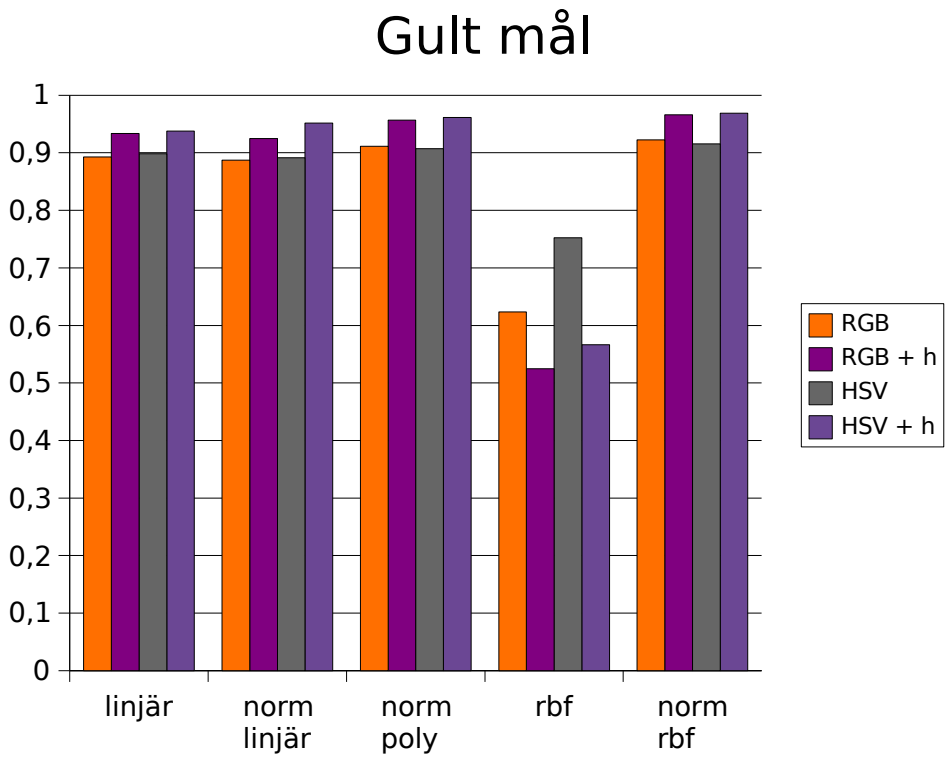
2 Linjär, normaliserad linjär, normaliserad polynomisk, radial basis samt normaliserad radial basis kärna.

3 RGB (Red, Green, Blue), RGB + pixelns y-värde i fotot, HSV (Hue, Saturation, Value), samt HSV + y-värde.





Man kan konstatera att blått mål (till skillnad från gult mål och boll) inte nämnvärt påverkas av introduktionen av en fjärde y-parameter. Detta beror givetvis på att gult mål och bollen har likartade färger, vilket till viss mån kan åtgärdas om man introducerar en y-parameter eftersom gult mål tenderar att ligga högt upp i fotot och bollen närmare mitten.



Praktiska resultat

Precis som de teoretiska resultaten redovisar, hade roboten inga (seriösa) problem med klassificeringsdelen. Däremot hade den en tendens att fastna längs väggen, eftersom det inte hade skrivits någon kod för att undvika sådana ödesdigra parkeringar. Då roboten aldrig deltog i tävlingen kändes det inte speciellt nödvändigt att åtgärda detta.

Utan enkodrar hade roboten en urusel manövreringsförmåga, och det medförde en ganska lång tidsåtgång för att göra mål. Om den inte fastnat rörde det sig om ungefär 20-60 sekunder.

När roboten skulle sikta in sig på ett objekt, var ett ständigt återkommande problem ett slags “hackigt” rörelsemönster. Det var inga problem att räkna ut hur mycket den behövde rotera för att hamna rakt framför objektet, men utan fungerande enkodrar medförde det stora svårigheter att uppnå en acceptabel precision.

Effektiviteten hade ökat markant om roboten hade haft möjlighet att köra rakt. Som nödlösning fick den sikta in sig något snett på motståndarmålet, så att den inte skulle fastna vid ena stolpen.

Slutsatser och avslutning

Det största problemet i detta projekt visade sig vara att åtgärda de manövreringsproblem som uppstod i samband med att en enkoder gick sönder.

Kamerans stabilitetsbesvär med autobrightness innebar inledningsvis, innan man var medveten om detta problem, tråkigt missvisande resultat i samband med klassificeringen. Men trots detta hade klassificerarna i det stora hela en god precision.

Det visade sig att supportvektormaskinerna hade störst problem med att korrekt klassificera golfbollen och gult mål, högst troligtvis då dessas färger var snarlika. Introduktionen av en höjdparameter i färg-klassificeraren medförde en prestandaförbättring på c:a 5%. Vidare förbättringar kunde åskådas i samband med introduktionen av ytterligare en klassificerare, den så kallade area-klassificeraren.

Den långsamma processorn och att TinySVM renderade en stor mängd supportvektorer, medförde att med avseende på prestanda, var det svårt att utnyttja annat än linjära kärnor i klassificeringsuppgifterna.

Källor

1: http://robotics.ee.uwa.edu.au/eyebot/
2: http://www.csc.kth.se/utbildning/kth/kurser/DD2426/rules/
3: http://en.wikipedia.org/wiki/Support_vector_machine
4: http://chasen.org/~taku/software/TinySVM/
5: http://www.gnu.org/software/bash/
6: http://en.wikipedia.org/wiki/C%2B%2B
7: http://www.gnu.org/software/octave/

Bilagor

Källkod

```
/*
 * main.cc
 */

#include "robot.hh"
#include "class.hh"
#include "classifier.hh"
#include <math.h>

int main() {
    Robot c;

    BYTE *tmp_pic = new BYTE [3*WIDTH*HEIGHT];
    //autobrightness eller whatever
    int count = 0;
    while (!KEYRead()) {
        CAMGetFrameRGB(tmp_pic);
        ++count;
        if (!(count%10)) {
            LCDClear();
            LCDPrintf("%d\n", ++count);
        }
    }
    delete []tmp_pic;


    //bollen
    Class ball(c.image, 0.0209229 , -0.0182353 , -0.000261873 , 0.00648072,
2.69411);
    ball.typ = 1;

    //blå mål
    Class blue_goal(c.image, -0.0567411 , -0.0437567 , 0.0655035 , -0.0405322 ,-
3.70485);
    blue_goal.typ = 2;

    //gult mål
    Class yellow_goal(c.image, 0.0253956 , -0.0174306 , -0.278747 , -0.0123208 ,
-3.60956);
    yellow_goal.typ = 3;

    //marken
    Class ground(c.image, -0.0405773 , 0.0419386 , -0.0143119 , 0.037585 ,
1.31846);
    ground.typ = 4;

    //area för bollen
    ClassifierSVM ball_area_classifier(-0.0052788 , -0.0816207 , 0.17352 ,
0.206457 , -0.783924);
    ball.area_classifier = &ball_area_classifier;

    //area för blå
    ClassifierSVM blue_area_classifier(0.0151087 , -0.0667322 , -0.0016728 ,
0.00323663 , 1.192);
    blue_goal.area_classifier = &blue_area_classifier;
```

```
    //area för gul
    ClassifierSVM yellow_area_classifier(0.0154586 , -0.00109549 , -0.0017041 ,
0.000608606, 2.34465);
    yellow_goal.area_classifier = &yellow_area_classifier;


    //ställ in varje scanner så den funkar okej
    Scanner ball_scanner(&ball);
    ball_scanner.height = 110;
    //ball_scanner.border = HEIGHT/6; //obs funkar tydligen inte så bra

    Scanner yellow_goal_scanner(&yellow_goal);
    yellow_goal_scanner.height = 30;

    Scanner blue_goal_scanner(&blue_goal);
    blue_goal_scanner.height = 30;

    Scanner ground_scanner(&ground);
    //ground_scanner.border = 0;
    ground_scanner.x_border = WIDTH/3;
    ground_scanner.set_width(WIDTH/3);
    ground_scanner.height = 100;//HEIGHT / 4;


    Scanner *goal_scanner = &blue_goal_scanner; //eller yellow
    Scanner *my_goal_scanner = &yellow_goal_scanner; //eller blue


    //ganska lat lösning men ok
    Robot::my_goal = &yellow_goal_scanner;
    Robot::opponent_goal = &blue_goal_scanner;


    int state = FIND_BALL;

    OSWait(30);
    c.engine.drive_straight(0.2);

#define STATE(s) LCDPrintf("%s\n",s);

    while (KEYRead() != KEY4) {
        switch (state) {
        case FIND_BALL:
            STATE("FIND_BALL");
            //snurra och se till så att bollen är rakt framför
            state = c.search(ball_scanner, BALL_AHEAD, MOVE_AROUND);
            break;

        case BALL_AHEAD:
            STATE("BALL_AHEAD");
            //åk framåt tills bollen är under kontroll
            state = c.drive_ahead(ball_scanner, FIND_GOAL, MOVE_AROUND);
            break;

        case FIND_GOAL:
            STATE("FIND_GOAL");
            //OSWait(300);
            state = c.search(*goal_scanner, GOAL_AHEAD, MOVE_AROUND);
            break;
```

```
        case GOAL_AHEAD:
            STATE("GOAL_AHEAD");
            state = c.drive_ahead(*goal_scanner, FIND_BALL, FIND_BALL);
            break;

        case MOVE_AROUND:
            STATE("MOVE_AROUND");
            //hitta på nåt kul
            c.move_around(ground_scanner);
            state = FIND_BALL;
            break;
        }
    }

    return 0;
}
```

```
/*
 * robot.hh
 */

#ifndef __ROBOT_HH
#define __ROBOT_HH

#include "scanmap.hh"
#include "state.h"
#include "engine.hh"

#define WIDTH 176
#define HEIGHT 144

struct Robot {
    Robot()
        : randomized(false), dir(1), last_y(0)
    {
        init_cam();
        image = new BYTE [3*WIDTH*HEIGHT];
    }
    ~Robot() {
        CAMRelease();
        delete []image;
    }
    bool classify(Scanner &, int *x, int *y, int *area, bool do_take_pic = true);
    bool fixate(Scanner &, int x = -1, int y = -1, int s = 0);
    State search(Scanner &, State next_state, State fail_state, int s = 0);
    State drive_ahead(Scanner &, State next_state, State fail_state);
    void move_around(Scanner &);
    void init_cam();
    void cl_test(Scanner &);

//private:
    Engine engine;
    void take_pic();
    BYTE *image;
    bool randomized;
    int dir;

    //fult men nåväl
    static Scanner *my_goal;
    static Scanner *opponent_goal;
    int last_y;
    static int h_g;
};


#endif
```

```
/*
 * robot.cc
 */

#include "robot.hh"
#include "class.hh"
#include <stdlib.h>
#include <math.h>


Scanner *Robot::my_goal = 0;
Scanner *Robot::opponent_goal = 0;
int Robot::h_g = 0;

#define TURN_RAD ((float)M_PI/(float)8.0)
#define TURNS_PER_LAP ((int)(TURN_RAD/(2*M_PI)))


void Robot::take_pic() {
    OSWait(10); //XXX: behövs inte?
    CAMGetFrameRGB(image);

    if (!randomized) {
        randomized = true;
        srand(image[WIDTH*HEIGHT*3/2]);
    }
}

bool Robot::classify(Scanner &sc, int *x, int *y, int *area, bool do_take_pic) {
    if (do_take_pic)
        take_pic();

    sc.new_scan();

    int w,h;
    while (1) {
        *area = sc.get_area(x, y, &w, &h);
        if (*area) {
            h_g = h;
            last_y = *y;
            //LCDPrintf("%d,%d wh%d,%d a%d\n", *x, *y, w, h, area);
            //OSWait(200);
            if (sc.cl->yes(*x,*y,*area, w, h)) {
                //LCDPrintf("yes\n");
                LCDPrintf("%d,%d wh%d,%d a%d\n", *x, *y, w, h, *area);
                //OSWait(300);
                return true;
            }
        } else break;
    }

    return false;
}

void Robot::cl_test(Scanner &sc) {
    //LCDClear();
    LCDPrintf("...\n");
    take_pic();
    sc.new_scan();
    int area, w,h,x,y;
```

```
    while (1) {
        area = sc.get_area(&x, &y, &w, &h);
        if (area) {
            //LCDPrintf("%d,%d wh%d,%d a%d\n", *x, *y, w, h, area);
            //OSWait(200);
            if (area>500) {
                //LCDPrintf("yes\n");
                LCDPrintf("%d,%d wh%d,%d a%d\n", x, y, w, h, area);
            }
        } else break;
    }

    OSWait(500);
}

bool Robot::fixate(Scanner &sc, int x, int y, int s) {
    int area;
    if (x == -1 && !classify(sc, &x, &y, &area))
        return false;

    int middle = WIDTH/2;
    int time_out = 30;
    //XXX: obs tmp
    int dd = 7;//+10;
    //int s = 0;//35;// -8; //justera skiten

    //testa ev s+=nånting för blå
    if (sc.cl->typ == 2) {
        s -= 20;
    }

    while (1) {
        x += s;
        LCDPrintf("m:%d xy:%d %d\n", middle, x,y);
        //OSWait(200);
        if (x >= middle-dd && x <= middle+dd) {
            return true;
        }
        else {
            float rot = (float)0.4*(float)(x - WIDTH/2)/(float)(HEIGHT - y);
            if (rot > 0.7) rot = 0.7; else if (rot < -0.7) rot = -0.7;
            LCDPrintf("%drot:%.2f\n", sc.cl->typ, rot);
            engine.turn( rot ); //XXX: kolla tecknet
            //OSWait(100);
        }

        if (--time_out <= 0){
            LCDPrintf("time out\n");
            return false;
        }

        //int old = x;
        if (!classify(sc, &x, &y, &area))
            return false;
    }

    return false;

}
```

```
State Robot::search(Scanner & sc, State next_state, State fail_state, int s) {
    int turns = TURNS_PER_LAP;
    int x,y,area;
    while (1) {
        if (classify(sc, &x, &y, &area)) {
            if (sc.cl->typ == 1 && classify(*my_goal, &x, &y, &area, false)) {
                if (area < 1000) {
                    engine.drive_straight(0.2);
                    engine.drive_straight(-0.05);
                }
            }
            else {
                if (fixate(sc, x, y, s))
                    return next_state;
                else
                    return fail_state;
            }
        }
        else if (sc.cl->typ == 1) {
            if (classify(*opponent_goal, &x, &y, &area, false)) {
                if (area > 300) {
                    engine.drive_straight(-0.1);
                    engine.drive_straight(0.05);
                }
                else {
                    engine.drive_straight(0.25);
                    engine.drive_straight(0.05);
                }
            }
        }

        if (--turns > 0) break; //vafan?
        engine.turn((float)dir*(float)TURN_RAD);
    }
    return fail_state;
}

State Robot::drive_ahead(Scanner & sc, State next_state, State fail_state) {
    int pos = -1;
    int x,y,area;
    if (!classify(sc, &x, &y, &area)) return fail_state;

    while (1) {
        //drive ahead
        int speed = (110 - y)/2;
        if (speed < 30) speed = 30;
        if (speed > 45) speed = 45;
        engine.set_engines(speed, speed*21/20);

        LCDPrintf("%ddrive_ahead %d y%d\n",sc.cl->typ, speed, y);

        int new_area;
        if (!fixate(sc) || !classify(sc, &x, &y, &new_area)) {
            engine.set_engines(0, 0);
            return fail_state;
        }

        if (sc.cl->typ == 1) {
            for (int i = 0; i < 20; ++i) LCDPrintf("%d  ", y);
            LCDPrintf("\n");
```

```
        }

        if (y > sc.cl->near(this, new_area)) {
            engine.drive_straight(0.08);
            engine.set_engines(0, 0);
            LCDPrintf("yes\n",y);
            return next_state;
        }
    }
    engine.set_engines(0, 0);
    return fail_state;
}

void Robot::move_around(Scanner &g_sc) {
    dir *= -1; //byt sök-riktning
}



void error(char *str)
{
    LCDPrintf("ERROR: %s\n", str);
    OSWait(200);

    exit(0);
}

void Robot::init_cam() {
    int camera;
    int bright, hue, sat;

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

    if (camera < COLCAM)
        error("No color camera!");
    else
        {
            if (camera == NOCAM)
                error("No camera!\n");
            else
                if (camera == INITERROR)
                    error("CAMInit!\n");
        }

}
```

```
/*
 * class.hh
 */

#ifndef __CLASS_HH
#define __CLASS_HH

#include "eyebot.h"
#include "state.h"
#include "classifier.hh"
#include "scanmap.hh"
#include "engine.hh"


#define WIDTH 176
#define HEIGHT 144

#define USE_MOTORS

void error(char *str);

struct Robot;

struct Class {
    Class(BYTE *img, float _r, float _g, float _b, float _row, float bias, int
_scale = 100000)
        : image(img), half_reversed_y(false), area_classifier(0)
    {
        svm = new ClassifierSVM(_r, _g, _b, _row, bias, _scale);
    }
    ~Class() { delete svm; }

    bool classify(int x, int y);
    bool yes(int x, int y, int area, int w, int h);
    int near(Robot *, int);

    ClassifierSVM *svm;
    ClassifierSVM *area_classifier;

    int typ; //fult men skitsamma
    bool half_reversed_y;
    BYTE *image;
};


#endif
```

```
/*
 * class.cc
 */

#include "class.hh"
#include "scanmap.hh"
#include "robot.hh"

//kollar så inte bollen är helt fel
bool ball_sanity_check(int y, int w, int h) {
    if (w <= 0 || h <= 0 || y < 10) return false;

    //tillhöftade från mina härliga träningsdata
    if (y < 20) {
        int d = w < h ? h/w : w/h;
        if (y < 14) {
            return (w <= 5 && h <= 5 && (d==1 || d==2));
        }
        else {//y<20
            return (w <= 7 && h <= 7 && d == 1);
        }
    }
    else if (y < 30) return (w <= 10 && h <= 10 && h > 1 && w > 3);
    else if (y < 60) return (w <= 25 && h <= 25 && h > 3 && w > 5);
    else return (w <= 45 && h <= 45 && h > 7 && w > 12);
}

bool Class::classify(int x, int y) {
    int red = image[y*3*WIDTH + x*3];
    int green = image[y*3*WIDTH + x*3 +1];
    int blue = image[y*3*WIDTH + x*3 +2];

    if (half_reversed_y && y < HEIGHT/2) y = HEIGHT-y; //XXX: OBS test

    if (svm->classify(red, green, blue, y) == 1)
        return true;
    else
        return false;
}

//XXX: !!!
//inga arv och virtuella metoder, förmodligen gör det absolut ingen skillnad i
prestanda men whatever
bool Class::yes(int x, int y, int area, int w, int h) {
    bool b = false;
    if (typ == 1) { //bollen
        if (w > 0 && h > 0) {
            bool b1 = (bool)(area_classifier->classify(area, y, w, h) == 1);
            bool b2 = (bool)(ball_sanity_check(y, w, h));
            b = b1 && b2;
        }
    }
    else if (typ == 2) { //blå mål
        //antingen funktionen eller svm:en
        //b = is_a_goal(y,area);
        b = (bool)(area_classifier->classify(area, y, w, h) == 1);
    }
    else if (typ == 3) { //gult mål
        //antingen funktionen eller svm:en
        //b = is_a_goal(y,area);
```

```
        //b = (bool)(area > 30 && area_classifier->classify(area, y, w, h) == 1);
        b = (bool)(area_classifier->classify(area, y, w, h) == 1);
    }
    else if (typ == 4) { //marken
        b = (bool)(area > 500);
    }

    return b;
}


//när man är tillräckligt nära för att göra nåt annat
int Class::near(Robot *cl, int area) {
    if (typ == 1) { //bollen
        return 60;
    }
    else if (typ == 2) { //blå mål, eller snarare motståndarmålet
        LCDPrintf("h:%d\n", Robot::h_g);
        //OSWait(500);
        if (Robot::h_g > 25) {
            cl->engine.drive_straight(0.25);

            //tjänar på att köra in i väggen :)
            for (int i = 0 ; i < 7; ++i) {
                cl->engine.drive_straight(0.10);
            }

            cl->engine.drive_straight(-0.15);
            for (int i = 0; i < 3; ++i) {
                cl->fixate(*Robot::opponent_goal);
                //OSWait(10);
                cl->engine.drive_straight(-0.1); //kör tillbaka
            }
            //OSWait(500);
            return 0; //så går den till nästa state
        }
        return 500;//last_area > 3000;
        //return 70;
    }
    else if (typ == 3) { //gult mål, eller eget mål
        return 70;
    }
    else if (typ == 4) { //hm...höftar vilt
        return HEIGHT / 8;
    }

    return 0;
}
```

```
/*
 * classifier.hh
 */

#ifndef __CLASSIFIER_HH
#define __CLASSIFIER_HH

struct ClassifierSVM {
    ClassifierSVM(float _r, float _g, float _b, float _row, float bias, int
_scale = 100000);
    int classify(int red, int green, int blue, int _row);
    int get_scale() { return scale; }
private:
    int r,g,b,row,scale,bias;
};

#endif
```

```
/*
 * classifier.cc
 */

#include "classifier.hh"

ClassifierSVM::ClassifierSVM(float _r, float _g, float _b, float _row, float
_bias, int _scale) {
    scale = _scale;
    r = (int)((float)scale*_r);
    g = (int)((float)scale*_g);
    b = (int)((float)scale*_b);
    row = (int)((float)scale*_row);
    bias = (int)((float)scale*_bias);
}

int ClassifierSVM::classify(int red, int green, int blue, int _row) {
    int sum = red*r + green*g + blue*b + _row*row  -  bias;
    //cout << sum << " ";
    if (sum < 0)
        return 0;
    else
        return 1;
}
```

```
/*
 * engine.hh
 */

#ifndef __ENGINE_HH
#define __ENGINE_HH

#include "eyebot.h"

#define USE_MOTORS

struct Engine {
    Engine();
    ~Engine() {
#ifdef USE_MOTORS
        MOTORRelease (right_motor);
        MOTORRelease (left_motor);
#endif
    }
    void turn(float rad);
    void drive_straight(float meters, int speed = 60);
    void set_engines(int l, int r);
 private:
#ifndef USE_MOTORS
    VWHandle vw;
    void vw_stop();
    int vw_start();
#endif
    MotorHandle right_motor;
    MotorHandle left_motor;
};


#endif
```

```
/*
 * engine.cc
 */

#include "engine.hh"
#include "class.hh"
#include <math.h>

Engine::Engine() {
#ifdef USE_MOTORS
        right_motor=MOTORInit(RIGHTMOTOR);
        left_motor=MOTORInit(LEFTMOTOR);
        if (right_motor <= 0 || left_motor <= 0) {
            error("fel på motorn\n");
        }
#endif
}


void Engine::turn(float rad) {
#ifdef USE_MOTORS
    if (right_motor <= 0 || left_motor <= 0) {
        error("fel på motorn\n");
    }

    set_engines(0, 0); //lika bra kanske
    //ev oswait
    OSWait(10);

    int speed = 50;
    if (rad < 0) {
        MOTORDrive(right_motor,speed);
        MOTORDrive(left_motor,-speed);
    } else {
        MOTORDrive(right_motor,-speed);
        MOTORDrive(left_motor,speed);
    }

    rad = rad<0?-1*rad:rad;
    if (rad >= M_PI) //180
        OSWait((int)((float)rad*(float)21.7));
    else if (rad >= M_PI/2.0) //90
        OSWait((int)((float)rad*(float)23.6));
    else if (rad >= M_PI/4.0) //45
        OSWait((int)((float)rad*(float)29.0));
    else if (rad >= M_PI/8.0) //22.5
        OSWait((int)((float)rad*(float)59.0));
    else if (rad >= M_PI/16.0)//11.25
        OSWait((int)((float)rad*(float)77.0));
    else
        OSWait((int)((float)rad*(float)87.0));

    MOTORDrive(right_motor,0);
    MOTORDrive(left_motor,0);
#else
    if (vw_start() == -1) error("Can't init the f-g vw again"); //ooops inte bra
    VWDriveTurn(vw, rad, M_PI*4.0);
    VWDriveWait(vw);
    vw_stop();
#endif
```

```
}

void Engine::set_engines(int l, int r) {
    MOTORDrive(right_motor, r);
    MOTORDrive(left_motor, l);
}

void Engine::drive_straight(float meters, int speed) {
#ifdef USE_MOTORS
    int speed_l = speed; //XXX: obs!
    int speed_r = (speed*21)/20; //XXX: obs!
    //int speed_l = 60; //XXX: obs!
    //int speed_r = 63; //XXX: obs!

    //int speed_l = 50; //XXX: obs!
    //int speed_r = 43; //XXX: obs!
    if (meters < 0) {
        speed_l *= -1;
        speed_r *= -1;
        meters *= -1.0;
    }
    MOTORDrive(right_motor,speed_r);
    MOTORDrive(left_motor,speed_l);

    OSWait((int)((float)meters*500.0));

    MOTORDrive(right_motor,0);
    MOTORDrive(left_motor,0);

    OSWait(30);
#else
    if (vw_start() == -1) error("Can't init the f-g vw again"); //ooops inte bra
    //OSWait(200);
    VWDriveStraight(vw, meters, 2*M_PI);
    VWDriveWait(vw);
    vw_stop();
#endif

}

#ifndef USE_MOTORS
void Engine::vw_stop()
{
    VWSetSpeed(vw,0,0);
    VWStopControl(vw);
    VWRelease(vw); // exit driver
    //OSWait(30);

}

int Engine::vw_start()
{
    vw = VWInit(VW_DRIVE,1); // init v-omega interface
    if(vw == 0) {
        LCDPutString("VWInit Error!\n");
        OSWait(200);
        return -1;
    }
    VWStartControl(vw,7,0.3,7,0.1);
```

```
    return 0;
}
#endif
```

```
/*
 * scanmap.hh
 */

#ifndef __SCANMAP_HH
#define __SCANMAP_HH

struct Class;

#define WIDTH 176
#define HEIGHT 144

struct Scanmap {
    Scanmap() {
        width = WIDTH;
        for (int i = 0; i < WIDTH*HEIGHT; ++i)
            scan_map[i] = 0;
        tick = 0;
    }

    int get(int x, int y) {
        int r = scan_map[width*y + x];
        if (r >> 8 == tick)
            return (r & 0xff);
        else
            return 0;
    }

    void tag(int x, int y, int flag) {
        int r = scan_map[width*y + x];
        if (r >> 8 != tick || (r & 0xff) < flag)
            scan_map[width*y + x] = tick << 8 | flag;
    }

    void new_scan() {
        ++tick;
    }

    static int scan_map[WIDTH*HEIGHT];
    int width;
    int tick;
};

struct Scanner {
    Scanner(Class *c) : cl(c) {
        y_pos = 1;
        border = HEIGHT;
        height = HEIGHT;
        x_border = 0;
    }
    Class *cl;

private:
    bool classify(int x, int y);
    bool get_pos(int *x, int y, int xe, int dir, bool do_classify);
    int scan_row(int x, int y, int *sweep_xs, int *sweep_xe);

public:
    //returnerar ett område, eller area, i aktuellt kamerafoto, eller 0 om det
inte finns något
```

```
    int get_area(int *center_x, int *center_y, int *w, int *h);
    void new_scan();

    int border;
    int x_border;
    int height;

    void set_width(int w) { scan_map.width = w; }

private:
    int y_pos;
    Scanmap scan_map;
};


#endif
```

```
/*
 * scanmap.cc
 */

#include "scanmap.hh"
#include "class.hh"

int Scanmap::scan_map[WIDTH*HEIGHT];

bool Scanner::classify(int x, int y) {
    if (y == HEIGHT-1 || x == scan_map.width -1)
        return false; //av nån jävla anledning så segfaultar den här annars

    return cl->classify(x + x_border, y);
}


bool Scanner::get_pos(int *x, int y, int xe, int dir, bool do_classify) {
    bool hit = false;
    int step = y >= border ? 2 : 1;
    if (step > 1) *x &= 0xfe;
    while (1) {
        int step_x = dir;
        while ( (step_x = scan_map.get(*x, y)) ) {
            *x += step_x;
        }
        if (*x >= scan_map.width || *x >= xe)
            break;

        bool cl = classify(*x,y);
        if (cl) scan_map.tag(*x,y, step);//XXX:

        if (do_classify) {
            if (!cl) break; //om vi letar efter +1
        } else {
            if (cl) break;  //om vi letar efter 0
        }

        *x += dir*step;
        hit = true;

        if (*x >= xe) break;

        if (*x < 0) {
            *x = 0;
            break;
        }
        else if (*x >= scan_map.width) {
            *x = scan_map.width;
            break;
        }
    }
    return hit;
}


int Scanner::scan_row(int x, int y, int *sweep_xs, int *sweep_xe) {
    //s_x
    int step_x = y >= border ? 2 : 1;
    if (step_x > 1) x &= 0xfe;
```

```
    int s_x = x;
    bool hit = get_pos(&s_x, y, *sweep_xe, -1, true);

    if (!hit) {
        hit = get_pos(&s_x, y, *sweep_xe, +1, false);
    } else {
        if (!classify(s_x, y)) s_x += step_x;
    }

    if (!hit || s_x >= scan_map.width ) return 0;


    //e_x
    int e_x = s_x;
    get_pos(&e_x, y, scan_map.width, +1, true);

    *sweep_xs = s_x <= scan_map.width ? s_x : scan_map.width;
    *sweep_xe = e_x <= scan_map.width ? e_x : scan_map.width;

    if (x < s_x) //ex: x00s_x11e_x0 => 60011100
        scan_map.tag(x, y, e_x-x);
    else         //ex: 000s_x1xe_x0 => 00031100
        scan_map.tag(s_x, y, e_x-s_x);

    return e_x-s_x;
}

int Scanner::get_area(int *center_x, int *center_y, int *x_width, int *y_height)
{
    //cout << endl;
    int area = 0;
    int y = y_pos;
    int x_s = 0, x_e = scan_map.width;
    int x = 0;

    *x_width = *y_height = 0;

    if (y >= height) return 0;

    int max_x = 0, min_x = WIDTH;

    while (1) {
        int step_y = y >= border ? 2 : 1;

        //++iter;
        int aa = scan_row(x,y, &x_s, &x_e) * step_y;
        area += aa;

        if (aa) {
            if (x_s < min_x) min_x = x_s;
            if (x_e > max_x) max_x = x_e;
        }

        if (aa > *x_width) *x_width = aa;

        if (area && x_s == x_e ) break;

        if ( (x_s >= scan_map.width || x_s == 0) && x_e >= scan_map.width) {
            y_pos += step_y;
```

```
            if (area)
                break;
        }
        x = x_s;
        //x = x_s + aa/2;//börja sökning på nästa rad i mitten istället för x_s
        if (x < 0 || x >= scan_map.width) {x = 0;}
        y += step_y;
        if (y >= height) break;
    }
    if (area) {
        int y_len = y - y_pos;
        *y_height = y_len;
        *center_y = y_pos + y_len/2;
        *center_x = min_x + (max_x-min_x)/2;
    }
    return area;
}

void Scanner::new_scan() {
    scan_map.new_scan();
    y_pos = 1;
}
```

```
/*
 * state.h
 */

#ifndef __STATE_H
#define __STATE_H

enum State {
    FIND_BALL,
    BALL_AHEAD,
    FIND_GOAL,
    GOAL_AHEAD,
    MOVE_AROUND,
    GROUND_AHEAD,
    GO_PAST_BALL,
    FIND_MY_GOAL,
};


enum SearchType {
    BALL,
    YELLOW_GOAL,
    BLUE_GOAL,
    GROUND
};

#endif
```