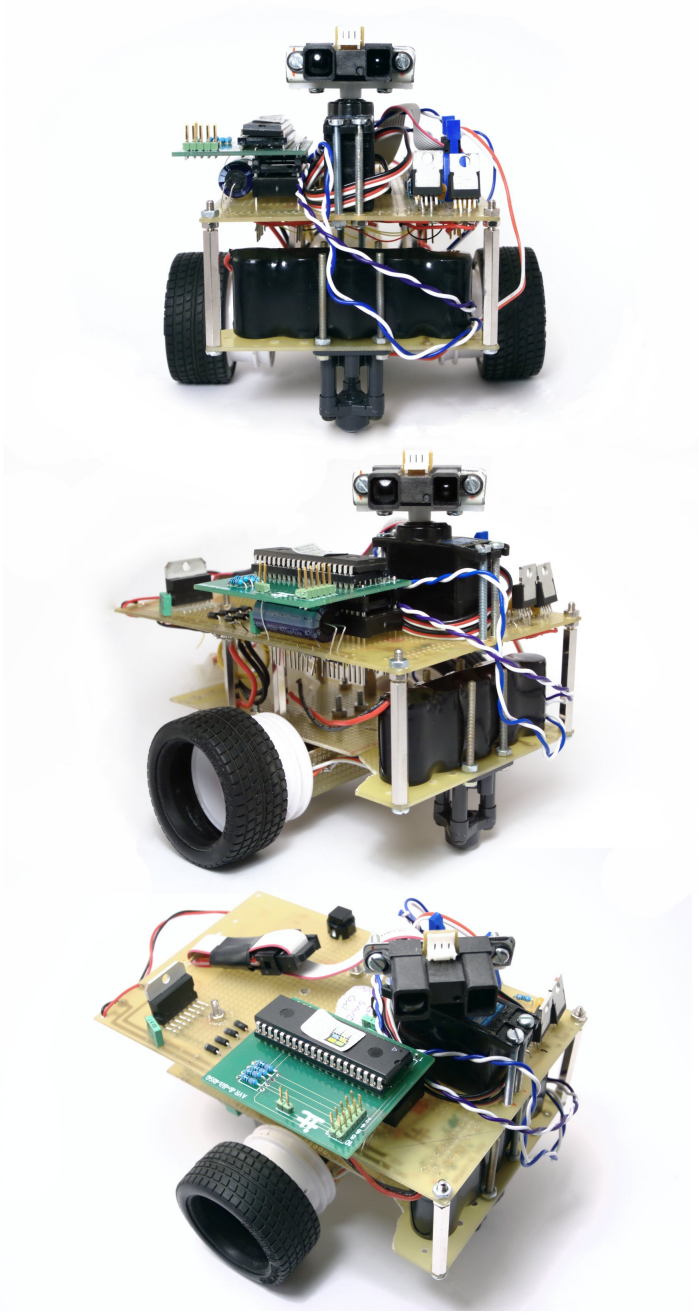
Arne Dito

Projektrapport
Digitala Projekt EDI022

Grupp DIGPSK2
Leo Bärring E06 870120-4177
Jakob Hägg E06 870518-3518

27 maj 2009



Abstract

We have designed and built a mobile robot which purpose is to navigate an unknown area and building a map of it as it discovers obstacles. The project contains both communication protocol design, for exchanging information between the base station and the mobile unit, as well as more mechanic aspects of robot building. We have successfully managed to construct a platform on which more sofisticated algorithms—like path finding—can be implemented, but we find the precision of the movement somewhat lacking, mostly due to the motor being difficult to control.

Innehåll

Kapitel 1

Inledning

Som projektarbete har vi valt att utveckla en robot som har till uppgift att bland annat köra runt och mäta avstånd. Den information som roboten samlar in ska den sedan skicka trådlöst till en dator som plottar denna data på ett lämpligt sätt. För att kunna realisera detta var vi tvungna att bygga projektet runt tre delar. Första delen är själv roboten, andra delen är en kommunikationsbrygga mellan datorn och roboten och sista själva datorn med huvudprogrammet. Vi har valt att använda MATLAB som program för att styra roboten och plotta avståndsdatan.

Kapitel 2

Kravspecifikation

Målet är att konstruera en robot som kan köra omkring i ett ”okänt” utrymme, detektera och undvika hinder, samt kommunicera information trådlöst om dessa till en dator, som i sin tur kan använda datan för att bygga upp en karta av området.

Det krävs även en modul som tar emot radiosignalerna på datorsidan utför viss grundläggande bearbetning (t.ex. felhantering) och kommunicerar dessa till datorn. På grund av att kommunikationen sker över radio, hade det varit fördelaktigt med tvåvägskommunikation, eftersom man då kan skicka tillbaka information från datorn om huruvida sänd data har kommit fram oskadad.

Roboten ska innehålla följande:
Atmel Atmega16 som controller
Avståndssensor, för att detektera hinder
Servo för att rikta avståndssensor
Hjulplattform med växellåda och motorer
Motordrivkrets
Trådlös kommunikation med dator (robot->dator eller, helst, tvåvägs)
Optiska sensorer för att mäta hjulens rullning

Hålla koll på sin position Avståndsdetektering+vinkel, (varpå en koordinat, för det detekterade objektet kan beräknas trigonometriskt) Protokoll med felkorrigering för att överföra data mellan mobil och fast enhet, hur omfattande detta behöver göras beror på hur mycket som redan finns inbyggt i transcievermodulen. (På datorn) Program för kommunikation med sändar/-mottagrenheten och beräkning av hinderpunkters position och presentera dessa grafiskt.

I mån av tid: - Detektera om roboten återvänder till tidigare undersökt område och i så fall välja en annan riktning (än att fortsätta i samma bana), eller med lämplig algoritm välja en väg till okända fält.

Kapitel 3

Roboten (Arne)

3.1 Viktiga komponenter

- Atmega 16 Mikrokontroller (16MHz 16kiB programminne, 1kiB RAM-minne)
- Motorpaket med hjul
- Standard hobbyservo
- Analog IR-baserad avståndsensor
- Radiomodul
- H-brygga
- 5V och 3,3V spänningsregulatorer

3.2 Motorpaket och motorstyrning

Framdrivningen av Arne realiserades med hjälp av ett färdigt motorpaket från Tamiya. Motorpaketet är uppbyggt av två likströms-motorer som via två separata växellådor driver de två hjulen. Motorerna drivs direkt från batteriet via en H-brygga (ST L298) vars insignaler genereras av Arnes mikrocontroller. Även i nerväxlat läge gick motorerna för fort för vårt behov och krav på precision. Lösningen på detta problem blev att pulsbreddsmodulera ingångarna på H-bryggan som styr av- och påslag av motorerna. Den pulsbreddsmodulerade signalen generades med hjälp av mikrocontrollers inbygg-

da 8-bitars timers och motorerna modulerades med separata signaler i försök att jämna ut växellådornas olika motstånd och därmed få en jämnare gång.

Då kontroll på färdriktning och körlängd är av största vikt i vårt projekt var Arnes rörelser tvungna att mätas och styras på ett noggrant sätt. Ett försök i att åstadkomma detta var att montera positionshjul och läsgafflar på hjulens axlar. Läsgafflen som användes heter ITR8102 och positionshjulen är tagna direkt ur en gammal kulmus från Microsoft. Lämpliga motstånd på ca 10 kΩ sattes i serie med läsgafflarnas lysdioder för att få till en lagom stark belysning och läsgafflarnas utsignal kopplades in till mikrocontrollers avbrottsingångar via en Schmitt-trigger för att generera en digital signal.

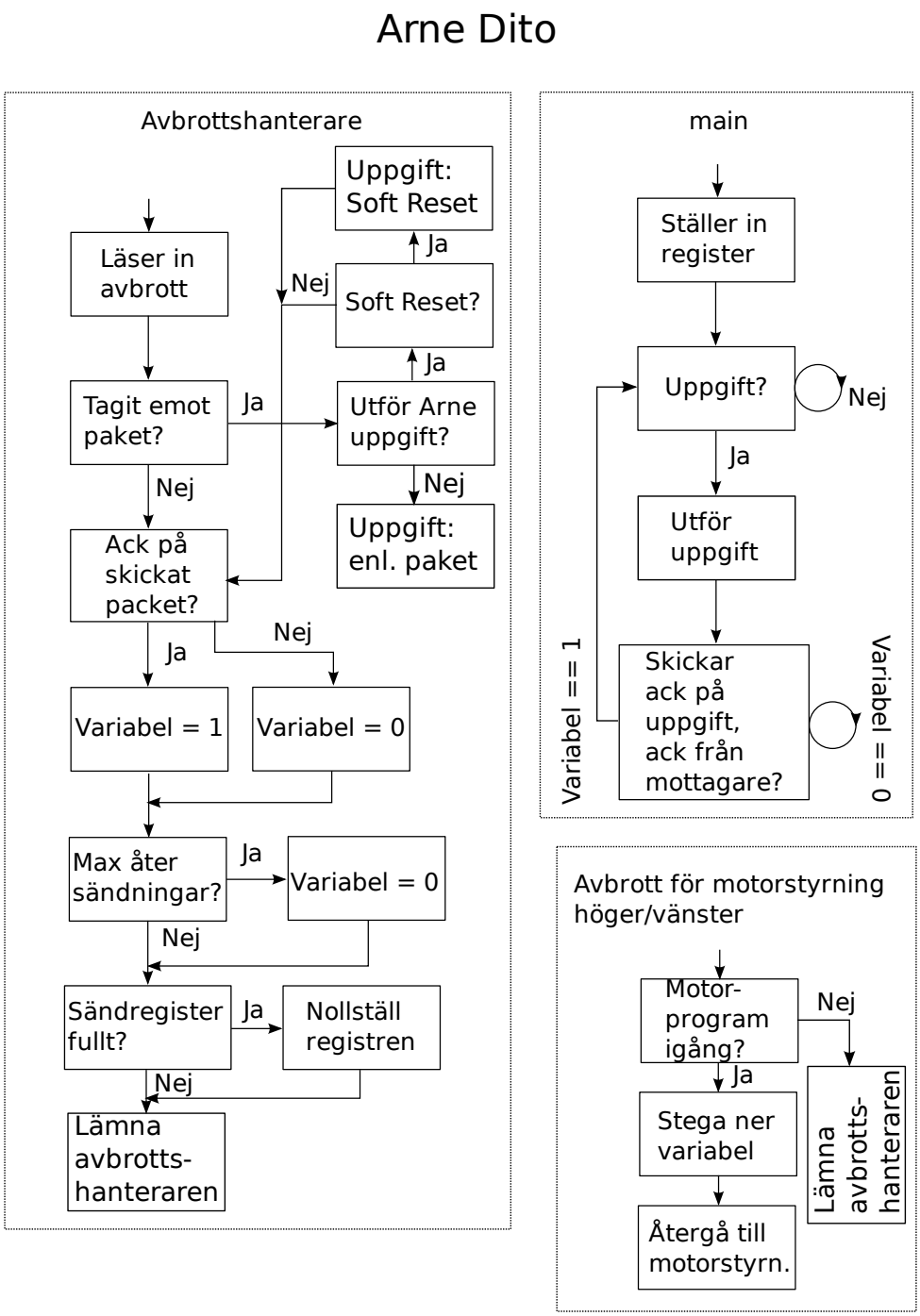
Då Arne ska köra framåt eller bakåt väljs först hur långt han ska gå, denna längd sparas som två separata värden för varje hjul och dessa värden stegas sedan ner efter varje avbrott som genereras då läsgafflarnas strålar antingen bryts eller går igenom positionshjulet. Positionshjulen har 35 hål vilket då motsvarar 70 avbrott per varv. Om det ena hjulet ligger före det andra med mer än en viss gräns vilken sätts i programmet kommer det hjulet att stoppas för en kort stund så att det andra hjulet hinner ikapp. Då Arne ska svänga höger eller vänster väljs antal steg på samma sätt men nu sätts styrsignalerna för H-bryggan så att hjulen går i olika riktning. Detta eftersom Arne då kommer stå på samma position som innan men nu med en annan riktning. Detta medför enklare uträkningar för Arnes rörelser. För att undvika att hjulen spinner vid start föregås alla olika rörelser av en mjukstart där moduleringens duty cycle ökas från halva duty cyclen upp i linjära steg till den fulla duty cyclen. Hjulspinn kommer annars att medföra fel i uträkningen av hans position.

3.3 Avståndsmätningar

För att mäta avstånd används en analog IR-modul från Sharp vid namn 2Y0A02. Utsignalen från sensorn varierar mellan 0 och ca 3 V och för att få så hög upplösning och noggrannhet som möjligt vid A/D-omvandlingen användes 3,3 V regulatorn som spänningsreferens och alla A/D-omvandlarens tio bitar användes. För möjlighet att kunna ändra sensorns riktning monterades den på ett servo. Under avståndsmätningarna är det inte bara avståndet som är intressant utan även vinkel i förhållande till Arnes färdriktning måste vara känt. Under ett svep sattes därför riktningarna till bestämda vinklar från 20 – 160 grader i steg om 7 grader.

3.4 Arnes mjukvara

Huvudtanken för skrivandet av kod för Arne är att han hans beslut fullständigt ska baseras från de kommandon som MATLAB-programmet ger honom. Flödesschemat i figur 3.1 är tänkt att ge en bild av hur koden är uppbyggd, den är dock något förenklad.



Figur 3.1: Flödesschema över programmet som körs på Arne.

Kapitel 4

Kommunikationssystem

4.1 Trådlöst och trådbundet

Mikrokontrollern som sitter på Arne har inte resurser till att behandla den data vi vill samla in på egen hand. Man skulle kunna tänka sig någon form av externt minne för att lagra datapunkter, men beräkningskapaciteten är ändå begränsad. Dessutom ville vi kunna presentera datan som samlats in. Det ideala verktyget för detta är en dator som, jämfört med en mikrokontroller, har obegränsat med minne och är hur snabb som helst. För att kommunicera med datorn fick vi två radiomoduler för den trådlösa delen. Den ena integrerades med Arne, och den andra på en stationär enhet med ytterligare en mikrokontroller av samma typ. Enheten bestyckades också med en MAX232CPE för att kunna kommunicera med datorn via dess serieport och mikrokontrollerns USART-pinnar. Tre lysdioder används för att indikera statusen på kommunikationen.

4.2 Radio

Radiomodulerna är baserade på kretsen NRF24L01 från Nordic Semiconductor, dessa arbetar på det fria 2,4GHz-bandet d.v.s. samma som WLAN. Denna kontrolleras med kommandon skickade över SPI. Modulen är väldigt lättanvänd och implementerar själv OSI-nivå ett och två, med data uppdelade i paket, grundläggande addressering och CRC för kontroll av dataintegritet. Dessutom håller sändaren koll på om den får en bekräftelse från mottagaren på att paketet mottagits, är intakt och sänder annars paketet ännu en gång.

4.3 Protokoll

Kommunikationen mellan dator och Arne sker på enklast möjliga sätt, data skickas och tas emot mellan datorn och den stationära enheten i paket med samma längd som ett radiopaket. Vid uppstart lyssnar den stationära enheten endast på datorn, när 32 bytes har mottagits sänds dessa över radio, och sedan väntar enheten på ett svarspaket. Alltså är kommunikationsriktningen hårt styrd och för varje paket i ena riktningen, skickas ett i motsatt riktning. Detta som ett resultat av att försöket att implementera ett mer flexibelt (och därmed avancerat) kommunikationssystem stött på problem med att felsökning och design av denna del tog för lång tid. Eftersom vi parallellt med denna process bestämde oss för att i princip göra all databehandling och helt och hållet implementera styrsystemet på datorsidan så är det egentligen ingen nackdel förutom kanske vid hög belastning, men eftersom kraven på throughput är marginella är detta inget problem. Dessutom är serieportskommunikationen begränsad till 9600 baud på grund av att det är den högsta hastighet som går att implementera med tillräckligt stor felmarginal med vald klockhastighet på mikrokontrollern som finns listad som valbar hastighet på datorn. Radiomodulerna å andra sidan jobbar på upp till 2Mb/s.

4.4 Paket

Paketformatet som används har en fast längd på 32 bytes, eftersom detta är den största mängden data som kan skickas utan att radion behöver bryta upp meddelandet i flera delpaket. Den första byten i paketet fungerar dels som ett kommando om det skickas från datorn, samt avgör vad resten av paketet innehåller.

`packet[0]`	Betydelse
`0x00`	soft reset
`0x01`	echo, skicka tillbaka samma paket, test av radiolänken
`0x02`	kör framåt x antal steg
`0x03`	kör bakåt x antal steg
`0x04`	rotation vänster
`0x05`	rotation höger
`0x06`	avståndsmätningssekvens 1
`0x07`	avståndsmätningssekvens 2
`0x08`	mätdata 1
`0x09`	mätdata 2
`0x0a`	ändra duty cycle för vänster och höger motor

Kapitel 5

Styrsystem

För tillfället är styrsystemet väldigt enkelt, allting sköts av huvudprogrammet som körs på datorn. Programmet är baserat på en huvudloop där området framför och delvis brevid Arne scannas med avståndssensorn. Efter detta skickas resultatet över till datorn och om området direkt framför verkar fritt förflyttar Arne sig 25cm framåt. Annars jämförs området framför till höger respektive vänster och Arne roterar åt det håll som erbjuder störst utrymme, varpå en ny avskanning utförs.

Avståndet som rapporteras är ett tio bitars värde, som vi omvandlar till centimeter i MATLAB med hjälp av ett polynom som vi tidigare konstruerat under en kalibreringsomgång genom att göra mätningar centimetervis, plotta avståndet som en funktion av det uppmätta värdet och göra en polynomapproximation.

All mätdata som insamlas lagras i en `struct`-vektor i MATLAB som sparar Arnes position, vinkel samt den relativa vinkeln och avståndet för varje mätning. Datan kan senare omvandlas till x,y-positioner för respektive mätpunkt och på detta sättet kan en karta över området konstrueras. I MATLABkoden har en nivå-terminologi införts. Nivå noll är mätdatan precis som den kommer från Arne, alla mätvinklar är relativt Arnes vinkel och så vidare. På nivå ett har mätpunkterna fått absoluta vinklar (relativt Arnes utgångsvinkel) och även x och y-koordinater har beräknats för punkterna.

En tredje men i koden icke namngiven data-nivå finns också, där vektordatan har lagts in i en matris där varje index motsvarar en ruta och värdet beror på om och hur många hinderpunkter inom rutan som uppmätts, om några mätvektorer passerat området, men inte registrerat något hinder (detta avgörs med Bresenhams linjealgoritm), eller om Arne har kört igenom rutan.

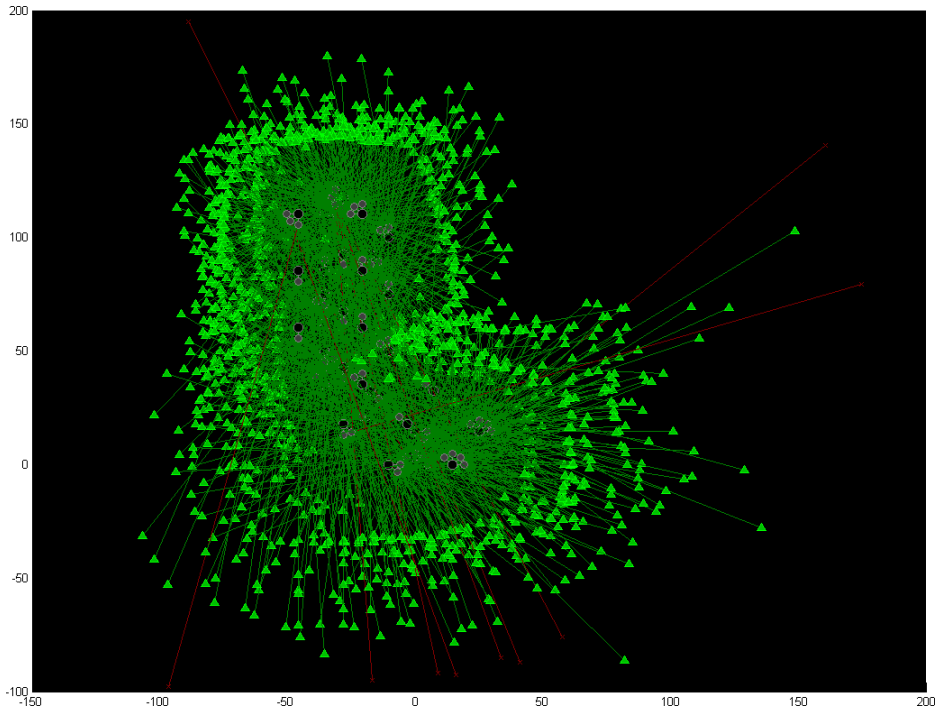
Denna omvandling (vektor till raster) är viktig om man vill kunna implementera någon form av navigeringsalgoritm (t.ex. A*) för att ta sig mellan olika punkter. I skrivande stund har en sådan funktion inte implementerats, men förutsättningarna finns.

Kapitel 6

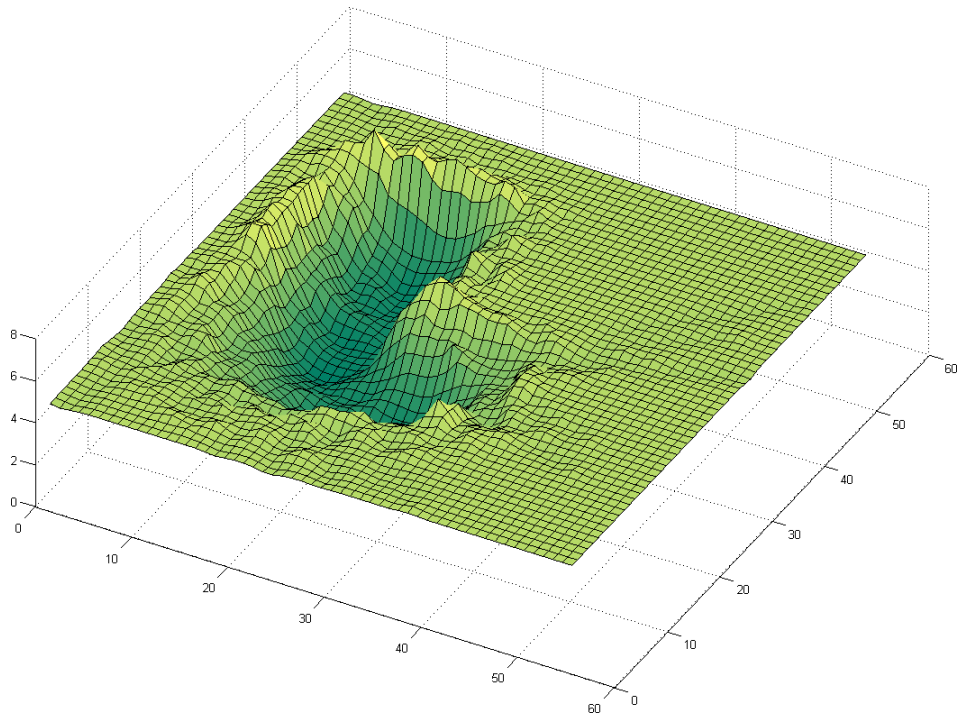
Visualisering av insamlad data

Samtidigt som huvudprogrammet kör uppdateras en plot i MATLAB som visar Arnes position relativt utgångsläget och ritar ut linjer från Arne i de riktningar som mäts från den aktuella positionen. Plotfunktionen arbetar direkt på den insamlade datan och är egentligen en konverteringsfunktion mellan nivå noll och ett, även om en annan funktion för detta används. När datan har konverterats till nivå tre, representeras den av en matris med topografiska data eller hur man nu vill se det, och kan alltså ritas upp med vilken MATLABfunktion som helst som accepterar en sådan (t.ex. `surf()` eller `contourf()`). Detta kan dock inte för tillfället göras i realtid, eftersom övergången mellan vektorer och raster medför att man begränsar största möjliga område som kan kartläggas. I realiteten skulle man dock kunna definiera ett tillräckligt stort raster, så att gränserna ligger utanför aktionsradien, antingen på grund av fysiska hinder som väggar, eller på grund av batteripaketets livslängd innan det behöver laddas.

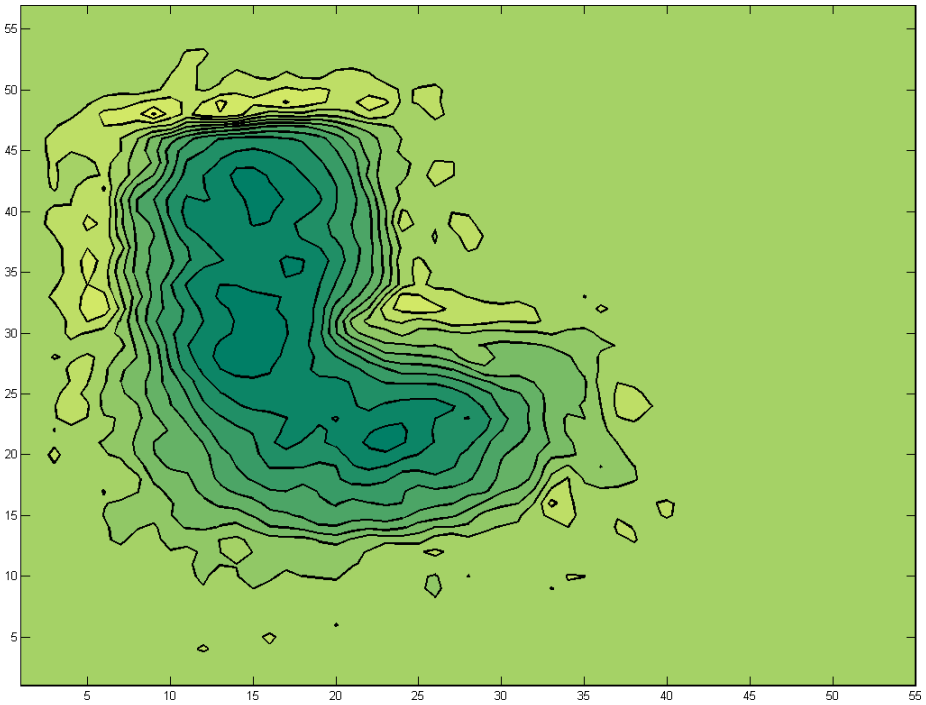
Tre stycken plottar följer här, figur 6.1 visar en bild av hur mätdatan ritas upp i realtid under datainsamlingen. Giltiga data mäts ritas ut med en grön linje som avslutas i en triangel. Data där värdet motsvarar ett avstånd på gränsen till avståndssensorns område kasseras och markeras med ett röd linje och ett kryss. I denna bilden är så gott som alla värden giltiga, eftersom mätningen skedde på en sluten och ganska liten testbana (se foto i figur 6.4. Figur 6.2 och 6.3 visar på hur `smoothplot()` ritar datan. Notera att man i figur 6.1 kan urskilja en viss avdrift. Denna beror på onoggrannheten i motorsystemet.



Figur 6.1: Hur Arne uppfattar världen



Figur 6.2: En tredimensionell bild över hindertätheten (`surf()`).



Figur 6.3: Ett annat sätt att visualisera mätdatan (`contourf()`).

Figur 6.4: Ett fotografi av hinderbanan.

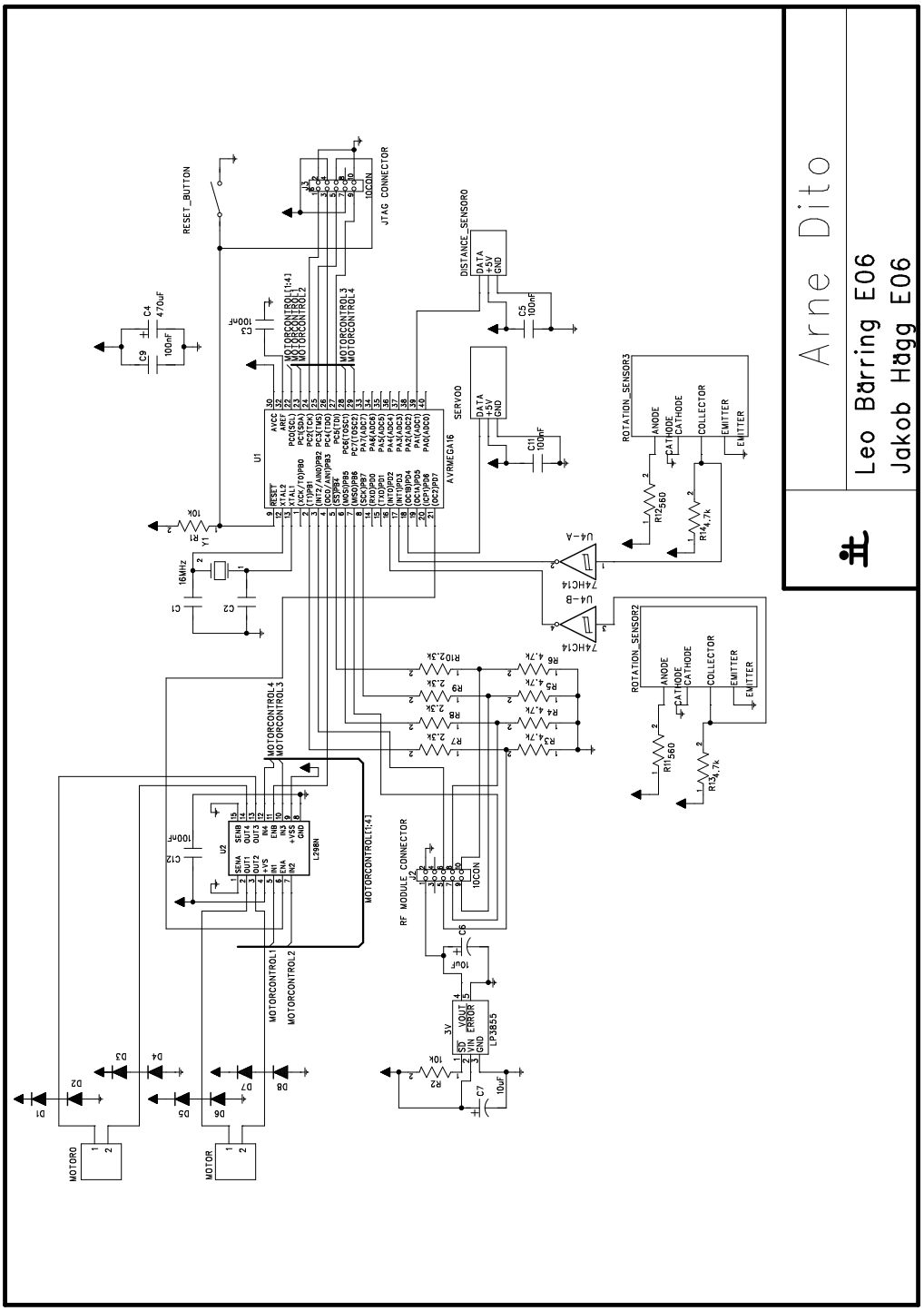
Kapitel 7

Resultat och diskussion

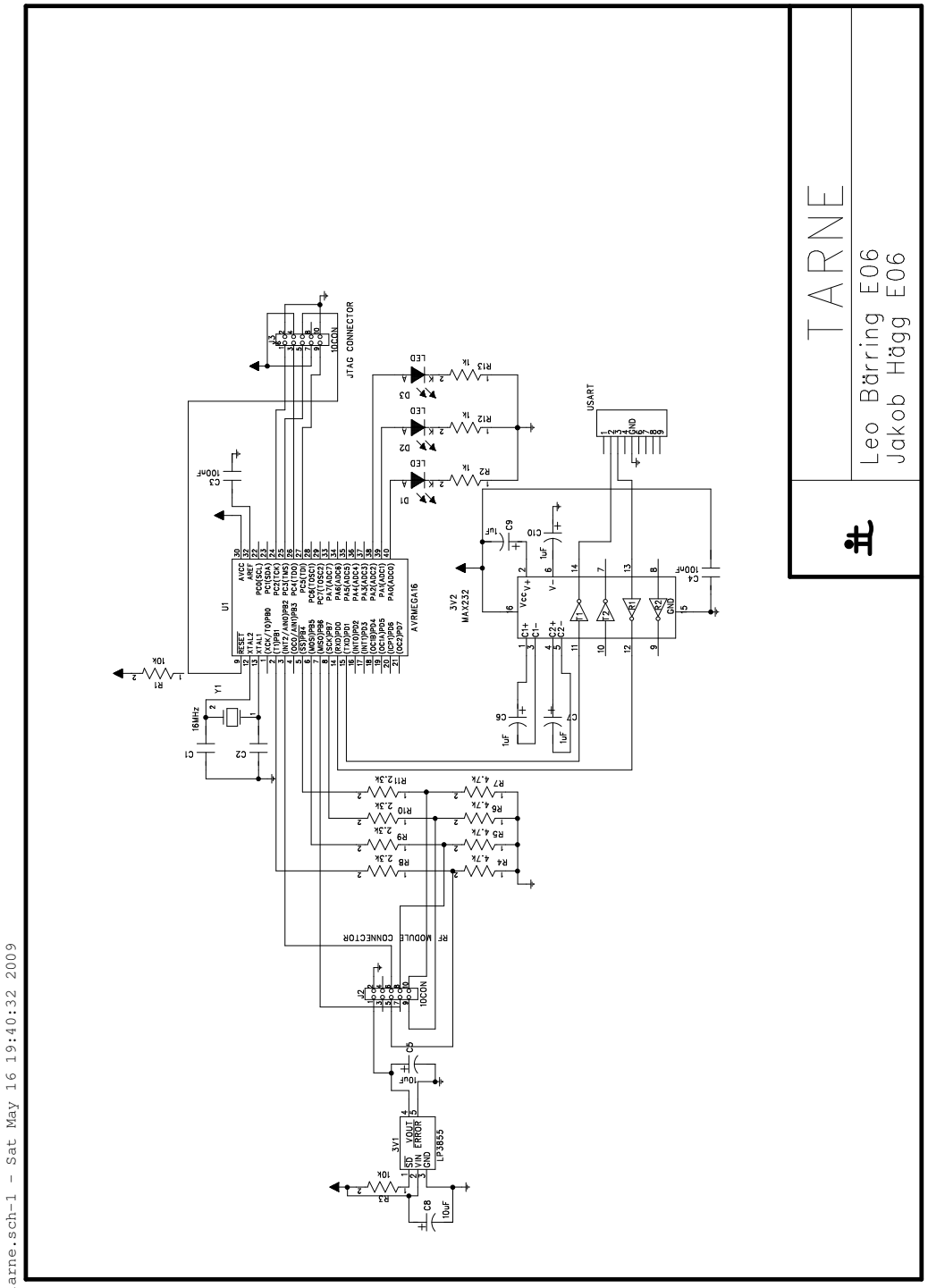
Till en början var tanken att Arne själv skulle besluta var och hur den skulle gå och kontinuerligt skicka data till datorn. Problemet med detta var bland annat att det krävde större krav på kommunikationen. Och efter många försök att implementera ett lämpligt kommunikationsprotokoll mellan datorn och kommunikationsbryggan visade det sig vara för svårt och dessutom långsamt. Detta ledde till vi istället lät MATLAB bestämma Arnes rörelser vilket i sig inte var någon förlust. Andra problem som vi också hade var att motorpaketet visade sig ha alldeles för dålig precision för våra krav. Detta innebar att det var bättre att använda en extra timer för att styra den andra motorn som försök att kompensera för att det är olika mycket friktion i de olika växellådorna. Ett annat problem som vi hade var att läsgafflarna inte gick att använda med positionshjul med många små hack utan vi fick använda oss av positionshjul med litet färre hack vilket påverkade precisionen negativt. Att köra framåt och bakåt för Arne fungerar ganska bra med avseende på precision i riktning och steglängd. Glappet i växellådorna verkar dock göra att precisionen blir något sämre när Arne ska svänga vänster eller höger. Sammanfattningsvis kan man säga att vi har lyckats med dom övergripande målen med projektet, datan som samlas in gör det möjligt att skapa en virtuell bild av utrymmet kring Arne och den kan röra sig fritt på en plan yta och undvika hinder. Vi har än så länge inte kunnat avsöka något större område, delvis på grund av att ackumulatorpaketet inte räcker. Ett annat problem är att Arne har en förmåga att efter en tids mätningar frysa eller rapportera ogiltiga mätdata. Detta skulle eventuellt kunna förklaras med strömförsörjningsproblem, eller glappkontakt på ett eller flera ställen. Det händer också ibland att Arne kommer ur kurs vilket gör att plotten blir skev, detta är dock till viss del en fråga om kalibrering.

Bilaga A

Ritningar



Figur A.1: Elektriskt schema över Arne.



Figur A.2: Elektriskt schema över Tarne.

Bilaga B

Källkod

B.1 Arne

B.1.1 Huvudprogrammet

```
1  #include "../movement/movement.h"
2  #include "../radio/radio.h"
3  #include "../distance_sensor/distance.h"
4  #include "../servo/servo.h"
5  #include "../motor/motor.h"
6  #define F_CPU 16000000
7  #include <util/delay.h>
8  #include <avr/io.h>
9  #include <avr/wdt.h>
10
11 #define soft_reset()        \
12 do                          \
13 {                          \
14     wdt_enable(WDTO_15MS);  \
15     for(;;)                 \
16     {                       \
17     }                       \
18 } while(0)
19
20 #define SOFT_RESET                  0x00
21 #define ECHO                        0x01
22 #define FORWARD                     0x02
23 #define BACKWARD                    0x03
24 #define TURN_LEFT                   0x04
25 #define TURN_RIGHT                  0x05
26 #define STANDARDSCANSEQ1            0x06
27 #define STANDARDSCANSEQ2            0x07
28 #define SCANDATA1                   0x08
29 #define SCANDATA2                   0x09
30 #define CHANGE_DUTY                 0x0a
31
32 volatile uint8_t state = 255;
33 volatile uint8_t status;
```

```
34 volatile uint8_t transmitted;
35 volatile uint8_t again;
36 volatile uint8_t packet[32];
37 volatile uint8_t recievedpacket[32];
38
39 int main() {
40
41         sei(); // Aktivera avbrott
42         _delay_ms(20);
43         radio_setup(32, 33); // 32 bytes paket och radiokanal 33
44         radio_config_rx(); // Starta i mottagar lage
45         CE_HIGH(); // CHIP ENABLE pa radiomottagaren
46         distance_setup(); // Avstandsensor installningar
47         servo_setup(); // Servo installningar
48         movement_setup(); // Avbrottsinstallningar for lasgafflar
49         motor_setup(); // Installningar for motorstyrning
50         servo_turnback(); // Stall servot i ursprungslage
51         _delay_ms(200);
52         motor_duty_left(172); // Duty max = 255
53         motor_duty_right(177);
54         uint8_t distance_value_sensor1[22];
55         uint8_t distance_value_sensor2[20];
56         uint8_t standard_scan_duty_sensor1a[11] = {0x04,0x05,0x07,0x08,0x09
               ,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f};
57         uint8_t standard_scan_duty_sensor1b[11] = {0xd0,0xec,0x09,0x26,0x42
               ,0x5f,0x7c,0x98,0xb5,0xd2,0xee};
58         uint8_t standard_scan_duty_sensor2a[10] = {0x0f,0x0e,0x0d,0x0c,0x0a
               ,0x09,0x08,0x07,0x06,0x05};
59         uint8_t standard_scan_duty_sensor2b[10] = {0x60,0x43,0x27,0x0a,0xed
               ,0xd1,0xb4,0x98,0x7b,0x5e};
60         uint8_t distance_index, i, b, distance_index_extra;
61         distance_index = 2;
62         distance_index_extra = 0;
63         servo_flip1(); // For koll att ARNE ar omstartad
64         for(;;) {
65
66                 /* State variabeln ar normalt satt till 255 forutom da Arne
                       har blivit atsagd att
67                 utfora nagot */
68                 switch(state) {
69                         case ECHO:
70                                 packet[0] = ECHO;
71                                 CE_LOW();
72                                 radio_config_tx();
73                                 /* Skickar ivag paketet sa lange mottagaren
                                       inte skickar tillbaka ett ack */
74                                 do {
75                                         again = 0;
76                                         radio_transmit(&packet[0]);
77                                         while(again != 1);
78
79                                 } while(transmitted != 1);
80                                 radio_config_rx();
81                                 CE_HIGH();
82                                 state = 255;
83                                 break;
84
85                         case STANDARDSCANSEQ1: // Forsta scansekvensen gar
                               fran 20 till 160 grader i steg om 14 grader
86
87                                 for(i = 0; i < 11; i++) {
88
```

```
89                                  servo_turn(1,
                                        standard_scan_duty_sensor1a[i],
                                        standard_scan_duty_sensor1b[i]);
90                                  _delay_ms(100);
91                                  distance_readsensor(1);
92                                  while((ADCSRA & (1<<ADIF)) == 0); //
                                        Vantar pa att AD-omvandlingen
                                        ska bli fardig
93                                  distance_value_sensor1[
                                        distance_index_extra + 1] = ADCL
                                        ;
94                                  distance_value_sensor1[
                                        distance_index_extra] = ADCH;
95                                  ADCSRA |= (1<<ADIF); // Nollstaller
                                        interrupt flaggan
96                                  distance_index_extra += 2;
97
98                              }
99                              packet[0] = 0x07;
100                             packet[1] = 11;
101                             for(b = 0; b < 2*11; b++) {
102
103                                     packet[b+distance_index] =
                                            distance_value_sensor1[b];
104
105                             }
106                             distance_index_extra = 0;
107                             CE_LOW();
108                             radio_config_tx();
109                             /* Skickar ivag paketet sa lange mottagaren
                                    inte skickar tillbaka ett ack */
110                             do {
111                                     again = 0;
112                                     radio_transmit(&packet[0]);
113                                     while(again != 1);
114
115                             } while(transmitted != 1);
116                             radio_config_rx();
117                             CE_HIGH();
118                             state = 255;
119                             break;
120
121                     case STANDARDSCANSEQ2: // Andra scansekvensen som
                            gar fran 153 till 27 grader i steg om 14 grader
122
123                             for(i = 0; i < 10; i++) {
124
125                                     servo_turn(1,
                                            standard_scan_duty_sensor2a[i],
                                            standard_scan_duty_sensor2b[i]);
126                                     _delay_ms(100);
127                                     distance_readsensor(1);
128                                     while((ADCSRA & (1<<ADIF)) == 0); //
                                            Vantar pa att AD-omvandlingen
                                            ska bli fardig
129                                     distance_value_sensor2[
                                            distance_index_extra + 1] = ADCL
                                            ;
130                                     distance_value_sensor2[
                                            distance_index_extra] = ADCH;
131                                     ADCSRA |= (1<<ADIF); // Nollstaller
                                            interrupt flaggan
```

```
132                                     distance_index_extra += 2;
133
134                                 }
135                                 servo_turn(1,standard_scan_duty_sensor1a[0],
                                        standard_scan_duty_sensor1b[0]);
136
137                                 packet[0] = 0x08;
138                                 packet[1] = 10;
139                                 for(b = 0; b < 2*10; b++) {
140
141                                     packet[b+distance_index] =
                                            distance_value_sensor2[19-b];
142
143                                 }
144
145                                 distance_index_extra = 0;
146                                 CE_LOW();
147                                 radio_config_tx();
148                                 /* Skickar ivag paketet sa lange mottagaren
                                        inte skickar tillbaka ett ack */
149                                 do {
150                                     again = 0;
151                                     radio_transmit(&packet[0]);
152                                     while(again != 1);
153
154                                 } while(transmitted != 1);
155                                 radio_config_rx();
156                                 CE_HIGH();
157                                 state = 255;
158                                 break;
159
160                             case FORWARD: // Arne kor framat i packet[1] antal
                                    steg
161
162                                 motor_step(packet[1],packet[1],0);
163                                 packet[0] = 0x06;
164                                 packet[1] = 0x02;
165                                 CE_LOW();
166                                 radio_config_tx();
167                                 /* Skickar ivag paketet sa lange mottagaren
                                        inte skickar tillbaka ett ack */
168                                 do {
169                                     again = 0;
170                                     radio_transmit(&packet[0]);
171                                     while(again != 1);
172
173                                 } while(transmitted != 1);
174                                 radio_config_rx();
175                                 CE_HIGH();
176                                 state = 255;
177                                 break;
178
179                             case BACKWARD: // Arne kor bakat i packet[1] antal
                                    steg
180
181                                 motor_step(packet[1],packet[1],1);
182                                 packet[0] = 0x06;
183                                 packet[1] = 0x03;
184                                 CE_LOW();
185                                 radio_config_tx();
186                                 do {
187                                     again = 0;
```

```
188                                     radio_transmit(&packet[0]);
189                                     while(again != 1);
190
191                             } while(transmitted != 1);
192                             radio_config_rx();
193                             CE_HIGH();
194                             state = 255;
195                             break;
196
197
198
199                     case TURN_LEFT: // Arne svanger vanster i packet[1]
                            antal steg
200
201                             motor_step(packet[1],packet[1],2);
202                             packet[0] = 0x06;
203                             packet[1] = 0x04;
204                             CE_LOW();
205                             radio_config_tx();
206                             /* Skickar ivag paketet sa lange mottagaren
                                    inte skickar tillbaka ett ack */
207                             do {
208                                     again = 0;
209                                     radio_transmit(&packet[0]);
210                                     while(again != 1);
211                             } while(transmitted != 1);
212                             radio_config_rx();
213                             CE_HIGH();
214                             state = 255;
215                             break;
216
217                     case TURN_RIGHT: // Arne svanger vanster i packet[1]
                             antal steg
218
219                             motor_step(packet[1],packet[1],3);
220                             packet[0] = 0x06;
221                             packet[1] = 0x05;
222                             CE_LOW();
223                             radio_config_tx();
224                             /* Skickar ivag paketet sa lange mottagaren
                                    inte skickar tillbaka ett ack */
225                             do {
226                                     again = 0;
227                                     radio_transmit(&packet[0]);
228                                     while(again != 1);
229
230                             } while(transmitted != 1);
231                             radio_config_rx();
232                             CE_HIGH();
233                             state = 255;
234                             break;
235
236                     case SOFT_RESET: // Mjukvaru aterstallning av Arne
237                             packet[0] = 0x06;
238                             packet[1] = 0x08;
239                             CE_LOW();
240                             wdt_enable(WDTO_15MS);
241                             radio_config_tx();
242                             do {
243                                     again = 0;
244                                     radio_transmit(&packet[0]);
245                                     while(again != 1);
```

```
246
247                         } while(transmitted != 1);
248                         radio_config_rx();
249                         CE_HIGH();
250                         state = 255;
251                         cli();
252                         soft_reset();
253                         break;
254
255                 /*case 10: // AD test for avstandssensorn
256
257                         packet[0] = 0x0b;
258                         distance_readsensor(1);
259                         while((ADCSRA & (1<<ADIF)) == 0); // Wait
                                for A/D conversion to finish
260                         packet[2] = ADCL;
261                         packet[1] = ADCH;
262                         ADCSRA |= (1<<ADIF); // Clear interrupt flag
263                         CE_LOW();
264                         radio_config_tx();
265                         do {
266                                 again = 0;
267                                 radio_transmit(&packet[0]);
268                                 while(again != 1);
269
270                         } while(transmitted != 1);
271                         radio_config_rx();
272                         CE_HIGH();
273                         state = 255;
274                         break;*/
275
276                 case CHANGE_DUTY: // For mojlighet att andra
                        motorernas duty cycle via MATLAB
277
278                         OCR1BH = packet[1];
279                         OCR1BL = packet[2];
280                         packet[0] = 0x06;
281                         packet[1] = 0x0c;
282                         CE_LOW();
283                         radio_config_tx();
284                         do {
285                                 again = 0;
286                                 radio_transmit(&packet[0]);
287                                 while(again != 1);
288
289                         } while(transmitted != 1);
290                         radio_config_rx();
291                         CE_HIGH();
292                         state = 255;
293                         break;
294
295                 case 13: // Staller in servo enligt packet[1] och
                        packet[2]
296
297                         OCR0 = packet[1];
298                         OCR2 = packet[2];
299                         packet[0] = 0x06;
300                         packet[1] = 0x0d;
301                         CE_LOW();
302                         radio_config_tx();
303                         do {
304                                 again = 0;
```

```
305                                     radio_transmit(&packet[0]);
306                                     while(again != 1);
307
308                                 } while(transmitted != 1);
309                                 radio_config_rx();
310                                 CE_HIGH();
311                                 state = 255;
312                                 break;
313
314                             case 14: // Kor framat i 3 sekunder (for bestamning
                                    av de olika duty cyclens)
315
316                                 motor_forward();
317                                 _delay_ms(3000);
318                                 motor_stop();
319                                 packet[0] = 0x06;
320                                 packet[1] = 0x0e;
321                                 CE_LOW();
322                                 radio_config_tx();
323                                 do {
324                                     again = 0;
325                                     radio_transmit(&packet[0]);
326                                     while(again != 1);
327
328                                 } while(transmitted != 1);
329                                 radio_config_rx();
330                                 CE_HIGH();
331                                 state = 255;    break;
332
333                                 case 15: // Test av olika servovinklar
334
335                                 if(packet[1] == 0) {
336
337                                     servo_turn(1,
                                        standard_scan_duty_sensor1a[
                                        packet[2]],
                                        standard_scan_duty_sensor1b[
                                        packet[2]]);
338
339                                 }
340                                 if(packet[1] == 1) {
341
342                                     servo_turn(1,
                                        standard_scan_duty_sensor2a[
                                        packet[2]],
                                        standard_scan_duty_sensor2b[
                                        packet[2]]);
343
344                                 }
345                                 packet[0] = 0x06;
346                                 packet[1] = 0x0f;
347                                 CE_LOW();
348                                 radio_config_tx();
349                                 do {
350                                     again = 0;
351                                     radio_transmit(&packet[0]);
352                                     while(again != 1);
353
354                                 } while(transmitted != 1);
355                                 radio_config_rx();
356                                 CE_HIGH();
357                                 state = 255;
```

```
358                                 break;
359
360                         }
361                 }
362 }
363
364 /* Avbrottshanterare for radiokommunikationen */
365 ISR(INT2_vect) {
366         radio_config_rx(); // Staller in sa att det gar att skicka
                meddelande till Arne aven om han ar upptagen
367         CE_HIGH();
368         status = radio_status(); // Laser av radiomodulens statusregister
369
370         if((status & RX_DR) != 0) { // Kontroll om paket mottaget
371                 transmitted = 0;
372                 radio_rw_combo(W_REGISTER, STATUS, (status|RX_DR)); //
                        Nollstall flaggan
373                 radio_recieve(&recievedpacket[0]);
374                 if(state == 255) { // Om arne ej upptagen andra hans state
                        variabel
375
376                         state = recievedpacket[0];
377                         uint8_t i;
378                         for(i = 0; i < 32; i++) {
379                                 packet[i] = recievedpacket[i];
380                         }
381                 }
382                 if(recievedpacket[0] == SOFT_RESET) { // Kor SOFT_RESET aven
                        om Arne upptagen
383
384                         state = recievedpacket[0];
385                         uint8_t i;
386                         for(i = 0; i < 32; i++) {
387                                 packet[i] = recievedpacket[i];
388                         }
389                 }
390         }
391         if((status & TX_DS) != 0) { // Kontroll av om paket kommit fram, om
                det kommit fram kan Arne hoppa ur sandningsloopen
392                 radio_rw_combo(W_REGISTER, STATUS, status|TX_DS);
393                 transmitted = 1;
394                 again = 1;
395         }
396         if((status & MAX_RT) != 0) { // Kontroll om radiomodulen kommit upp
                i maximalt antal omsandningar, isafall fortsatt sand
397                 radio_rw_combo(W_REGISTER, STATUS, status|TX_DS);
398                 transmitted = 0;
399                 again = 1;
400                 CE_LOW();
401                 radio_config_tx();
402         }
403         if((status & TX_FULL) != 0) { // Kontroll om mottagarresigterna ar
                fulla, isafall nollstall dem
404
405                 radio_rw(FLUSH_TX, NOP);
406                 CE_LOW();
407                 radio_config_tx();
408                 transmitted = 1;
409                 again = 1;
410         }
411 }
```

B.1.2 Motorstyrning

motor.h

```
#ifndef MOTOR_H
#define MOTOR_H
#include <avr/io.h>
#include <avr/interrupt.h>

#define F_LEFT 0b00000001
#define B_LEFT 0b00000010
#define S_LEFT 0b11111100
#define F_RIGHT 0b10000000
#define B_RIGHT 0b01000000
#define S_RIGHT 0b00111111


void motor_setup(void);
void motor_stop(void);
void motor_forward(void);
void motor_backward(void);
void motor_turn_left(void);
void motor_turn_right(void);
void motor_drive_soft(uint8_t type, uint8_t duty_left, uint8_t duty_right);
void motor_forward_left(void);
void motor_forward_right(void);
void motor_stop_left(void);
void motor_stop_right(void);
void motor_backward_left(void);
void motor_backward_right(void);
void motor_forward_soft(uint8_t duty_left, uint8_t duty_right);
void motor_change_direction_left(void);
void motor_change_direction_right(void);
void motor_step(uint8_t stepright, uint8_t stepleft, uint8_t move);
uint8_t motor_duty_right(uint8_t duty);
uint8_t motor_duty_left(uint8_t duty);
void motor_turn_right_soft(uint8_t duty_left, uint8_t duty_right);

#endif
```

motor.c

```
#define F_CPU 16000000
#include <util/delay.h>
#include "motor.h"


volatile uint8_t count_left, count_right, stepleft, stepright, stepactivated
    ;

void motor_setup(void) {

        DDRB |= (1<<PB3); // Satter OC0 som utgang
        DDRC |= (1<<PC0)|(1<<PC1)|(1<<PC6)|(1<<PC7); // Satter
            motorstyrningsutgangarna

```

```
13          TCCR0 |= (1<<WGM00)|(1<<COM01)|(1<<CS01); // PWM installningar for
                hogra motorn, Phase Correct
14          OCR0 = 0x00; // Duty hogra motorn
15
16          DDRD |= (1<<PD7); // Satter OC2 som utgang
17          TCCR2 |= (1<<WGM20)|(1<<COM21)|(1<<CS21); // PWM installningar for
                den vanstra motorn, Phase Correct
18          OCR2 = 0x00; // Duty vanstra motorn
19
20  }
21
22  // Installning av vanstra motorns duty cycle
23  uint8_t motor_duty_left(uint8_t duty) {
24
25          OCR2 = duty;
26          return duty;
27
28  }
29
30  // Installning av hogra motorns duty cycle
31  uint8_t motor_duty_right(uint8_t duty) {
32
33          OCR0 = duty;
34          return duty;
35
36  }
37
38  // Satter styrsignalerna till H-bryggan sadant att motorerna stannar
39  void motor_stop(void) {
40
41          PORTC = 0b00000000;
42
43  }
44
45  // Satter utgangarna sadant att motorerna gar framatriktning
46  void motor_forward(void) {
47
48          PORTC = 0b10000001;
49
50  }
51
52  /* Mjukstart av motorn, borjar pa halva vardet av de valda duty cyclarna och
        okar sedan linjart upp till
53  de valda vardena */
54  void motor_forward_soft(uint8_t duty_left, uint8_t duty_right) {
55
56          OCR0 = 0;
57          OCR2 = 0;
58          PORTC = 0b10000001;
59          uint8_t i;
60          uint8_t test1, test2;
61          test1 = duty_left/64;
62          test2 = duty_right/64;
63          OCR0 = (duty_right/2);
64          OCR2 = (duty_left/2);
65          for(i = 0; i < 32; i++) {
66
67                  OCR0 = OCR0 + test1;
68                  OCR2 = OCR2 + test2;
69                  _delay_ms(12);
70
71          }
```

```
72          OCR0 = duty_right;
73          OCR2 = duty_left;
74  }
75
76  // Satter utgangarna sadant att motorerna gar bakattriktning
77  void motor_backward(void) {
78
79          PORTC = 0b01000010;
80
81  }
82
83
84  // Satter utgangarna sadant att vanstra motorn gar i framatriktning och
        hogra i bakatriktning
85  void motor_turn_left(void) {
86
87
88          PORTC = 0b10000010;
89
90  }
91
92
93  // Satter utgangarna sadant att hogra motorn gar i framatriktning och
        vanstra i bakatriktning
94  void motor_turn_right(void) {
95
96
97          PORTC = 0b01000001;
98
99  }
100
101 void motor_drive_soft(uint8_t type, uint8_t duty_left, uint8_t duty_right) {
102
103         switch(type) {
104
105                 case 0:
106                 motor_forward();
107                 break;
108
109                 case 1:
110                 motor_backward();
111                 break;
112
113                 case 2:
114                 motor_turn_left();
115                 break;
116
117                 case 3:
118                 motor_turn_right();
119                 break;
120         }
121         OCR0 = 0;
122         OCR2 = 0;
123         uint8_t i;
124         uint8_t test1, test2;
125         test1 = duty_left/64;
126         test2 = duty_right/64;
127         OCR0 = (duty_right/2);
128         OCR2 = (duty_left/2);
129         for(i = 0; i < 32; i++) {
130
131                 OCR0 = OCR0 + test1;
```

```
132                     OCR2 = OCR2 + test2;
133                     _delay_ms(12);
134
135             }
136             OCR0 = duty_right;
137             OCR2 = duty_left;
138
139     }
140
141     // Satter utgangarna sadant att vanstra motorn gar i framatriktning
142     void motor_forward_left(void) {
143
144             PORTC = 0b00000001;
145
146     }
147
148     // Bytar riktning for den vanstra motorn
149     void motor_change_direction_left(void) {
150
151             if((PORTC & 0b00000011) == 1) {
152                     PORTC &= 0b11111100;
153                     PORTC |= 0b00000010;
154                     return;
155             }else if((PORTC & 0b00000011) == 2) {
156                     PORTC &= 0b11111100;
157                     PORTC |= 0b00000001;
158                     return;
159             }else{
160                     return;
161             }
162     }
163
164
165     // Satter utgangarna sadant att hogra motorn gar i framatriktning
166     void motor_forward_right(void) {
167
168             PORTC = 0b10000000;
169
170     }
171
172     // Bytar riktning for den hogra motorn
173     void motor_change_direction_right(void) {
174
175             if(((PORTC & 0b11000000)>>6) == 1) {
176                     PORTC &= 0b00111111;
177                     PORTC |= 0b10000000;
178                     return;
179             }else if(((PORTC & 0b11000000)>>6) == 2) {
180                     PORTC &= 0b00111111;
181                     PORTC |= 0b01000000;
182                     return;
183             }else{
184                     return;
185             }
186     }
187
188     // Stannar den vanstra motorn
189     void motor_stop_left(void) {
190
191             PORTC &= S_LEFT;
192
193     }
```

```
194
195 // Stannar den hogra motorn
196 void motor_stop_right(void) {
197
198         PORTC &= S_RIGHT;
199
200 }
201
202 // Satter den vanstra motorn i bakatriktning
203 void motor_backward_left(void) {
204
205         PORTC |= 0b00000010;
206         PORTC &= 0b11111110;
207
208 }
209
210 // Satter den hogra motorn i bakatriktning
211 void motor_backward_right(void) {
212
213         PORTC |= 0b01000000;
214         PORTC &= 0b01111111;
215
216
217 }
218
219 /* Funktion for styrning av hur manga steg motorn ska kora dar ett steg
       motsvarar att avbrottshanteraren registerar en positiv/negativ flank
       fran utsignalen av lasgaffeln. Fyra olika rorelsemonster valjs, framat,
       bakat, rotera vanster och rotera hoger. Antalet hoger och vanstersteg ar
220 alltid detsamma och vid varje steg raknas variablerna ner. Om den ena hjulet
       har snurrat for langt i forhallande till det andra stangs den motorn av
       sa att den andra motorn kan kora ifatt skillnaden. Gransen for hur stor
       skillnaden ska vara innan den stanger av en motorn ar valbar for alla
       rorelsemonster */
221
222 void motor_step(uint8_t stepsright, uint8_t stepsleft, uint8_t move) {
223
224         stepactivated = 1; // Variabel for aktivering av nedrakning
225         stepright = stepsright;
226         stepleft = stepsleft;
227         uint8_t duty_left_orig = motor_duty_left(OCR2);
228         uint8_t duty_right_orig = motor_duty_right(OCR0);
229         count_left = 0; // Variabel for avbrottsavstangning efter att
                vanstra hjulet ar fardigt
230         count_right = 0;// Variabel for avbrottsavstangning efter att hogra
                hjulet ar fardigt
231         uint8_t running = 0; // Variabel for antal motorer som ar fardiga
232         uint8_t limit = 0; // Grans for hur stor stegskillnaden far vara
                hjulen emellan
233         uint8_t stepleftorig = stepleft; // Sparar originalvardet for antal
                steg for dubbelkoll ifall antal steg kvar gatt over fran 0 till
                255
234         uint8_t steprightorig = stepright;
235         void (*functionptr)(void);
236         /* Kontroll av vilket rorelsemonster som valts*/
237         if(move == 0) {
238                 motor_drive_soft(0,OCR2,OCR0);
239                 functionptr = &motor_forward;
240                 limit = 3;
241         }else if(move == 1) {
242                 motor_drive_soft(1,OCR2,OCR0);
243                 functionptr = &motor_backward;
```

```
244             limit = 4;
245         }else if(move == 2) {
246             motor_drive_soft(2,OCR2,OCR0);
247             functionptr = &motor_turn_left;
248             limit = 5;
249         }else if(move == 3) {
250             motor_drive_soft(3,OCR2,OCR0);
251             functionptr = &motor_turn_right;
252             limit = 5;
253         }else{
254             return;
255         }
256         functionptr();
257         /* Sa lange bada hjulen inte snurrat tillrackligt fortsatter loopen
                att ga */
258         while(running < 2) {
259
260             if(((stepright == 0) | (stepright > steprightorig)) & (
                    count_right == 0)) {
261
262                 count_right = 1;
263                 running++;
264                 motor_change_direction_right();
265                 _delay_ms(40);
266                 motor_stop_right();
267
268             }
269             if(((stepleft == 0) | (stepleft > stepleftorig)) & (
                    count_left == 0)) {
270
271                 count_left = 1;
272                 running++;
273                 motor_change_direction_left();
274                 _delay_ms(40);
275                 motor_stop_left();
276
277             }
278             if(((stepleft > 1 )| (stepright > 1)) & (count_left == 0) &
                    (count_right == 0)) {
279
280                 if((stepleft - stepright >= limit)) {
281
282                     motor_stop_right();
283                     _delay_ms(10);
284
285                 if(((stepright == 0) | (stepright >
                        steprightorig))) {
286
287                     count_right = 1;
288                     running++;
289                     motor_change_direction_right();
290                     _delay_ms(20);
291                     motor_stop_right();
292
293                 }else if(((stepleft == 0) | (stepleft >
                        stepleftorig))) {
294
295                     count_left = 1;
296                     running++;
297                     motor_change_direction_left();
298                     _delay_ms(20);
299                     motor_stop_left();
```

```
300
301                                     } else {
302                                             functionptr();
303                                     }
304
305                             }else if((stepright - stepleft >= limit)) {
306
307                                             motor_stop_left();
308                                             _delay_ms(10);
309
310                                     if(((stepleft == 0) | (stepleft >
                                            stepleftorig))) {
311                                             count_left = 1;
312                                             running++;
313                                             motor_change_direction_left();
314                                             _delay_ms(40);
315                                             motor_stop_left();
316                                     } else  if(((stepright == 0) | (stepright >
                                            steprightorig))) {
317
318                                             count_right = 1;
319                                             running++;
320                                             motor_change_direction_right();
321                                             _delay_ms(40);
322                                             motor_stop_right();
323
324                                     } else {
325                                             functionptr();
326                                     }
327                             }
328                     }
329             }
330             motor_duty_left(duty_left_orig);
331             motor_duty_right(duty_right_orig);
332             stepactivated = 0;
333
334     }
335
336     /* Avbrottshanterare for hogra hjulets lasgaffeln */
337     ISR(INT0_vect) {
338
339             if((count_right == 0) & (stepactivated == 1)) {
340                     stepright--;
341                     return;
342             }
343     }
344
345     /*Avbrottshanterare for vanstra hjulets lasgaffel */
346     ISR(INT1_vect) {
347
348             if((count_left == 0) & (stepactivated == 1)) {
349                     stepleft--;
350                     return;
351             }
352     }
```

movement.h

```
#ifndef MOVEMENT_H
#define MOVEMENT_H

#include <avr/io.h>

void movement_setup(void);

#endif
```

movement.c

```
#include <avr/io.h>

/* Staller in portarna och registren i AVRen for att gora avbrottshantering
    mojligt. Anvands vid motorstyrningen da
lasgafflarnas utsignaler genererar avbrott */
void movement_setup(void) {
        GICR |= ((1<<INT0) | (1<<INT1)); // Aktiverar avbrott for pinne INT0
             och INT1
        MCUCR |= ((1<<ISC00) | (1<<ISC10)); // Triggar pa logisk andring
}
```

B.1.3 Servostyrning

servo.h

```
#ifndef SERVO_H
#define SERVO_H

#include <avr/io.h>

void servo_setup(void);
void servo_turn(uint8_t servo, uint8_t x1, uint8_t x2);
void servo_turnback(void);
void servo_flip1(void);
void servo_flip2(void);

#endif
```

servo.c

```
#define F_CPU 16000000UL
#include <util/delay.h>
#include "servo.h"

/*
        f       16e6 n/s
        tt      62.5e-9 s/n
        presc   8
        n       t/tt/presc
```

```
10          20ms     40k
11          1ms      2k
12          1.5ms    3k
13          2ms      4k
14  */
15
16  void servo_setup(void) {
17
18          DDRD |= (1<<4)|(1<<5); // Satter pinne OC1A och OC1B till utgang
19
20          /* Fast PWM med ICR1 som storsta varde samt prescaler pa 8*/
21          TCCR1A |= (1<<COM1A1)|(1<<COM1B1)|(1<<WGM11);
22          TCCR1B |= (1<<WGM13)|(1<<WGM12)|(1<<CS11);
23
24          /* Storsta varde satts till 40000 vilket ger 50 Hz alternativt 20 ms
                */
25          ICR1 = 40000;
26          OCR1B = 2650;
27          OCR1A = 2525;
28  }
29
30  /* andrar duty cycle for valbart servo */
31  void servo_turn(uint8_t servo, uint8_t x1, uint8_t x2) {
32
33          if(servo == 1) {
34                  OCR1B = 255*x1 + x2;
35          }
36
37          if(servo == 2) {
38                  OCR1A = 255*x1 + x2;
39          }
40  }
41
42  /* Satter bada servona i deras grundlage */
43  void servo_turnback(void) {
44
45          OCR1B = 1232;
46          OCR1A = 1136;
47  }
48
49
50  /* Testsekvens for anvandning vid felsokning */
51  void servo_flip1(void) {
52
53          OCR1B = 1400;
54          _delay_ms(40);
55          OCR1B = 1232;
56          _delay_ms(40);
57
58  }
59
60  /* Testsekvens for anvandning vid felsokning */
61  void servo_flip2(void) {
62
63          OCR1A = 1400;
64          _delay_ms(200);
65          OCR1A = 1136;
66          _delay_ms(200);
67
68  }
```

B.1.4 Avståndsmätning

distance.h

```
#ifndef DISTANCE_H
#define DISTANCE_H

#include <avr/io.h>

void distance_setup(void);
void distance_powerdown(void);
uint8_t distance_readsensor(uint8_t sensor);

#endif
```

distance.c

```
#include "distance.h"

/* Staller in alla portar och register for AD-omvandling av
    avstandssensorsernas utsignaler */
void distance_setup(void)
{
        DDRA &=~(1<<PA1);
        ADCSRA |= (1<<ADEN); // Aktiverar AD-omvandling
        ADMUX &= ~((1<<REFS1) | (1<<REFS0) | (1<<MUX4) | (1<<MUX3) | (1<<
            MUX2) | (1<<MUX1)); // Satter AREF som referens och ADC1 som
             forsta omvandlings kanal
        ADMUX |= (1<<MUX0);
        ADCSRA |= ((1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0)); // Prescaler satts
             som 128 => 16 MHz/128 = 125 kHz

}

/* For inaktivering av AD-omvandling */
void distance_powerdown(void)
{

        ADCSRA &= ~(1<<ADEN);

}

/*startar AD-omvandlingen */
uint8_t distance_readsensor(uint8_t sensor)
{

        if ((ADCSRA&(1<<ADSC)) == 0) { // Kontrollerar sa att ingen
            omvandling redan pagar

                if (sensor == 1) {

                        //ADMUX |= (1<<MUX0); // andrar till kanalen for
                            avstandssensor1
                        ADCSRA |= (1<<ADSC); // Paborjar AD-omvandling
                        return 1;

                }
```

```
35          }
36          return 0;
37  }
```

B.2 Tarne

B.2.1 Huvudprogrammet

source.c

```
1   #include <avr/io.h>
2   #include <avr/interrupt.h>
3   #define F_CPU 16000000UL
4   #include <util/delay.h>
5
6   #include "../usart/usart.h"
7   #include "../radio/radio.h"
8   #include "../led/led.h"
9
10  #define PACKET_SIZE 32
11
12  #define USART_LISTEN 0
13  #define USART_SEND 1
14  #define RADIO_RECIEVE 2
15  #define RADIO_RECIEVE2 3
16  #define RADIO_TRANSMIT 4
17
18  volatile uint8_t state = 0;
19  volatile uint8_t i;
20  volatile uint8_t packet[PACKET_SIZE];
21  volatile uint8_t recievedpacket[PACKET_SIZE];
22  volatile uint8_t transmitted = 0;
23  volatile uint8_t recieved = 0;
24  volatile uint8_t wait = 0;
25  volatile uint8_t status;
26
27  int main (void) {
28
29
30          DDRA |= 0x07;
31          _delay_ms(50);
32          usart_setup();
33          radio_setup(PACKET_SIZE, 33);
34          radio_config_rx();
35          CE_HIGH();
36          led_setup();
37          status = 0;
38
39          cool_sequence(); // Flash the leds to tell that the device has
                booted and is ready
40
41          sei();
42
43          for (;;) {
```

```
44                  switch (state) {
45                          case USART_LISTEN: // waiting for data from the
                                computer
46                                  LED = RED | YELLOW;
47                                  break;
48
49                          case USART_SEND: // transmitting to the computer
50                                  LED = RED;
51                                  break;
52
53                          case RADIO_RECIEVE: // waiting for data from arne
54                                  LED = GREEN | YELLOW;
55                                  radio_config_rx();
56                                  CE_HIGH();
57                                  state = RADIO_RECIEVE2;
58
59                          case RADIO_RECIEVE2:
60                                  if(recieved) {
61                                          UDR = packet[0];
62                                          state = USART_SEND;
63                                          recieved = 0;
64                                          i = 0;
65                                  }
66                                  break;
67
68                          case RADIO_TRANSMIT: // transmitting data to arne
69                                  LED = GREEN;
70                                  CE_LOW();
71                                  radio_config_tx();
72                                  do {
73                                          wait = 0;
74                                          radio_transmit(packet);
75                                          while(wait != 1);
76
77                                  } while(transmitted != 1);
78                                  state = RADIO_RECIEVE;
79                                  transmitted = 0;
80                                  radio_config_rx();
81                                  CE_HIGH();
82                                  break;
83                  }
84          }
85  }
86
87  ISR(USART_RXC_vect) { // usart recieve complete interrupt, load the recieved
        byte into the array
88
89          if (state == USART_LISTEN) {
90                  LED = 0x00;
91                  packet[i++] = UDR;
92                  if (i == PACKET_SIZE) {
93                          state = RADIO_TRANSMIT;
94                  }
95          }
96
97  }
98
99  ISR(USART_TXC_vect) { // usart transmit complete interrupt, send next byte
100
101         if (state == USART_SEND) {
102                 LED = RED | GREEN | YELLOW;
103                 if (i == PACKET_SIZE-1) {
```

```
104                         i = 0;
105                         state = USART_LISTEN;
106                 } else {
107                         UDR = packet[++i];
108                 }
109         }
110 }
111 /* Avbrottshanterare for radiokommunikationen */
112 ISR(INT2_vect) {
113
114         radio_config_rx();
115         CE_HIGH();
116         status = radio_status(); // Laser av radiomodulens statusregister
117
118         if((status & RX_DR) != 0) { // Kontroll om paket mottaget
119                 radio_rw_combo(W_REGISTER, STATUS, (status|RX_DR)); //
                        Nollstall flaggan
120                 radio_recieve(&recievedpacket[0]);
121                 if((state == RADIO_RECIEVE) | (state == RADIO_RECIEVE2)) {
122                         recieved = 1;
123                         uint8_t i;
124                         for(i = 0; i < 32; i++) {
125                                 packet[i] = recievedpacket[i];
126                         }
127                 }
128         }
129         if((status & TX_DS) != 0) { // Kontroll av om paket kommit fram
130                 radio_rw_combo(W_REGISTER, STATUS, status|TX_DS);
131                 transmitted = 1;
132                 wait = 1;
133         }
134         if((status & MAX_RT) != 0) { // Kontroll om radiomodulen kommit upp
                i maximalt antal omsandningar, isafall fortsatt sand
135                 radio_rw_combo(W_REGISTER, STATUS, status|TX_DS);
136                 transmitted = 0;
137                 wait = 1;
138                 CE_LOW();
139                 radio_config_tx();
140         }
141         if((status & TX_FULL) != 0) { // Kontroll om mottagarresigterna ar
                fulla, isafall nollstall dem
142
143                 radio_rw(FLUSH_TX, NOP);
144                 CE_LOW();
145                 radio_config_tx();
146                 transmitted = 1;
147                 wait = 1;
148         }
149 }
```

B.2.2 Seriell kommunikation

usart.h

```
1 #ifndef USART_H
2 #define USART_H
3
```

```
4  #include <avr/io.h>
5
6  uint8_t usart_buffer[32];
7
8  void usart_setup(void);
9  /*
10
11 Only used during testing
12
13 void usart_sends(uint8_t *s);
14 void usart_sendhex(uint8_t c);
15 void usart_sendn(uint8_t *data, uint8_t n);
16 */
17 #endif
```

usart.c

```
1  #include "usart.h"
2
3  void usart_setup(void)
4  {
5          // transmit/recieve enable and corresponding interrupts
6          UCSRB |= (1<<TXEN)|(1<<RXEN)|(1<<RXCIE)|(1<<TXCIE);
7
8          // no parity, one stop bit
9          UCSRC |= (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
10
11         // 9600bd @ 16MHz
12         UBRRH = 0x00;
13         UBRRL = 0x67;
14 }
15
16 /*
17
18 Only used during testing
19
20 void usart_sends(uint8_t *s)
21 {
22         while (*s) {
23                 while (!(UCSRA & (1<<UDRE)));
24                 UDR = *s;
25                 s += 1;
26         }
27 }
28
29 void usart_sendhex(uint8_t c)
30 {
31         uint8_t tmp[5] = "0xA5\0";
32         tmp[2] = (c>>4) + 0x30;
33         if (tmp[2] > 0x39)
34                 tmp[2] += 0x07;
35         tmp[3] = (c & 0x0f) + 0x30;
36         if (tmp[3] > 0x39)
37                 tmp[3] += 0x07;
38         usart_sends(&tmp[0]);
39 }
40
41 void usart_sendn(uint8_t *data, uint8_t n)
```

```
42 {
43         uint8_t i;
44         for (i=0; i<n; i++) {
45                 while (!(UCSRA & (1<<UDRE)));
46                 UDR = data[i];
47         }
48 }
49 */
```

B.3 Gemensam kod

B.3.1 Radiokommunikationssystemet

radio.h

```
1  #ifndef RADIO_H
2  #define RADIO_H
3
4  #include <avr/io.h>
5
6  #define RADIO_DDR DDRB
7  #define RADIO_PORT PORTB
8  #define RADIO_PIN PINB
9
10 #define RADIO_CE PB1
11 #define RADIO_IRQ PB2
12 #define RADIO_CSN PB4
13
14 // command definitions
15 #define R_REGISTER              0b00000000
16 #define W_REGISTER              0b00100000
17 #define R_RX_PAYLOAD            0b01100001
18 #define W_TX_PAYLOAD            0b10100000
19 #define FLUSH_TX                0b11100001
20 #define FLUSH_RX                0b11100010
21 #define REUSE_TX_PL             0b11100011
22 #define ACTIVATE                0b01010000
23 #define R_RX_PL_WID             0b01100000
24 #define W_ACK_PAYLOAD           0b10101000
25 #define W_TX_PAYLOAD_NO_ACK     0b10110000
26 #define NOP                     0b11111111
27
28 // register definitions
29 #define CONFIG                  0x00
30 #define EN_AA                   0x01
31 #define EN_RXADDR               0x02
32 #define SETUP_AW                0x03
33 #define SETUP_RETR              0x04
34 #define RF_CH                   0x05
35 #define RF_SETUP                0x06
36 #define STATUS                  0x07
37 #define OBSERVE_TX              0x08
38 #define CD                      0x09
39 #define RX_ADDR_P0              0x0a
```

```
40 #define RX_ADDR_P1              0x0b
41 #define RX_ADDR_P2              0x0c
42 #define RX_ADDR_P3              0x0d
43 #define RX_ADDR_P4              0x0e
44 #define RX_ADDR_P5              0x0f
45 #define TX_ADDR                 0x10
46 #define RX_PW_P0                0x11
47 #define RX_PW_P1                0x12
48 #define RX_PW_P2                0x13
49 #define RX_PW_P3                0x14
50 #define RX_PW_P4                0x15
51 #define RX_PW_P5                0x16
52 #define FIFO_STATUS             0x17
53 #define DYNPD                   0x1c
54 #define FEATURE                 0x1d
55
56 // status register bitmasks
57 #define RX_DR                   (1<<6)
58 #define TX_DS                   (1<<5)
59 #define MAX_RT                  (1<<4)
60 #define RX_P_NO                 (1<<3)|(1<<2)|(1<<1)
61 #define TX_FULL                 (1<<0)
62
63 // config register bitmasks
64 #define MASK_RX_DR              (1<<6)
65 #define MASK_TX_DS              (1<<5)
66 #define MASK_MAX_RT             (1<<4)
67 #define EN_CRC                  (1<<3)
68 #define CRCO                    (1<<2)
69 #define PWR_UP                  (1<<1)
70 #define PRIM_RX                 (1<<0)
71
72 #define CSN_LOW() RADIO_PORT &=~ (1<<RADIO_CSN)
73 #define CSN_HIGH() RADIO_PORT |= (1<<RADIO_CSN)
74
75 #define CE_LOW() RADIO_PORT &=~ (1<<RADIO_CE)
76 #define CE_HIGH() RADIO_PORT |= (1<<RADIO_CE)
77
78 #define RADIO_IRQ_BIT (RADIO_PIN & (1<<RADIO_IRQ))
79
80 static uint8_t radio_packet_length = 1;
81
82 void spi_setup(void);
83 uint8_t spio(uint8_t b);
84 uint8_t radio_rw(uint8_t cmd, uint8_t data);
85 uint8_t radio_rw_combo(uint8_t cmd, uint8_t extra, uint8_t data);
86 void radio_setup(uint8_t plength, uint8_t channel);
87 void radio_config_tx(void);
88 void radio_config_rx(void);
89 uint8_t radio_status(void);
90 uint8_t radio_transmit(uint8_t *packet);
91 uint8_t radio_recieve(uint8_t *packet);
92 //void radio_regdump(void);
93
94 #endif
```

radio.c

```
1  #include "radio.h"
2
3  #define F_CPU 16000000
4  #include <util/delay.h>
5  //#include "../usart/usart.h"
6  /*
7  This is more or less an adaption of the example code that are provided for
       the module on Lawicel's homepage
8  */
9  uint8_t radio_buffer[32];
10
11 void spi_setup(void)
12 {
13         // setup SPI pins' io config
14         DDRB |= (1<<PB4)|(1<<PB5)|(1<<PB7);
15         DDRB &=~ (1<<PB6);
16         PORTB |= (1<<PB4);
17
18         // SPI enable, master mode, fcpu/4 (=4MHZ @ 16MHZ fcpu)
19         // DORD = 0 (MSb first)
20         // CPHA = 0
21         // CPOL = 0
22         SPCR |= (1<<SPE)|(1<<MSTR);
23
24         MCUCSR &= ~(1<<ISC2); // Trigg on negative slope
25         GICR |= (1<<INT2); // Activates interrupt vector 2
26 }
27
28 uint8_t spio(uint8_t b)
29 {
30         // Write a byte and read a byte.
31         SPDR = b;
32         while (!(SPSR & (1<<SPIF)));
33         return SPDR;
34 }
35
36 uint8_t radio_rw(uint8_t cmd, uint8_t data)
37 {
38         uint8_t status, tmp;
39         CSN_LOW();
40         status = spio(cmd);
41         if (cmd == R_RX_PAYLOAD) {
42                 tmp = spio(NOP);
43                 CSN_HIGH();
44                 return tmp;
45         }
46         if (cmd == W_TX_PAYLOAD) {
47                 spio(data);
48                 CSN_HIGH();
49                 return status;
50         }
51         CSN_HIGH();
52         return status;
53 }
54
55 uint8_t radio_rw_combo(uint8_t cmd, uint8_t reg, uint8_t data)
56 {
57         uint8_t compound,ret, i;
58         CSN_LOW();
59         compound = cmd | reg;
60         ret = spio(compound);
61         if (cmd == R_REGISTER) {
```

```
62                  if (reg == RX_ADDR_P0 || reg == RX_ADDR_P1 || reg == TX_ADDR
                        ) {
63                          for (i=0; i!=5; i++) {
64                                  radio_buffer[i] = spio(NOP);
65                          }
66                  } else {
67                          ret = spio(NOP);
68                  }
69          } else if (cmd == W_REGISTER) {
70                  if (reg == RX_ADDR_P0 || reg == RX_ADDR_P1 || reg == TX_ADDR
                        ) {
71                          for (i=0; i!=5; i++) {
72                                  spio(radio_buffer[i]);
73                          }
74                  } else {
75                          spio(data);
76                 }
77          }
78          CSN_HIGH();
79          return ret;
80  }
81
82  void radio_setup(uint8_t plength, uint8_t channel)
83  {
84          spi_setup();
85          // setup radio control signals' io config
86          RADIO_DDR |= (1<<RADIO_CE)|(1<<RADIO_CSN); // outputs
87          RADIO_DDR &=~ (1<<RADIO_IRQ);           // inputs
88          CE_LOW();
89          radio_rw_combo(W_REGISTER, RX_PW_P0, plength); // P0 packet length
                of 1 byte
90          radio_rw_combo(W_REGISTER, RX_PW_P1, plength); // P1 ...
91          radio_packet_length = plength;
92          radio_rw_combo(W_REGISTER, RF_CH, channel);
93          uint8_t status = radio_status();
94          radio_rw(FLUSH_RX, NOP);
95          radio_rw_combo(W_REGISTER, STATUS, (status|TX_DS));
96          radio_rw_combo(W_REGISTER, STATUS, (status|RX_DR));
97          radio_rw_combo(W_REGISTER, STATUS, (status|MAX_RT));
98  }
99
100 // Transmitting mode
101
102 void radio_config_tx(void)
103 {
104         radio_rw_combo(W_REGISTER, CONFIG, EN_CRC | CRCO | PWR_UP);
105 }
106
107 // Recieving mode
108
109 void radio_config_rx(void)
110 {
111         radio_rw_combo(W_REGISTER, CONFIG, EN_CRC | CRCO | PWR_UP | PRIM_RX)
                ;
112 }
113
114 // Return radio module status register
115
116 uint8_t radio_status(void)
117 {
118         return radio_rw(NOP, NOP);
119 }
```

```
120
121 // Transmit a packet
122
123 uint8_t radio_transmit(uint8_t *packet)
124 {
125         uint8_t status, i;
126         CSN_LOW();
127         status = spio(W_TX_PAYLOAD);
128         for (i=0; i<radio_packet_length; i++) {
129                 spio(packet[i]);
130         }
131         CSN_HIGH();
132         CE_HIGH();
133         _delay_us(128);
134         CE_LOW();
135         return status;
136 }
137
138 // Recieve a packet
139
140 uint8_t radio_recieve(uint8_t *packet)
141 {
142         uint8_t status, i;
143         CSN_LOW();
144         status = spio(R_RX_PAYLOAD);
145         for (i=0; i<radio_packet_length; i++) {
146                 packet[i] = spio(NOP);
147         }
148         CSN_HIGH();
149         return status;
150 }
151
152 // Returns all the radio module registers (for debug)
153
154 /*void radio_regdump(void)
155 {
156         uint8_t reg, value, i;
157         for (reg=0; reg < 0x1e; reg++) {
158                 usart_sendhex(reg);
159                 usart_sends(":  ");
160                 if (reg <= FIFO_STATUS || reg >= DYNPD) {
161                         value = radio_rw_combo(R_REGISTER, reg, NOP);
162                         if (reg == RX_ADDR_P0 || reg == RX_ADDR_P1 || reg ==
                                TX_ADDR) {
163                                 for (i=0; i!=5; i++) {
164                                         usart_sendhex(radio_buffer[i]);
165                                         usart_sends(" ");
166                                 }
167                         } else {
168                                 usart_sendhex(value);
169                         }
170                 }
171                 usart_sends("\n");
172         }
173 }*/
```

B.4 MATLAB

B.4.1 Styrsystem

main.m

```
% Arnes startvariabler %
%-----------------------%
xpos = 0;
ypos = 0;
angle = 0;
%-----------------------%

db0 = [];
s = serial_setup();
decision = 0; % Variabel for val av satt att undvika hinder
olddecision = 0; % Variabel for foregaende satt att undvika hinder
flushbuffer(s); % Nollstall SERIE buffer

while 1
    decision = 42;
    while decision ~= 0
        scandata = standardscan(s); % Startar en vanlig scan med
            avstandssensorn.
        db0 = update(db0, xpos, ypos, angle, 0, scandata); % Uppdaterar
            matdata vektorn
        latestentry = db0(length(db0)); % Sparar senaste matdata i en
            variabel
        pause(.1);
        plot_lvl0(latestentry); % Lagger till senaste matdatan i realplotten
        decision = distancecheck(latestentry, olddecision); % Anvander
            matdatan for kontroll av om Arne maste vaja far nagot foremal
        olddecision = decision; % Sparar undan vilket val funktionen gjorde
        if decision ~= 0 % Om decision inte ar 0 innebar det att Arne maste
            vaja
            if decision == -1
                turnleft45(s);
                angle = angle+45; % Arne svanger men han flyttar sig inte i
                    x- eller y-led alltsa behovs bara hans vinkel uppdateras
            else
                turnright45(s);
                angle = angle-45;
            end
        end
    end
    forward(s); % Arne kan nu kora framat igen och forst nu kommer han ha
        flyttat sig i x- och y-led
    xpos = xpos + 25*cosd(angle);
    ypos = ypos + 25*sind(angle);
end
```

standardscan.m

```
function values = standardscan(s)
%
```

```
3  % Does a standard scan and converts the ADC returned values into centimeters
       with the precalculated polynomial p
4  %
5  p = [-1.675357466357969e-026 9.509940672110916e-023 -2.399725686579022e-019
       3.546630585138074e-016 -3.403970098348621e-013 2.222570182313870e-010
       -1.004916699733606e-007 3.138800893594859e-005 -0.006625430364713
       0.900702727998019 -71.681680548677534 2.687064513824892e+003];
6
7  packet = zeros(1, 32);
8  packet(1) = 6;
9
10 fwrite(s, packet);
11 while s.BytesAvailable < 32, end
12 packet1 = fread(s, 32);
13 packet(1) = 7;
14
15 fwrite(s, packet);
16 while s.BytesAvailable < 32, end
17 packet2 = fread(s, 32);
18
19 values = zeros(21, 2);
20 values(:,1) = 20:7:160;
21 i = 1;
22 c = 3;
23 while(i <= 21)
24
25     values(i,2) = packet1(c)*255+packet1(c+1);
26     if values(i,2) > 850
27         values(i,2) = 10;
28     elseif values(i,2) < 110
29         values(i,2) = 210;
30     else
31         values(i,2) = polyval(p, values(i,2));
32     end
33     i = i + 2;
34     c = c + 2;
35
36 end
37
38 i = 2;
39 c = 3;
40
41 while(i <= 20)
42
43      values(i,2) = packet2(c+1)*255+packet2(c);
44     if values(i,2) > 850
45         values(i,2) = 10;
46     elseif values(i,2) < 110
47         values(i,2) = 210;
48     else
49         values(i,2) = polyval(p, values(i,2));
50     end
51     i = i + 2;
52     c = c + 2;
53
54 end
55 end
```

update.m

```
function updateddb = update(db, xpos, ypos, angle, t, data)
%
% updates the data vector with a new struct object
%
updateddb = [db; struct('x',xpos, 'y',ypos, 'a',angle, 'type',t, 'data',data
    )];

end
```

distancecheck.m

```
function state = distancecheck(db,last)

% Funktion for kontroll om Arne maste vaja
distance_full = db.data(:,2);
distance_small = distance_full(8:14); % Sparar avstandsvarden for nogot
    snavare vinklar
if min(distance_small) < 50 % Ifall att nagon av avstandsvardena skulle
    understiga 50 cm ska Arne svanga

    right = mean(distance_full(1:7));
    left = mean(distance_full(15:21));
    if right > left % Om hoger medelvarde ar storre an vanster sa satts
        state till 1
        state = 1;
        if last == -1 % Kollar om arne svangde vanster sist, fortsatter
            isafall att svanga vanster
            state = -1;
        end
    else
        state = -1; % Vanster medelvarde storre dvs svang vanster
        if last == 1 % Kollar om arne svangde hoger sist, fortsatter isafall
             att svanga hoger
            state = 1;
        end
    end

else

    state = 0; % Arne kan kora rakt fram

end

end
```

serial_setup.m

```
function serport = serial_setup
%
% Initiates the serial communication
%
```

```
5  serport = serial('COM1', 'BaudRate', 9600, 'Parity', 'none', 'StopBits', 1)
       ;
6  fopen(serport);
7
8  end
```

flushbuffer.m

```
1  function flushbuffer(s)
2  %
3  % Flushes the serial recieve buffer if there is any data in it.
4  %
5  if s.BytesAvailable > 0, fread(s, s.BytesAvailable); end
6  end
```

sendpacket.m

```
1  function sendpacket(s, firstbyte, secondbyte, thirdbyte)
2  %
3  % Sends a packet.
4  %
5  packet = zeros(1, 32);
6  packet(1) = firstbyte;
7  packet(2) = secondbyte;
8  packet(3) = thirdbyte;
9  fwrite(s, packet);
10 flushbuffer(s);
11 end
```

turnleft45.m

```
1  function turnleft45(s)
2  % Skickar paket till Arne med kommandot svang vanster 17 steg
3  sendpacket(s,4,17,0);
4  while s.BytesAvailable < 32, end
5  flushbuffer(s);
6
7  end
```

forward.m

```
1  function forward(s)
2  % Skickar paket till Arne med kommandot kor framat 100 steg vilket motsvarar
       ungefar 25 cm
3  sendpacket(s,2,100,0);
4  while s.BytesAvailable < 32, end
5  flushbuffer(s);
6
7  end
```

B.4.2 Databehandling och visualisering

plot_lvl0.m

```
function plot_lvl0(db)
%
% Plots the robot position, and all measurement points.
%
hold on

if db.type == 0
    arnex = db.x;
    arney = db.y;
    arnea = db.a;

    sensorx = arnex+cosd(arnea)*4.6;
    sensory = arney+sind(arnea)*4.6;
    sensoraadj = -90;

    sensorangle = db.data(:,1);
    sensordistance = db.data(:,2);

    realangle = sensorangle+sensoraadj+arnea;
    realdistance = sensordistance;

    for j = 1:length(realdistance)
        if (realdistance(j) == 10) || (realdistance(j) == 210) % 10 and 210
            - out of range, draw a red line and a cross
            invalidx = sensorx + cosd(realangle(j))*realdistance(j);
            invalidy = sensory + sind(realangle(j))*realdistance(j);
            plot([sensorx invalidx], [sensory invalidy], '--', 'Color', [.5
                0 0]);
            plot(invalidx, invalidy, 'x', 'MarkerEdgeColor', [.5 0 0], '
                MarkerFaceColor', [.25 0 0], 'MarkerSize', 4);
        else % in range, draw a green line and a triangle
            validx = sensorx + cosd(realangle(j))*realdistance(j);
            validy = sensory + sind(realangle(j))*realdistance(j);
            plot([sensorx validx], [sensory validy], ':', 'Color', [0 .5 0])
                ;
            plot(validx, validy, '^', 'MarkerEdgeColor', [0 1 0], '
                MarkerFaceColor', [0 .75 0], 'MarkerSize', 4);
        end
    end
    plot([arnex sensorx], [arney sensory], '-', 'Color', [.5 .5 .5]); % draw
         the robot position
    plot(arnex, arney, 'o', 'MarkerEdgeColor', [.5 .5 .5], 'MarkerFaceColor'
        , [0 0 0], 'MarkerSize', 8);
    plot(sensorx, sensory, 'o', 'MarkerEdgeColor', [.5 .5 .5], '
        MarkerFaceColor', [.25 .25 .25], 'MarkerSize', 7);
end

set(gca, 'Color', [0 0 0]);
hold off

end
```

lvl0_to_lvl1.m

```
function db1 = lvl0_to_lvl1(db0)
%
% Changes all measuremnt angles from robot relative to initial robot angle relative
% and calculates x,y coordinates for all measured points.
%
db1 = [];

for i = 1:length(db0)
    arnex = db0(i).x;
    arney = db0(i).y;
    arnea = db0(i).a;

    sensorx = arnex+cosd(arnea)*4.6;
    sensory = arney+sind(arnea)*4.6;
    sensoraadj = -90;

    sensorangle = db0(i).data(:,1);
    sensordistance = db0(i).data(:,2);

    a = sensorangle+sensoraadj+arnea;
    d = sensordistance;
    x = []; y = [];

    for j = 1:length(sensorangle)
        x(j) = sensorx + cosd(a(j))*d(j);
        y(j) = sensory + sind(a(j))*d(j);
    end
    db1 = [db1; struct('sx', sensorx, 'sy', sensory, 'da', a, 'dd', d, 'dx', x, 'dy', y)];
end


end
```

lvl1_to_matrix.m

```
function m = lvl1_to_matrix(db, gridsize)
%
% Takes a measurment data vector and a gridsize and creates a big enough matrix
% to contain all the data, and then changes the value at each index depending on
% the number of obstacle points found in the corresponding area and the number
% of ir beams that has passed the area without colliding with an obstacle.
%
magic0 = 1; % constants on how much to change the matrix depending on if data points have been or not been found in a certain area
magic1 = 1.05;
magic2 = 1.1;
minx = min(round([db.dx])); % what is the minimal and maximal values of the data points
maxx = max(round([db.dx]));
miny = min(round([db.dy]));
maxy = max(round([db.dy]));
```

```
15
16 width = maxx-minx; % environment dimensions
17 height = maxy-miny;
18
19 mw = ceil(width/gridsize); % matrix dimensions
20 mh = ceil(height/gridsize);
21
22 m = ones(mh, mw)*magic0;
23 for i = 1:length(db)
24     x0 = round(coerce(db(i).sx, minx, maxx, 1, mw)); % starting point
25     y0 = round(coerce(db(i).sy, miny, maxy, 1, mh));
26
27     m(y0, x0) = m(y0, x0)/magic1; % change matrix value
28
29     for j = 1:length(db(i).dx)
30
31         x1 = round(coerce(db(i).dx(j), minx, maxx, 1, mw)); % finish point
32         y1 = round(coerce(db(i).dy(j), miny, maxy, 1, mh));
33         points = round(bresenham([x0; y0], [x1; y1])); % calculate points in
               between
34
35         for k = 2:(length(points)-1)
36             x = points(1,k);
37             y = points(2,k);
38             m(y,x) = m(y,x)/magic1; % change matrix value (nothing found
                   here)
39         end
40         x = points(1,length(points));
41         y = points(2,length(points));
42         if db(i).dd(j) == 210
43             m(y,x) = m(y,x)/magic1; % change matrix value (nothing found
                   here (210=out of range))
44         else
45             m(y,x) = m(y,x)*magic2; % change matrix value (something found
                   here)
46         end
47     end
48 end
49 end
```

coerce.m

```
1 function newandimproved = coerce(number, oldmin, oldmax, newmin, newmax)
2 %
3 % Scales a number from one range to another.
4 %
5 newandimproved = (number-oldmin)*(newmax-newmin)/(oldmax-oldmin)+newmin;
6 end
```

bresenham.m

```
1 function points = bresenham(p0, p1)
2 %
3 % Creates a list of points needed to traverse when going from p0 to p1.
4 %
```

```
5  points = [];
6  if (p0(1) ~= p1(1)) || (p0(2) ~= p1(2))
7      x0 = p0(1); x1 = p1(1);
8      y0 = p0(2); y1 = p1(2);
9      if abs(x1-x0) > abs(y1-y0)
10         % x as iteration variable
11         deltax = 1;
12         if x0 > x1
13             deltax = -1;
14         end
15         deltay = (y1-y0)/abs(x1-x0);
16         y = y0;
17         for x = x0:deltax:x1
18             points = [points [x; y]];
19             y = y + deltay;
20         end
21     else
22         % y as iteration variable
23         deltay = 1;
24         if y0 > y1
25             deltay = -1;
26         end
27         deltax = (x1-x0)/abs(y1-y0);
28         x = x0;
29         for y = y0:deltay:y1
30             points = [points [x; y]];
31             x = x + deltax;
32         end
33     end
34 else
35     points = [p0];
36 end
37
38 end
```

smoothplot.m

```
1  function smoothplot(m)
2  %
3  % This code does a simple low pass filter on the matrix argument (to smooth
       it)
4  % and then creates a surf and contour plot of the result.
5  %
6  ms = conv2(m, [.5/sqrt(2) .5 .5/sqrt(2); .5 sqrt(2) .5; .5/sqrt(2) .5 .5/
       sqrt(2)]);
7  z = size(ms);
8  ms = ms(3:(z(1)-2),3:(z(2)-2));
9  figure
10 surf(ms)
11 alpha .8
12 colormap summer
13
14 figure
15 contourf(ms, linspace(min(min(ms)), max(max(ms)), 25))
16 colormap summer
```