

TSEA56 - Kandidatprojekt i elektronik LIPS Teknisk dokumentation för PigBot

Version 1.1

Grupp 4

Hynén Ulfsjöö, Olle ollul666

Wasteson, Emil emiwa068

Tronje, Elena eletr654

Gustafsson, Lovisa lovgu777

Inge, Zimon zimin415

Strömberg, Isak isast763

2 juni 2016

Status

Granskad	-	-
Godkänd	-	-

PROJEKTIDENTITET

2016/VT, Undsättningsrobot Gr. 4

Linköpings tekniska högskola, ISY

Namn	Ansvar	Telefon	E-post
Isak Strömberg (IS)	Projektledare	073-980 38 50	isast763@student.liu.se
Olle Hynén Ulfsjöö (OHU)	Dokumentansvarig	070-072 91 84	ollul666@student.liu.se
Emil Wasteson (EW)	Hårdvaruansvarig	076-836 61 66	emiwa068@student.liu.se
Elena Tronje (ET)	Mjukvaruansvarig	072-276 92 93	eletr654@student.liu.se
Zimon Inge (ZI)	Testansvarig	070-171 35 18	zimin415@student.liu.se
Lovisa Gustafsson (LG)	Leveransansvarig	070-210 32 53	lovgu777@student.liu.se

E-postlista för hela gruppen: isast763@student.liu.se

Kund: ISY, Linköpings universitet tel: 013-28 10 00, fax: 013-13 92 82

Kontaktperson hos kund: Mattias Krysander

tel: 013-28 21 98, e-post: matkr@isy.liu.se

Kursansvarig: Tomas Svensson

tel: 013-28 13 68, e-post: tomass@isy.liu.se

Handledare: Peter Johansson

tel: 013-28 13 45, e-post: peter.a.johansson@liu.se

Innehåll

1 Inledning	1
2 Produkten	1
2.1 Begränsningar	1
3 Teori	4
3.1 Labyrintnavigering	4
3.1.1 Väggföljare	4
3.1.2 <i>Dead-end filling</i>	4
3.2 Beräkning av kortaste väg	5
3.2.1 Dijkstras algoritm	5
3.2.2 A*	5
4 Systemet	6
4.1 Beskrivning av systemet	6
5 Modulerna	8
5.1 Huvudmodulen	8
5.1.1 Mjukvara	8
5.1.2 Styrläge	9
5.1.3 Intern kartläggning	9
5.1.4 Beräkning av kortaste väg	10
5.2 Sensormodulen	12
5.2.1 Mjukvara	13
5.2.2 Lidar-Lite v2, avståndssensor	13
5.2.3 SFH300, reflexsensor	13
5.2.4 GP2D120, avståndssensor	14
5.2.5 MLX90609, gyro	14
5.2.6 IRM-8601-S, IR-detektor	14
5.3 Styrmodulen	14
5.3.1 Mjukvara	15
5.3.2 PWM	15
5.3.3 LCD-display	16
5.3.4 Styrlägen	16
5.4 Datormodulen	17
5.4.1 Model	17
5.4.2 View	18
5.4.3 Controller	18
6 Intermodulär kommunikation	19
6.1 Informationsflöde	19
6.1.1 Kommunikationsprotokoll - styrkommandon	20
6.1.2 Kommunikationsprotokoll - kartinformation	20
6.1.3 Kommunikationsprotokoll - sensordata	21
6.1.4 Kommunikationsprotokoll - styrinställningar	21

7 Slutsatser	23
Referenser	24
O Kopplingsschema	24
P Kodexempel	27
P.1 Huvudmodul	27
P.2 Sensormodul	30
P.3 Styrmodul	31
P.4 Datormodul	33

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
1.1	-	Andra utkastet	Grupp 4	-
1.0	-	Första utkastet	Grupp 4	-

1 Inledning

Projektets syfte var att konstruera en undsättningsrobot med kartläggning. Detta dokument syftar till att ge en detaljerad teknisk beskrivning av projektet. Innehållet består av beskrivningar av systemets olika delar, dess mjuk- och hårdvarukomponenter samt hur dessa är implementerade.

2 Produkten

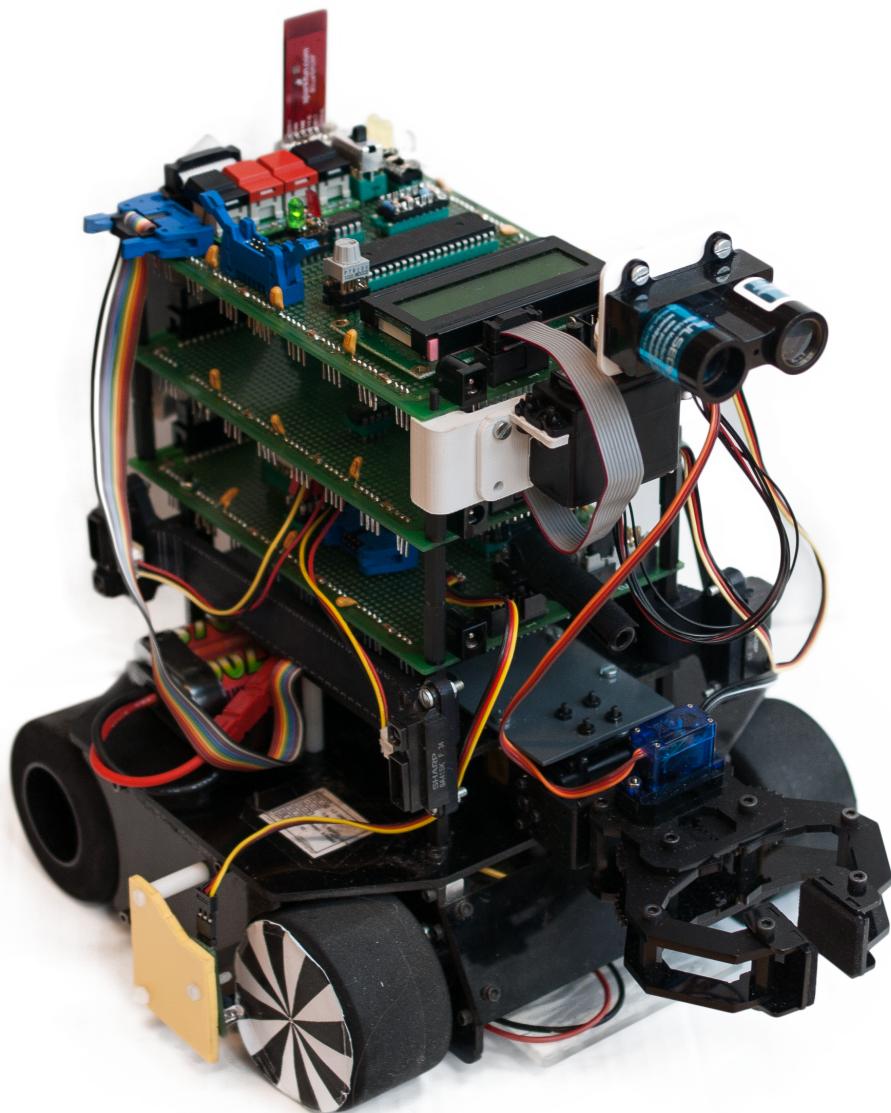
Produkten som konstruerats är en undsättningsrobot. Den består av ett chassi inklusive ett batteri, en av/på-brytare samt en gripklo, se figur 1 för den färdiga produkten. På chassit är huvudmodulen, sensormodulen och styrmodulen ihopkopplade med en I²C-buss som sköter kommunikationen mellan dem. Roboten kommunicerar med datormodulen via Bluetooth®. En gripklo kontrolleras av styrmodulen och sensormodulen tar in data från sensorerna som är kopplade till denna. För att montera sensorerna på chassit har fästen utskrivna med 3D-skrivare använts. En översiktlig bild av hur komponenterna är placerade på roboten återfinns i figur 2.

Varje modul har en processor och övrig nödvändig hårdvara, som lågpassfilter och brytare, kopplade på ett virkort. Virkorten kan användas som grund för att tillverka kretskort med motsvarande funktion.

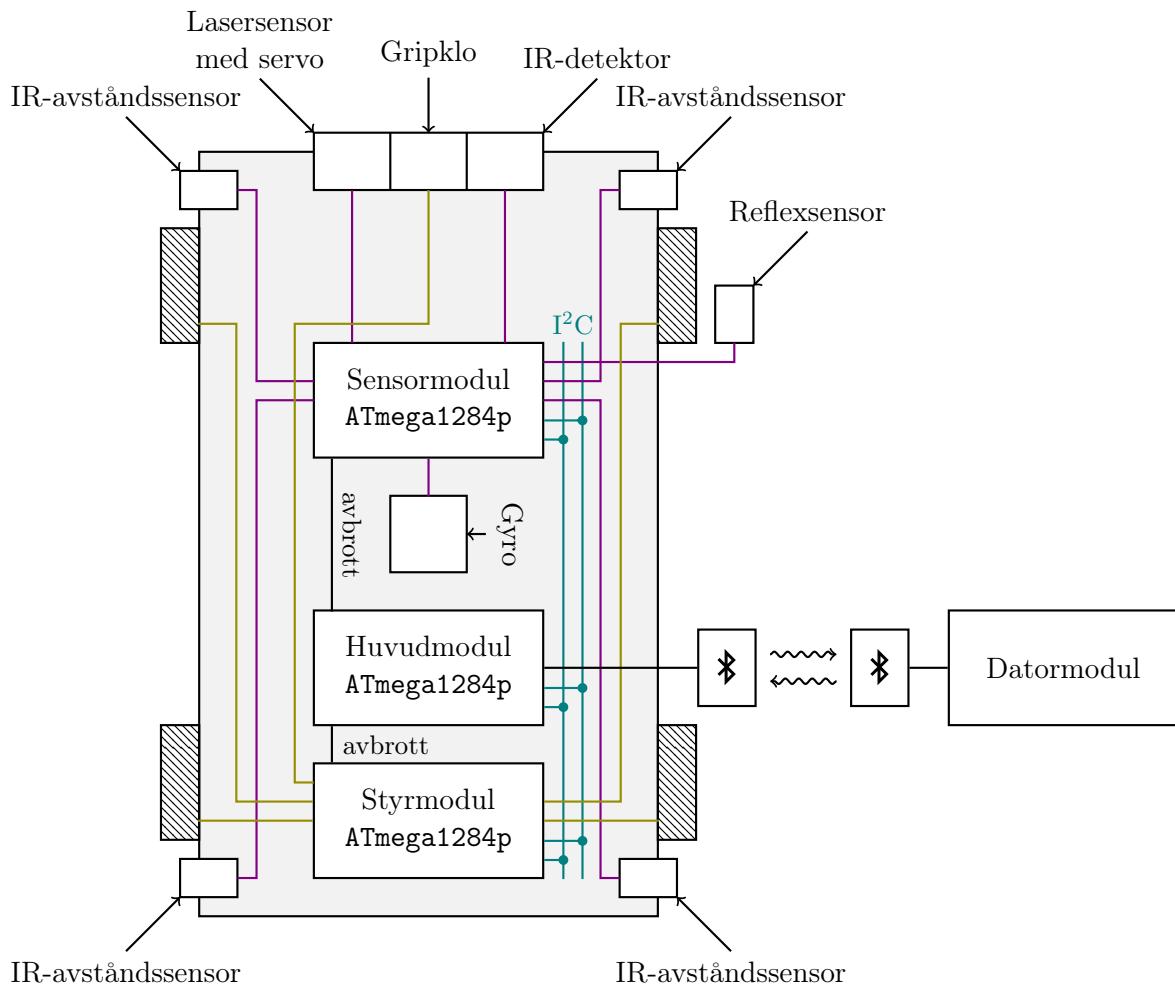
Robotens utför följande uppgift. Den söker av ett grottsystem och identifierar en nödställd som sänder ut IR-ljus. När målet är identifierat bestäms kortaste vägen mellan labyrintens ingång och målet. Hela labyrinten söks ej av utan en avsökningsalgoritm som går att läsa om i avsnitt 5.1.3 används. Efter det hämtar roboten en fornödenhet vid starten, kör kortaste vägen till den nödställda och lämnar av fornödenheten för att sedan ta sig tillbaka samma väg till starten. Utforskning och intern kartläggning av labyrinten sker simultant. Informationen lagras för att kunna skickas vidare till datormodulen för uppritning.

2.1 Begränsningar

Roboten är begränsad till att navigera i korridorer och kräver att alla korsningar består av 90-gradersvinklar. Labyrinten måste bestå av kartmoduler med storlek 40x40 centimeter, se banspecifikation i appendix B.



Figur 1: Den slutgiltiga produkten



Figur 2: Det totala systemet

3 Teori

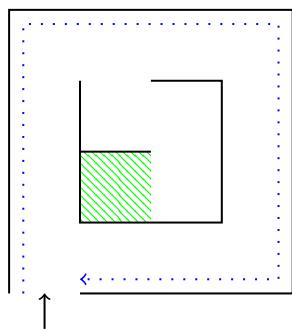
I följande avsnitt lyfts teori som är nödvändig för förståelse av implementationen.

3.1 Labyrintnavigering

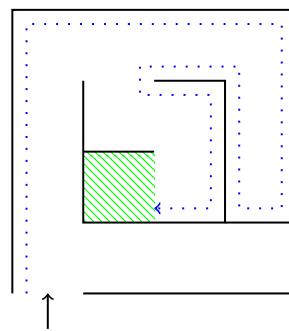
Nedan beskrivs principer för avsökning och kartläggning av labyrinter.

3.1.1 Väggföljare

Ifall labyrinten är enkelt sammanhängande räcker en väggföljar-algoritm för att kartlägga alla korridorer. I en enkelt sammanhängande labyrint är alla väggar kopplade till varandra, se figur 3 för exempel. En väggföljar-algoritm är antingen av höger- eller vänstertyp och bygger på att en av korridorens sidor alltid följs. Ifall algoritmen är av högertyp följs höger sida av korridoren, vice versa för vänstertyp.



(a) En inte enkelt sammanhängande labyrint



(b) En enkelt sammanhängande labyrint

Figur 3: Exempel på labyrint och rutt som skulle tas av en väggföljningsalgoritm av vänstertyp

Fördelar med algoritmen är att den är enkel att implementera men avsaknaden av intelligens gör att utforskanet av labyrinten tar lång tid. Algoritmen klarar endast att utforska de allra enklaste typer av labyrinter och ifall labyrinten inte är enkelt sammanhängande kan roboten fastna i en oändlig loop.

3.1.2 Dead-end filling

Dead-end filling är en djup-först-algoritm som utforskar en väg tills en återvändsgränd upptäcks. Därefter åker roboten tillbaka till den första korsningen som har outforskade utgångar och kartlägger dessa tills en ny återvändsgränd upptäcks. Denna process återupprepas tills hela labyrinten är kartlagd. För en mer ingående beskrivning av denna algoritmen se förstudien, appendix G.

3.2 Beräkning av kortaste väg

Nedan beskrivs två algoritmer för att ta fram kortaste väg.

3.2.1 Dijkstras algoritm

Dijkstras algoritm är en girig, bäst-först-algoritm. Algoritmen beräknar den billigaste vägen från startnoden till samtliga noder och fungerar på nätverk med icke-negativa bågkostnader.

Vid starten definieras två mängder, en med de avsökta och en med de oavsökta noderna. De oavsökta noderna har ett oändligt nodpris och startnoden har nodpriset noll. Med ursprung i startnoden söker algoritmen igenom alla grannar. Nodpriset uppdateras i det fall då sambandet

$$y_i + c_{i,j} < y_j$$

gäller, där y_i är nodpriset för den nod som söks av, y_j är nodpriset för nodgrannen och $c_{i,j}$ är kostnaden mellan nod i och j . Ovanstående görs successivt med ursprung i den nod med billigast nodpris som även tillhör den oavsökta mängden, tills slutnoden har undersöks. Den kortaste vägen kan sedan nästlas upp, förutsatt att den föregående noden som bidrog till den kortaste vägen sparades. För en mer ingående beskrivning av denna algoritm se förstudien, appendix G.

3.2.2 A*

A* är ytterligare en bäst-först-algoritm med målet att finna den billigaste vägen mellan två noder. I grund och botten finns samma maskineri som i Dijkstras algoritm, med skillnaden att noder avsöks i stigande ordning enligt

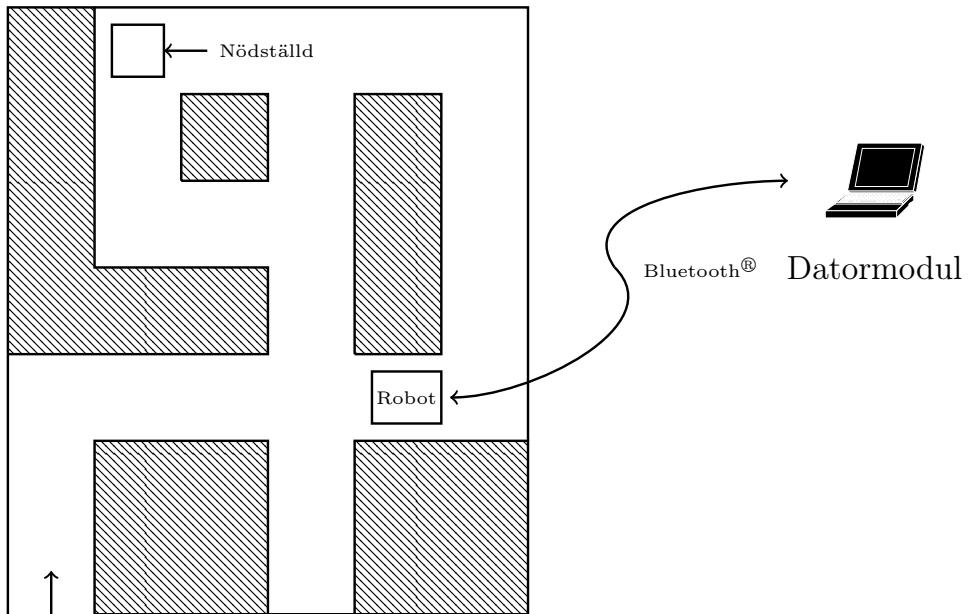
$$f(n) = g(n) + h(n)$$

där $g(n)$ utgör kostnaden av rutten från startnod till nod n och $h(n)$ en heuristik som estimerar den billigaste vägen från nod n till slutnod. Med andra ord är Dijkstras algoritm ett specialfall av A* där $h(n) = 0$.

Funktionen $h(n)$ kan ses som en strafffunktion som avtar nära slutnoden. A* kan alltså, till skillnad från Dijkstras algoritm, se framåt i nätverket och kan därför ta bättre beslut om vilka noder som ska uppdateras. För en mer ingående beskrivning av denna algoritm se förstudien, appendix G.

4 Systemet

Roboten i sin miljö finns illustrerad i figur 4. Den kan både köras manuellt och autonomt, något som bestäms av en brytare på roboten. Kommunikationen med datormodulen är dubbeldirkigad via Bluetooth®. Roboten ska klara sitt uppdrag utan kommunikation med datormodulen, det vill säga att kartläggning, styrning och optimering av kortaste väg sker lokalt hos roboten. Banan är uppbyggd enligt banspecifikationen och uppdraget utförs enligt tävlingsreglerna, se appendix B.



Figur 4: Översikt av banan

4.1 Beskrivning av systemet

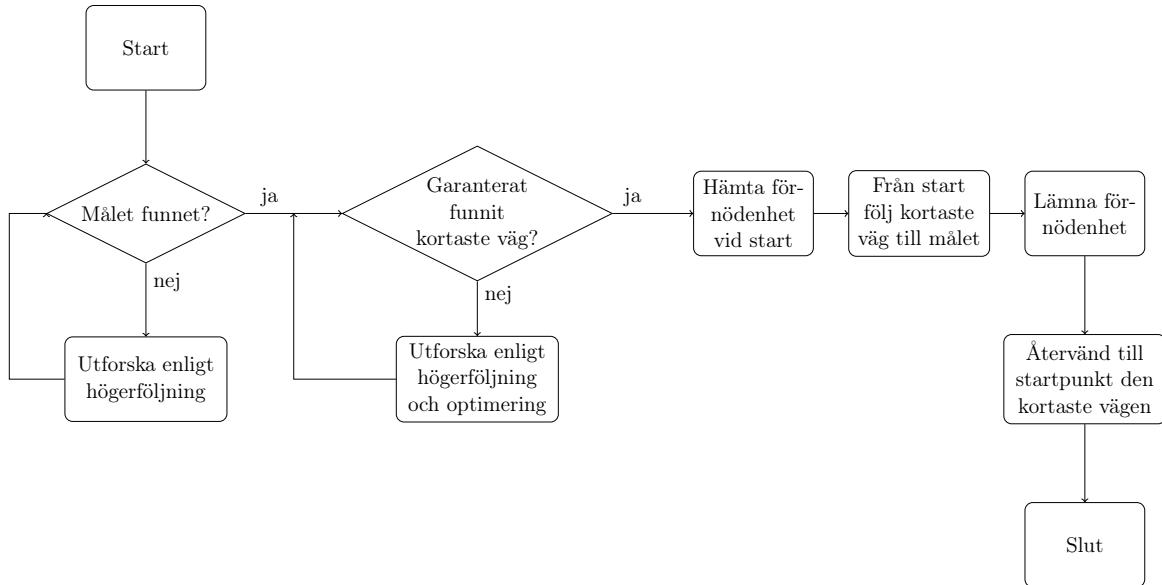
Roboten navigerar med hjälp av värden som fås från sensormodulens sensorer. En lasersensor som är fäst på ett servo mäter avstånd framåt, fyra IR-sensorer mäter robotens avstånd till sidoväggarna, ett gyroskop mäter robotens vinkelhastighet och en IR-detektor identifierar den nödställda. En reflexsensor sitter monterad mot hjulet för att beräkna tillryggalagd sträcka. Med jämna intervall kommunicerar sensormodulen dessa värden till huvudmodulen, som i sin tur vidarebefordrar dem till styrmodulen.

I styrmodulen är en regleringsmodell implementerad, vilken säkerställer att roboten färdas i mitten av korridorerna samt kan rotera både 90 och 180 grader utan att stöta emot väggar. Önskade värden kan skrivas ut på en LCD-display kopplad till styrmodulen.

Under färdens sker autonom kartläggning och beräkning av kortaste väg mellan ingången och den nödställda. När roboten är uppkopplad mot datormodulen ritar mjukvaran på datorn successivt upp en karta och presenterar utvalda mätvärden i realtid.

När den kortaste vägen är funnen använder roboten denna rutt för att förse den nödställda med en förnödenhet. Fornödenheten transporteras med hjälp av robotens gripklo som kontrolleras av styrmodulen.

Det som avgör vad roboten tar för beslut är om kortaste väg kan garanteras och om målet är funnet, vilket illustreras i figur 5.



Figur 5: Övergripande blockschema för systemet

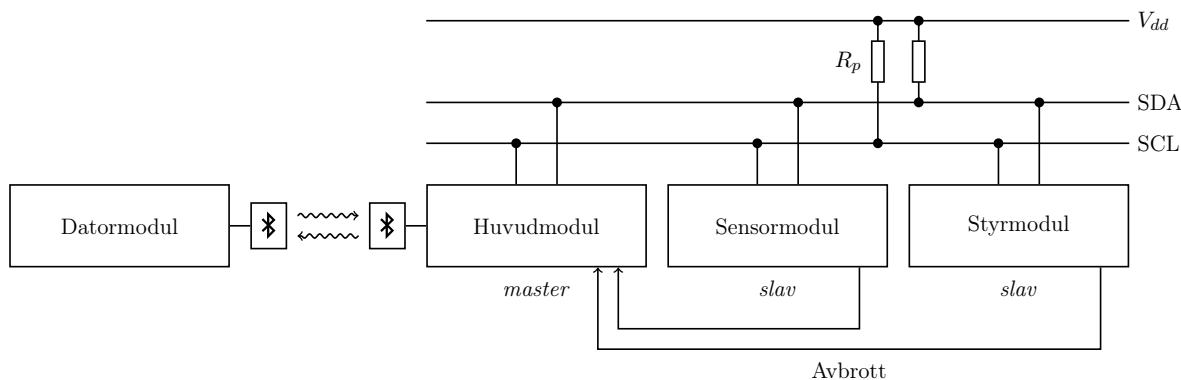
5 Modulerna

Nedan följer detaljerade beskrivningar av respektive modul.

5.1 Huvudmodulen

Huvudmodulen har som uppgift att ta emot och förmedla information mellan de andra modulerna, hantera den interna kartläggningen samt att ta alla övergripande beslut.

Bluetooth® används för att kommunicera med datormodulen och en avbrottsstyrd I²C-buss används för att kommunicera med sensor- och styrmodulen, vilket visas i figur 6. Vad gäller prioritering av sensor- och styrmodul har styrmodulen kopplats till en extern avbrottssingång med högre prioritet än sensormodulen.



Figur 6: Intermodulär kommunikation

Komponenterna som ingår i robotens huvudmodul är

- Mikroprocessor ATmega1284p
- Kristalloscillator EX0-3 på 14.745 MHz
- Bluetooth®-modul
- Resistorer
- Kondensator
- Studsfri tryckknapp

Huvudmodulen är virad på ett virkort enligt kretsschemat i figur ?? och är monterat på robotens chassi.

Nedan beskrivs huvudmodulens olika uppgifter.

5.1.1 Mjukvara

I huvudmodulen ingår följande mjukvara:

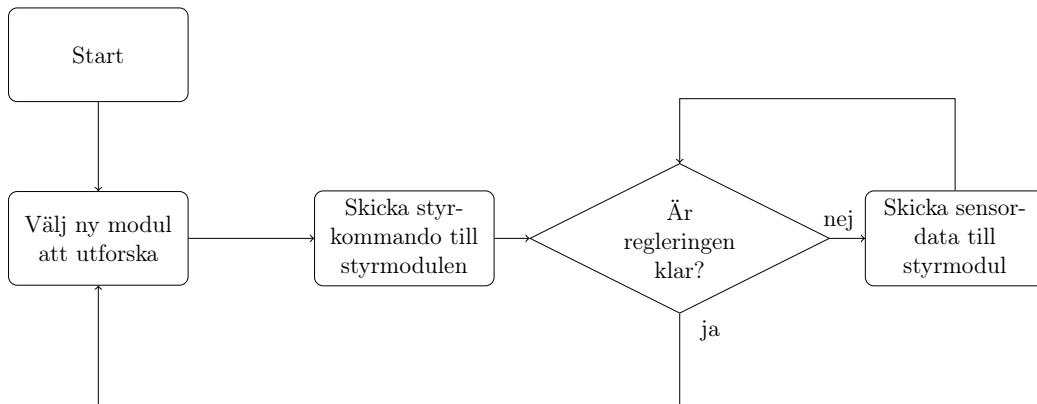
huvudMain.c som innehåller *main*-funktionen, I²C-kommunikationen och Bluetooth®-kommunikationen. I huvudloopen är det endast styrläget som uppdateras.

I2C_master.h som är ett paket för masterenheten på I²C-bussen.

searchPath.h som är ett paket som hanterar avsökning, kartläggning och beräkning av kortaste väg.

5.1.2 Styrläge

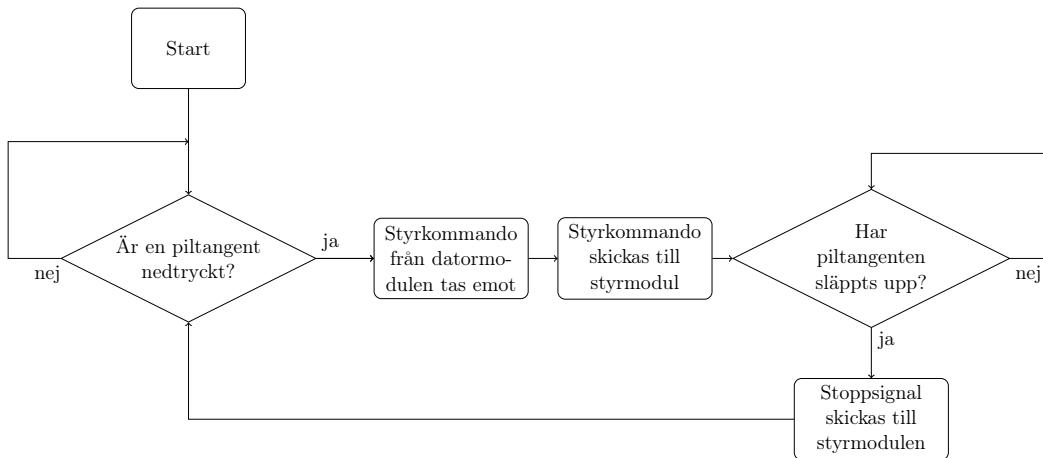
Det finns två olika styrlägen för roboten, ett autonomt och ett manuellt läge. Hårdvarumässigt styrs detta med hjälp av en brytare kopplad till huvudmodulen. När det autonoma läget är aktiverat tas körbeslut utifrån en avsökningsalgoritm som i grunden använder sig av väggföljning av högertyp och *dead-end filling*. Beslutet om vilken modul som ska besökas härnäst baseras på datan som skickas från sensormodulen och på den interna kartläggningen. Detta förklaras närmare i avsnitt 5.1.3 och 5.1.4. När beslutet är taget skickas motsvarande kommando till styrmodulen som sedan utför önskad operation. Ett nytt beslut tas när styrmodulen anser sig vara färdig med nuvarande kommando och begär ett nytt genom avbrott. Hela kommunikationsflödet i autonomt läge finns beskrivet i figur 7. I manuellt läge skickas styrkommandon från datormodulen, via Bluetooth®, till huvudmodulen som sedan vidarebefordrar dessa till styrmodulen. Informationen som skickas baseras på vilken pil tangent på datormodulen som är nedtryckt enligt flödet i figur 8.



Figur 7: Flödesschema som beskriver förloppet vid autonom styrning

5.1.3 Intern kartläggning

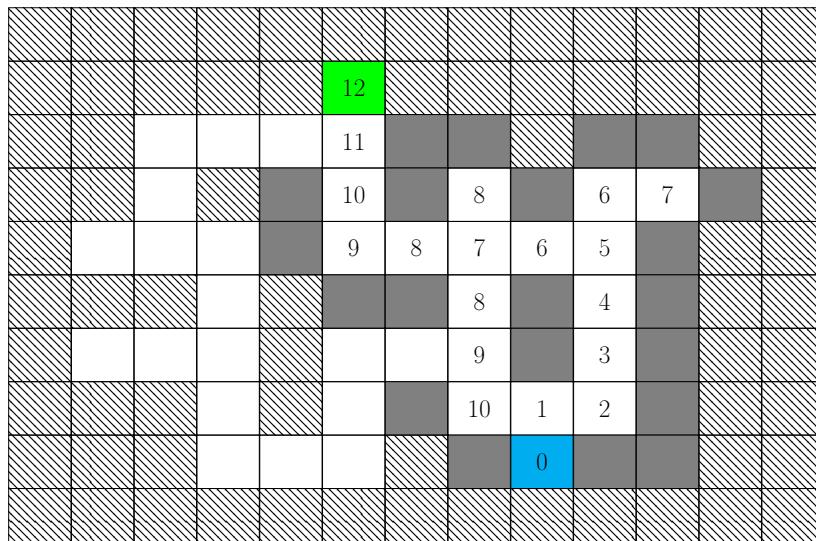
Den interna kartläggningen sker kontinuerligt allt eftersom roboten utforskar labyrinten. Den interna kartan representeras av ett 31x18 fält där robotens startkoordinater är (16,1) och startriktningen är norrut. I fältet lagras varje ny kartmodul som roboten besöker och var sensorerna har registrerat väggar. Längs den tillryggalagda sträckan får varje kartmodul ett värde som motsvarar den kortaste distansen roboten behöver gå för att ta sig dit (med befrintlig avsökningsalgoritm), se figur 9. Efter varje besökt modul med utforskade vägar



Figur 8: Flödesschema som beskriver förloppet vid manuell styrning

skickas de uppdaterade värdena i fältet till datormodulen för uppritning, förutsatt att den är ansluten. Är så ej fallet skickas informationen till datormodulen när anslutning har upprättats.

Återvändsgränder där mål ej hittas sparas i ett annat 31x18 fält, markerade röda i figur 10, för att förenkla beräkningen av kortaste väg.

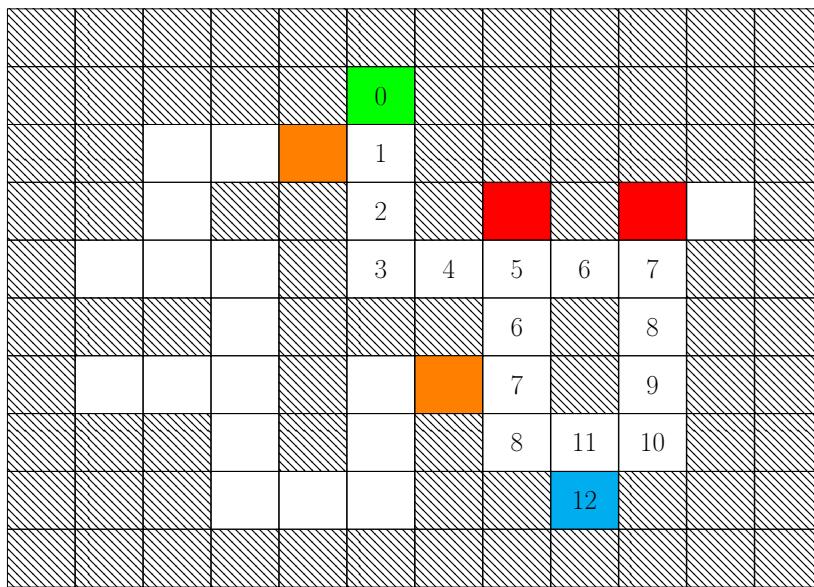


Figur 9: Schematisk bild av den interna kartan

5.1.4 Beräkning av kortaste väg

Beräkningen av kortaste vägen till målet sker med utgångspunkt från A^{*}-algoritmen. När målet är funnet numreras varje nod som roboten besöker från målet med en siffra som representerar ett avstånd till målet, se figur 10. För att inte utforska mer än nödvändigt

används en pessimistisk skattning av avståndet från start till mål som jämförelsevärde. Detta värde motsvarar det antal steg robotens första väg från start till mål blev, återvändsgränder borträknat. Under den fortsatta utforskningen av labyrinten jämförs denna skattning med summan av köravståndet från målet, det vill säga värdet den kartmodulen skulle ha i figur 10, och manhattanavståndet från start till aktuell nod. Är det avståndet lika med eller större än skattningen är det inte nödvändigt att utforska mer åt det hålet och motsvarande modul markeras för att i beräkningen inte ta hänsyn till den riktningen, se orangemarkerade moduler i figur 10. Skulle en bättre pessimistisk skattning dyka upp på vägen uppdateras värdet för att ytterligare minska ned antalet moduler som måste besökas.

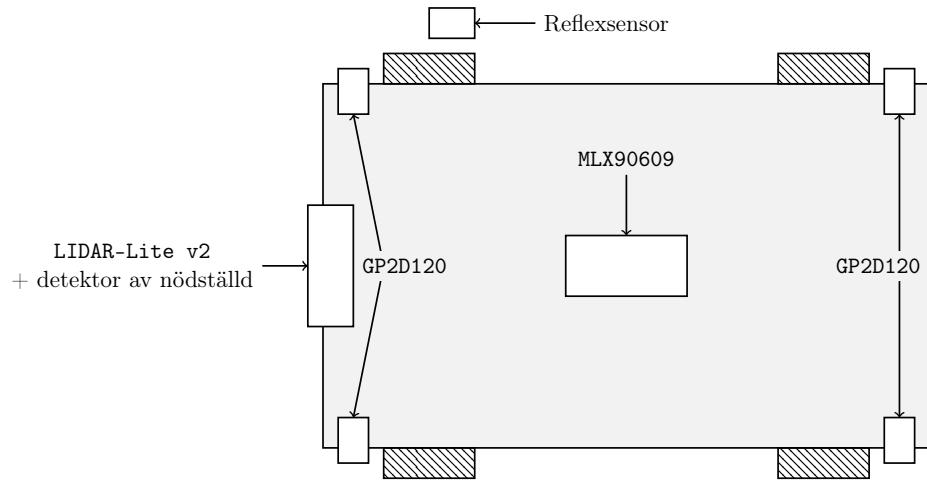


Figur 10: Schematisk bild av kartan för beräkning av kortaste väg

Kortaste väg från start till mål fås då genom att, för varje nod, välja den närliggande nod med lägst siffra enligt figur 10. De orange- och rödmarkerade kartmodulerna är implementerade med högre siffror än vad roboten skulle kunna köra enligt tävlingsreglerna, se appendix B.

5.2 Sensormodulen

Robotens sensormodul har som uppgift att läsa av sensorerna och kommunicera datan till huvudmodulen. De sensorer som används och placeringar av dessa illustreras i figur 11.



Figur 11: Sensorplacering

De komponenter som ingår i robotens sensormodul är följande:

- Lasersensor LIDAR-Lite v2
- IR-sensorer GP2D120
- Reflexsensor SFH300
- Gyro MLX90609
- IR-detektor IRM-8601-S
- Mikroprocessor ATmega1284p
- Kristalloscillator EX0-3 på 16 MHz
- Resistorer
- Kondensator
- Lågpassfilter
- Studsfree tryckknapp

Sensormodulen är ihopkopplad enligt kopplingsschema i figur ???. Den är sammankopplad på ett separat virkort, som är monterat på robotens chassi. Nedan följer utförligare beskrivningar av sensormodulens hårdvara, mjukvara och gränssnitt.

5.2.1 Mjukvara

I sensormodulen ingår följande mjukvara:

sensorModule.c som innehåller modulens *main*-funktion, i vilken omvandling av sensorvärdet till SI-enhet sker. Den innehåller dessutom kod för I²C-kommunikation mellan modulerna. I main-funktionens huvudloop läggs de omvandlade sensorenheterna in i ett fält, vilket senare kommuniceras till huvudmodulen.

sensorInit.h som är ett paket som initierar de olika gränssnitten för överföring av sensordata mellan mikroprocessorn och de olika sensorerna.

I2C_slave.h som är ett paket för slavarna inom I²C-bussen, lämpligt i detta fall då sensormodulen är slav i kommunikationen med huvudmodulen.

5.2.2 Lidar-Lite v2, avståndssensor

För att mäta avstånd framåt används lasersensorn **Lidar-Lite v2**. Den sänder ut en laserpuls och mäter tiden det tar för denna att reflekterats och återvända. På så sätt kan avstånd upp till 40 meter uppmäts med en felmarginal på 2,5 centimeter. Den kan dock ge felaktiga värden när den opererar i en omgivning som labyrinten. Det beror på att det finns en risk att sensorn mäter avstånd till en sidokorridor, när den tror att den mäter avståndet till väggen rakt fram.

Kommunikationen av sensordata mellan **Lidar-Lite v2** och mikroprocessorn **ATmega1284p** sker via pulsbreddsmodulering. Pulsbreddens arbetscykel är proportionell mot sensorns avstånd till närmaste föremål, vilket gör omvandlingen till SI-enhet enkel. För att mäta pulsbredden kopplas lasersensorns trigger pin till en av mikroprocessorns externa avbrottssingångar, se kopplingsschema i figur ??.

5.2.3 SFH300, reflexsensor

För att komplettera avståndsmätningen används även en reflexsensor av modell **SFH300**. Sensorn sänder ut ljus och mäter hur mycket av ljuset som reflekteras. Genom att fästa en skiva indelad i 16 lika stora tårtbitar, varannan vit och varannan svart, på ett av hjulen kan tillryggalagt avstånd bestämmas.

Hjulens omkrets kan mätas och där efter divideras med 16. På så sätt erhålls hur långt roboten åker per tårtbit. Då utsignalen är analog krävs AD-omvandling, varefter ett digitalt värde mellan 0 och 255 erhålls. Genom experiment kan ett lämpligt referensvärde hittas där värden över detta tolkas som svart yta och lägre värden tolkas som vit yta. Detta fungerar eftersom vita ytor reflekterar mycket mer ljus än vad svarta ytor gör. Genom att räkna hur många ljusskiftnings som har skett kan sedan tillryggalagt avstånd erhållas.

5.2.4 GP2D120, avståndssensor

För att mäta avståndet till sidoväggarna används fyra IR-sensorer av modell GP2D120. Avståndet mäts genom att IR-ljus med en viss våglängd skickas ut från sensorn, reflekteras mot väggarna för att sedan detekteras av sensorn. Genom att mäta den vinkel reflektionen sker med kan avståndet till väggen bedömas. Utsignalen är analog i form av spänning och behöver lågpassfiltreras på grund av mängden högfrekvent brus i signalen. Lågpassfiltreringen sker genom att leda signalen genom en RC-krets.

Portarna A0-A7 på ATmega1284p kan användas som AD-omvandlare, vilket utnyttjas för att generera ett digitalt värde. Det digitala värdet är inte linjärt beroende av avståndet utan sambandet är mer komplext. För att hitta sambandet genomförs tester för att mäta vilket avstånd som motsvarar vilket digitalt värde. Därefter kan funktionen bestämmas med hjälp av lämpligt dataprogram.

5.2.5 MLX90609, gyro

För att kunna utföra lagom stor rotation vid svängar används gyrot MLX90609 som mäter den momentana vinkelhastigheten. Att bestämma hur många grader robotten roterar görs genom att summera värdet på vinkelhastigheten med tillräckligt hög samplingsfrekvens, vilket motsvarar integrering av vinkelhastigheten. Kommunikation av informationen mellan sensorn och mikroprocessorn sker via ett SPI-gränssnitt, där mikroprocessorn fungerar som SPI-master. Vinkelhastigheten representeras av elva bitar men kapas till åtta innan informationen kommuniceras vidare. Då de tre minst signifikanta bitarna tas bort blir precisionen något sämre men informationen rymms på en byte. Tester kan sedan genomföras för att bestämma vilket värde som motsvarar en grads rotation.

5.2.6 IRM-8601-S, IR-detektor

Då den nödställd sänder ut en IR-signal krävs identifikation av ett specifikt bitmönster. För detta ändamål används IR-detektorn IRM-8601-S. Bitmönstret identifieras genom att ansluta insignalen till en extern avbrottssingång på mikroprocessorn. TIMER0 används därefter för att räkna tiden på de höga bitarna hos insignalen som IR-detektorn identifierar. Stämmer tiden för de höga bitarna överens med det bitmönster som den nödställd ska sända ut är den nödställd funnen.

5.3 Styrmodulen

Styrmodulen är den del av roboten som tar emot och verkställer styrkommandon och ser till att roboten tar sig fram på önskat sätt. Det som hanteras är styrning av chassis motorer för att få roboten att röra på sig och att styra gripklon.

De komponenter som ingår i styrmodulen är

- Mikroprocessor ATmega1284p

- LCD-display JM162A
- Kristalloscillator EX0-3 på 16 MHz
- LED-dioder
- Resistorer
- Kondensator
- Potentiometer
- Studsfri tryckknapp

Styrmodulen kopplas enligt kretsschemat i figur ???. Kopplingen sker på ett virkort som sedan monteras på robotens chassi. Här efter följer detaljerade beskrivningar av styrmodulens olika delar.

5.3.1 Mjukvara

Mjukvaran till styrmodulen består av följande filer:

controlModule.c som innehåller *main*-funktionen, I²C-kommunikationen och regleringen. I huvudloopen är den endast LCD-displayen och LED-dioderna som uppdateras.

PWM.h som är ett paket för pulsbreddsmodulering.

LCD.h som är ett paket för LCD-displayen.

constants.h som är ett paket med definitioner av konstanter till controlModule.c.

I2C_slave.h som är ett paket för slavar inom I²C-bussen.

5.3.2 PWM

Pulsbreddsmodulering används för att styra respektive hjulpar, gripklon och servot för den främre sensorn. På ATMega1284p har varje timer två utgångar, utgång A och B. En timer kopplas således till båda hjulparen så att de delar samma inställningar gällande frekvens och periodtid men styrs med olika arbetscykler. Gripklon och servot för den främre sensorn delar i sin tur även samma egenskaper och tilldelas en gemensam timer. Samtliga timers som används är av 16-bitarstyp, men alla bitar behövs inte och därfor utnyttjas möjligheten att konfigurera om dem så att enbart de nödvändiga används. Nedan finns konfigurationerna beskrivna mer detaljerat, där ICR är ett register som kan sättas till önskat värde.

Syfte	Läge	Antal bitar	Prescale	Toppvärde
Hjulpar	Fast PWM, 10-bit	10 bitar	clk/64	0x03FF
Gripklo och sensor servo	Fast PWM	16 bitar	clk/64	ICR

5.3.3 LCD-display

LCD-displayen, JM162A, består av två rader där respektive har plats för 16 tecken. På displayen visas sensordata under robotens färd som uppdateras i takt med att nya sensordata anländer. En potentiometer kopplas in för att justera kontrastspänningen och LCD:n konfigureras enligt nedan.

Modell	Bitar	Rader	Punkter	Pekare	Blink	Öka pekare	Skifta text
JM162A	8	2	5x7	Av	Av	På	Av

5.3.4 Styrlägen

Styrmodulen har två olika styrlägen: autonomt eller manuellt läge.

Autonomt läge I autonomt läge agerar styrmodulen på beslut som tas från huvudmodulen. Målet är att navigera modulvis genom den avsökningalgoritmen som är implementerad på huvudmodulen. Styrmodulen begär ett nytt styrkommando genom att skicka en positiv flank på en utgång som är kopplad till ett avbrott hos huvudmodulen. Huvudmodulen tar därefter ett beslut om vilket styrkommando som ska skickas. De möjliga styrkommandona är *Fram*, *Halv fram*, *Halv bak*, *Rotera 90° vänster*, *Rotera 90° höger*, *Rotera 180°* och *Stop*.

När ett styrkommando tas emot av styrmodulen sker regleringen i samband med inkommande sensordata. För att regleringen ska agera snabbt nog behöver sensordata skickas i takten 30 Hz. När sensordata tas emot sparas samtliga värden lokalt i styrmodulen. Därefter kallas den metod som sköter reglering av styrkommandot.

Styrkommandot *Fram* innebär att roboten ska flytta sig en modul föröver. Modullängden är densamma för hela banan och satt till 40 centimeter. Roboten beräknar tillryggalagd sträcka genom att beräkna antalet varv hjulen roterar och under färden sker reglering enligt avsnitt ???. När modulen är avverkad kallas återigen huvudmodulen för ett nytt styrkommando. Tack vare hastigheten hos I²C-bussen och fördelningen hos PWM-styrningen agerar roboten på nästa kommando utan ett ryck emellan.

Styrkommandot *Halv fram* tas emot när roboten befinner sig framför målet och ska lämna förnödenheten. Styrmodulen agerar på samma sätt som vid kommandot *Fram* fast med halverad önskad sträcka. När styrkommandot tas emot sparas främre sensorns värde, för att sedan jämföras med i nästa styrkommando, se nedan.

Styrkommandot *Halv bak* tas alltid emot efter ett kommando av typen *Halv fram*. Båda hjulparen drivs bakåt och roboten jämför det främre sensorvärdet med det som sparades vid *Halv fram*, för att försäkra sig om att den är tillbaka en halv modul.

Styrkommandot *Rotera 90° vänster/höger* innebär en vinkelrät rotation åt respektive håll. Här förutsätts roboten befina sig mitt i en modul och behöver med andra ord inte reglera sig i vertikal- eller horisontellt led. Regleringen av rotation sker med hjälp av det gyro som förmedlar vinkelhastighet till sensormodulen. När en rotation påbörjas summeras vinkelhastigheterna hos styrmodulen i takt med varenda nytt sensordata. Nivån som summationen sker till byter endast tecken mellan vänster- och högerrotation. Det är

rotationsregleringen som sätter kravet att sensordata ska skickas med 30 Hz för att minimera felet vid snabba rotationer.

Styrkommandot *Rotera 180°* innehåller ett halvt varvs rotation av roboten. Regleringen sker på samma sätt som den vinkelräta rotationen. Den enda skillnaden är det tak summationen sker till.

För att hantera det sensorfel som gyrot följer ytterligare en reglering efter båda typerna av rotation. Regleringen bygger på att det finns en vägg att förhålla sig till och kallas därför inte om roboten befinner sig i en fyrvägskorsning. När det finns en vägg på någon sida rätar roboten upp sig genom att minimera differensen mellan den främre och bakre sensorn.

Styrkommandot *Stopp* innehåller att båda hjulparen stannas och ingen reglering sker.

Manuellt läge I manuellt läge styrs roboten via styrkommandon från datormodulen. När styrmodulen tar emot ett styrkommando och befinner sig i detta läge sker körningen helt utan reglering.

Styrmodulen fortsätter med samma styrkommando tills nästa har skickats. Med andra ord krävs det ett stoppkommando för att stanna roboten. Stoppkommandot skickas per automatik när datormodulens piltangent har släppts upp och behöver därför inte tas hänsyn till av användaren.

När ett styrkommando skickas till styrmodulen i manuellt läge tolkas den tredje byten i sändningen som den önskade hastigheten. Hastigheten tas emot som ett heltal i intervallet noll till 100. Det skalas därefter till motsvarande intervall för pulsbreddsmoduleringen.

5.4 Datormodulen

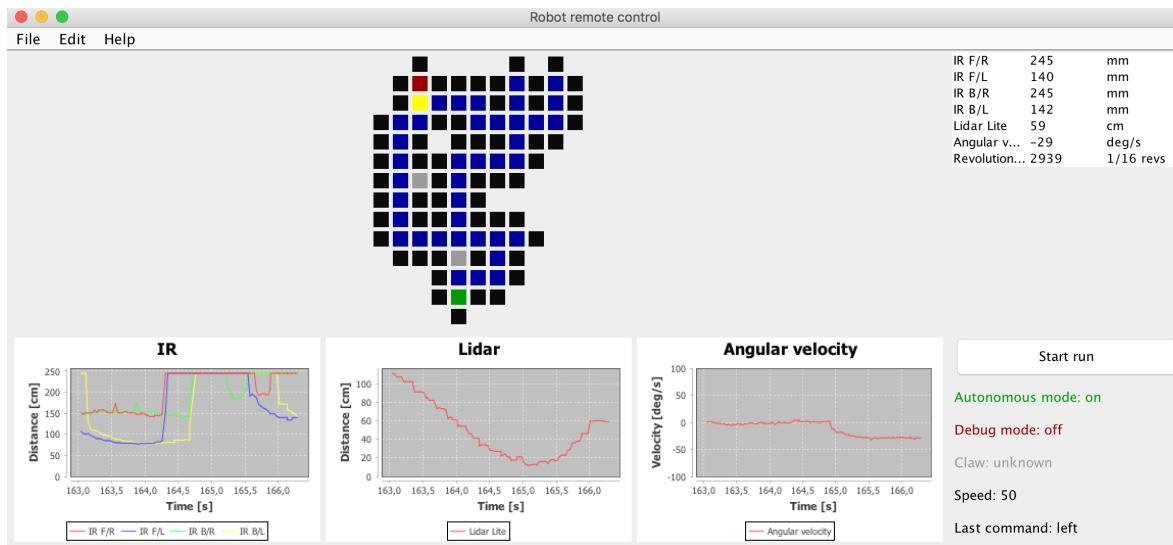
Datormodulen är skriven i Java och följer desingmönstret model-view-controller. Modulen har två huvudsakliga syften. Det ena är att skicka data till roboten, för att kunna styra den manuellt och på ett enkelt sätt ändra olika inställningar hos roboten. Det andra är att ta emot data från roboten för att呈现出 robotens omgivning i form av sensorvärdet och kartdata. Nedan följer en beskrivning av programmets tre delar. Utöver dessa delar finns en del stödklasser, bland annat för hantering av konstanter och text. Användargränssnittet visas i figur 12.

5.4.1 Model

”Model”-delen av programmet består av fyra huvudsakliga klasser: Log, RobotData, SensorData och MapData. Log-klassen används för att spara ner relevanta händelser under testkörningar till txt-filer. RobotData-klassen tar hand om den interna representationen av robotens tillstånd, det vill säga robotens nuvarande körläge, senast skickade körkommando med mera. SensorData-klassen sparar de senast mottagna sensorvärderna från roboten. MapData-klassen sparar alla mottagna kartdata från roboten. De sistnämnda tre data-klasserna används också för att, genom observatörer, ändra den grafiska ”view”-delen av programmet.

5.4.2 View

“View”-delen av programmet sköter programmets grafiska användargränssnitt och består av sex huvudsakliga klasser: Animator, MenuBar, MapPanel, GraphPanel, TablePanel och RobotStatusPanel. Animator-klassen samordnar de olika grafiska elementen och sköter instansieringen av användargränssnittet. MenuBar-klassen hanterar menyn i huvudfönstret. Genom den kan användaren välja körläge, hantera loggar och komma åt hjälpmenyn. MapPanel-klassen sköter den grafiska representationen av kartan. Varje sektion av kartan representeras med ett visuellt element med olika färg beroende på om det är en vägg, start, mål eller en vanlig ruta. Kartan expanderas dynamiskt allteftersom roboten utforskar sin omgivning. GraphPanel-klassen presenterar historiska sensordata i grafer och använder sig av det publika biblioteket JFreeChart 1.0.19. TablePanel-klassen presenterar de senast mottagna sensorvärdena i en tabell. Denna klass låter även användaren ändra vissa reglerkonstanter hos roboten i vissa lägen. RobotStatusPanel-klassen visar robotens nuvarande status, det vill säga autonomt eller manuellt läge, det senast skickade styrkommandot med mera.



Figur 12: Programvarans grafiska användargränssnitt

5.4.3 Controller

“Controller”-delen av programmet består av fyra olika klasser: ActionHandler, Handler, KeyHandler och SerialCommunicationsHandler. ActionHandler-klassen innehåller alla de Actions som används av programmet, det vill säga alla de metoder som anropas bland annat vid knapptryckningar. Handler-klassen sköter samordningen av alla klasser och även programmets uppstart och instansieringen av de olika delarna. KeyHandler-klassen sköter alla tangentbordsanrop. SerialCommunicationsHandler-klassen sköter kommunikationen med roboten. Det innebär uppkoppling, sändning och mottagande av data samt nedkoppling av avslutning av kommunikationen. Detta sker med det publika biblioteket jSSC 2.8.0 tillsammans med datorns egna operativsystem för parning med FireFly-enheten.

6 Intermodulär kommunikation

I denna del förklaras hur kommunikationen mellan robotens olika moduler fungerar.

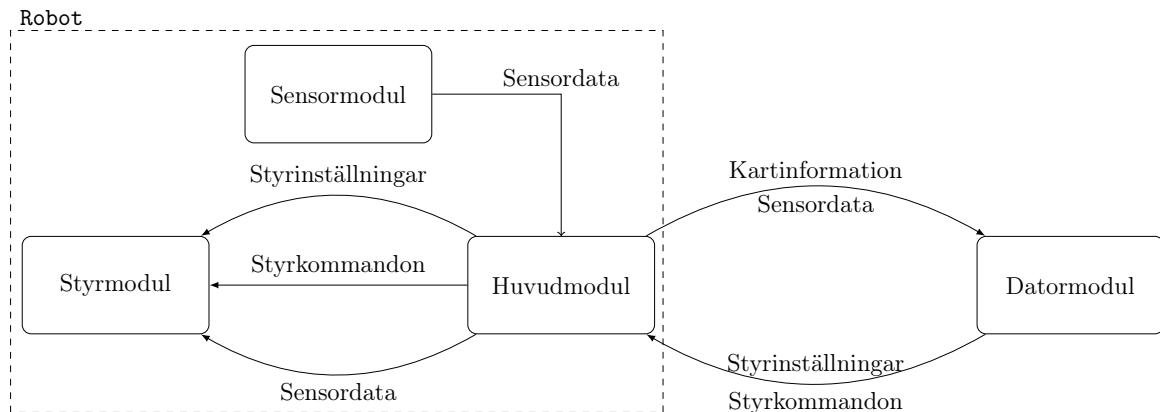
6.1 Informationsflöde

Eftersom flera olika typer av data skickas mellan de olika modulerna inleds kommunikationen med ett ID som betecknar vilken typ av data som skickas. Dessa ID:n har värden mellan 251-255, för att kunna skiljas från alla andra värden som skickas. Vad varje ID innebär visas i tabell 2. Andra värden än kommunikations-ID får alltså *inte* anta värden större än 245. Denna uppdelning har gjorts för att underlätta felsökning.

ID	Datotyp
255	Styrkommando
254	Kartinformation
253	Sensordata
252	Styrinställning
251	Positionsinformation

Tabell 2: Tabell över kommunikations-ID

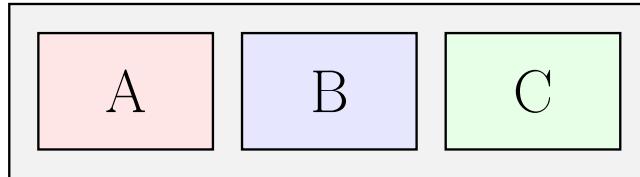
I figur 13 visualiseras informationsflödet. Där anges mottagare och sändare samt vilken typ av information som går mellan dem.



Figur 13: Schema över informationsflödet

6.1.1 Kommunikationsprotokoll - styrkommandon

Protokollet som används för att skicka styrkommandon, både datormodul → huvudmodul och huvudmodul → styrmodul finns illustrerat i figur 14.



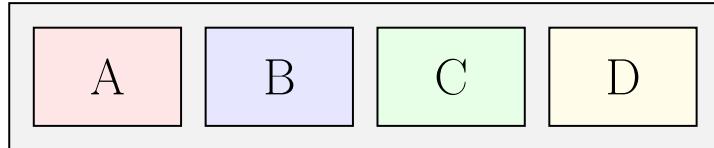
Figur 14: Protokoll för överföring av styrkommandon

Respektive segment motsvarar följande:

- A (1 byte) - Kommunikations-ID, se tabell 2.
- B (1 byte) - Definierar vilket kommando som skickas, se tabell 3.
- C (1 byte) - Hastighet om manuellt läge, valfritt i autonomt då det inte tas hänsyn till (tillåtna värden: 0-245).

6.1.2 Kommunikationsprotokoll - kartinformation

Huvudmodulen skickar för varje ny kartmodul data till datormodulen som ritar upp den nya informationen grafiskt. Varje kartmodul representeras av en position i ett tvådimensionellt fält. För överföring av denna information används protokollet i figur 15.



Figur 15: Protokoll för överföring av kartdata

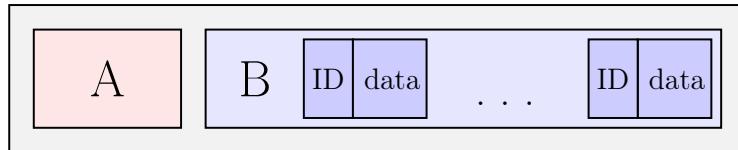
Respektive segment motsvarar följande:

- A (1 byte) - Kommunikations-ID, se tabell 2.
- B (1 byte) - x-koordinat (tillåtna värden: 0-30).
- C (1 byte) - y-koordinat (tillåtna värden: 0-17).
- D (1 byte) - Information om vad som finns i noden.
 - 0-225 - Utforskat
 - 242 - Startposition
 - 243 - Målets position
 - 244 - Vägg

- 245 - Upptäckt men ej utforskad

6.1.3 Kommunikationsprotokoll - sensordata

Sensordata skickas enligt protokollet i figur 16.



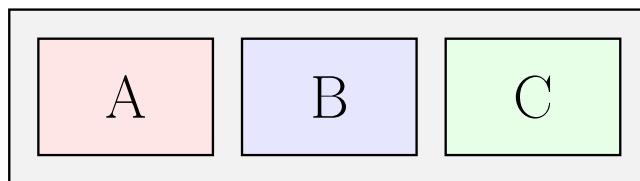
Figur 16: Protokoll för överföring av sensordata

Respektive segment motsvarar följande:

- A - Kommunikations-ID, se tabell 2.
- B - Sensordatapaket
 - ID (1 byte) - identifierar sensorn, se tabell 3.
 - Data (1 byte) - sensorns värde (tillåtna värden: 0-245).

6.1.4 Kommunikationsprotokoll - styrinställningar

Protokollet som används för att skicka styrinställningar, både datormodul → huvudmodul och huvudmodul → styrmodul finns illustrerat i figur 17.



Figur 17: Protokoll för överföring av styrinställning

Respektive segment motsvarar följande:

- A (1 byte) - Kommunikations-ID, se tabell 2.
- B (1 byte) - Definierar vilken styrinställning som skickas, se tabell 3.
- C (1 byte) - 0/1 om av/på, annars flyttal (C float).

Kommando	Id	Info	Kommentar
252	1	0/1	1 om reglering är på, 0 annars
252	2	Data	Reglerfunktionens P-värde
252	3	Data	Reglerfunktionens D-värde
252	9	0/1	1 om gripklo ska vara öppen, 0 om den ska vara stängd
253*	1	Data	Värde på den främre högra IR-sensorn
	2	Data	Värde på den främre vänstra IR-sensorn
	3	Data	Värde på den bakre högra IR-sensorn
	4	Data	Värde på den bakre vänstra IR-sensorn
	5	Data	Värde på Lidar-Lite v2:s 8 mest signifikanta bitar
	6	Data	Värde på Lidar-Lite v2:s 8 minst signifikanta bitar
	7	Data	Värde på gyrot
	8	Data	Värde för IR-detektorn (1 om målet är upptäckt, 0 annars)
	9	Data	Antal flanker reflexsensorn detekterat sen sist
255	0	-	Kommando att stanna
255	1	Data/-**	Kommando att köra framåt
255	2	-	Kommando att rotera 180 grader
255	3	Data/-**	Kommando att rotera höger
255	4	Data/-**	Kommando att rotera vänster
255	12	-	Kommando att köra en halv modul framåt
255	13	-	Kommando att köra en halv modul bakåt

* Gäller för alla efterföljande sensorer då alla värden skickas i samma paket

** Reagerar på data vid manuellt läge och ignoreras data vid autonomt

Tabell 3: Tabell över informationsflödet

7 Slutsatser

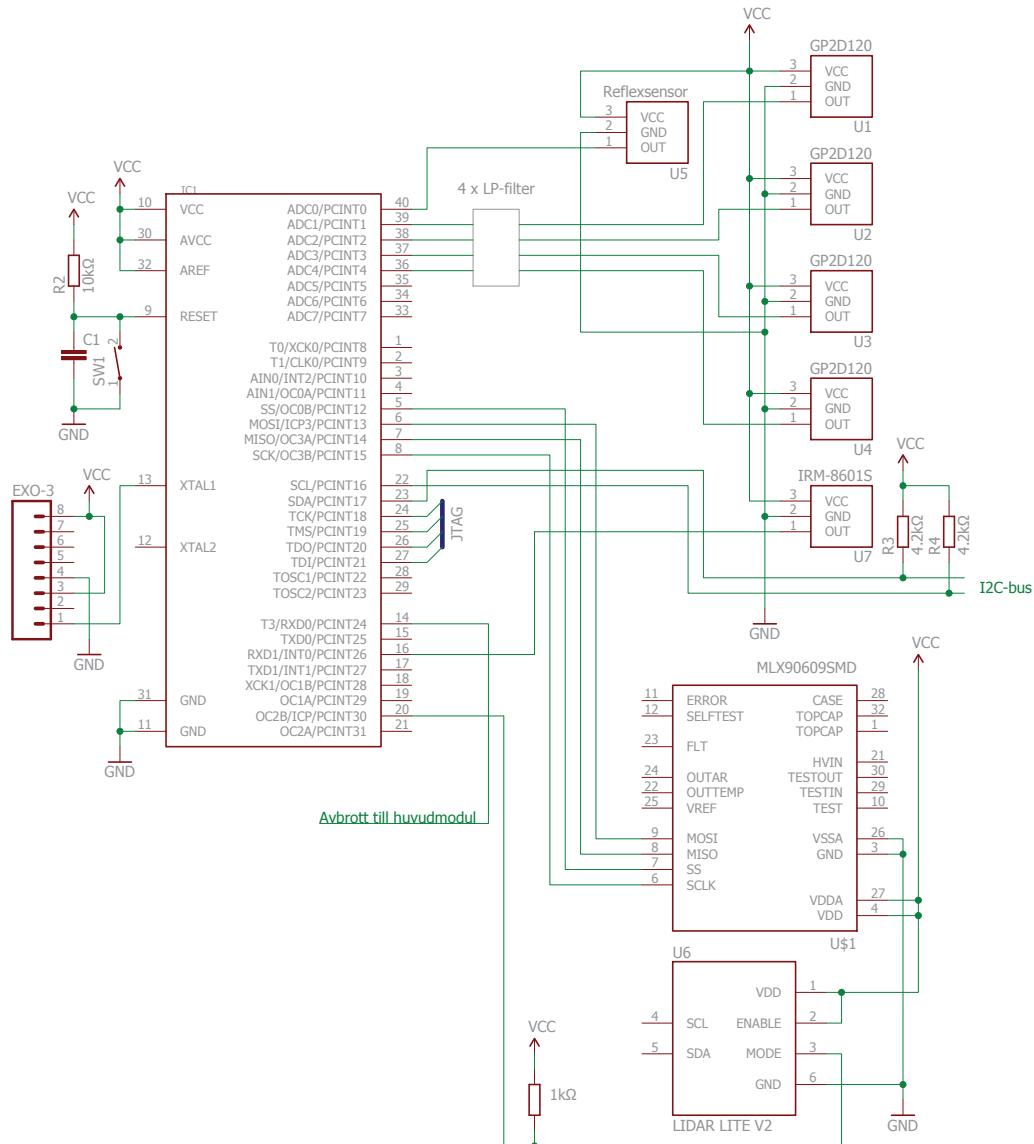
Gruppen känner sig nöjd med uppnått resultat. Produkten uppfyller samtliga förstahandskrav, ett andrahandskrav och färdigställdes i enlighet med uppställd tidsplan. Det finns samtidigt utvecklingmöjligheter av produkter inom några områden.

En av de naturligt följande förbättringarna är att få roboten att kunna navigera i öppna rum, vilket var ett krav av prioritet 2. Nu är roboten begränsad till att operera i en omgivning bestående av ett labyrintsystem med vinkelräta svängar. För att utöka dess användningsområde kan även produkten utvecklas så att den kan reglera vid alla typer av svängar, inte bara vinkelräta.

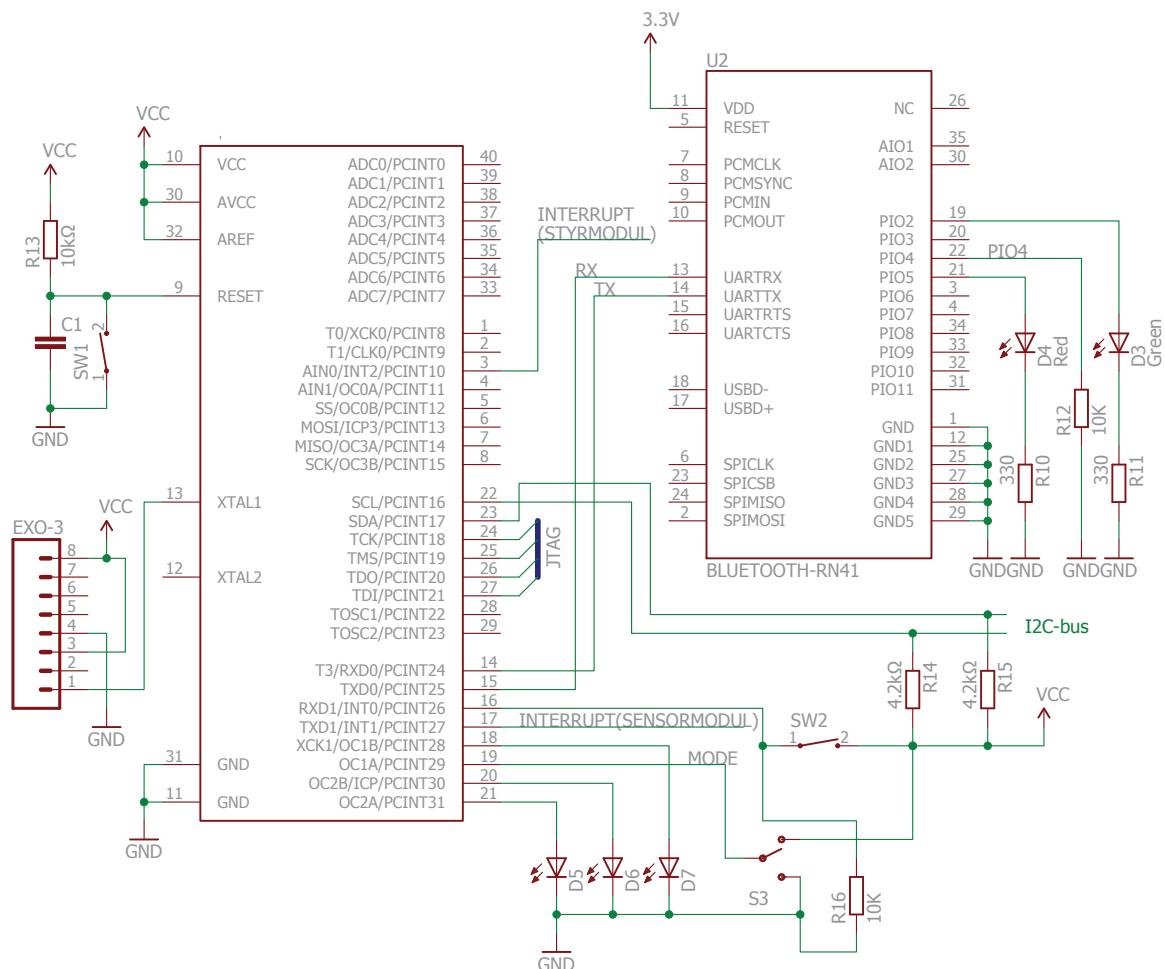
Något som gruppen hade velat utveckla i mån av mer tid är kartläggning i tre dimensioner. Det hade bidragit till en bättre användarupplevelse. I övrigt är gruppen nöjd med robotens funktioner.

Förbättringspotential finns även i de redan existerande funktionerna. Roboten skulle till exempel kunna göras smartare genom att förbättra avsökningsalgoritmen som i dagsläget inte är helt optimerad. Det skulle göra stor skillnad i en stor och komplifierad bana.

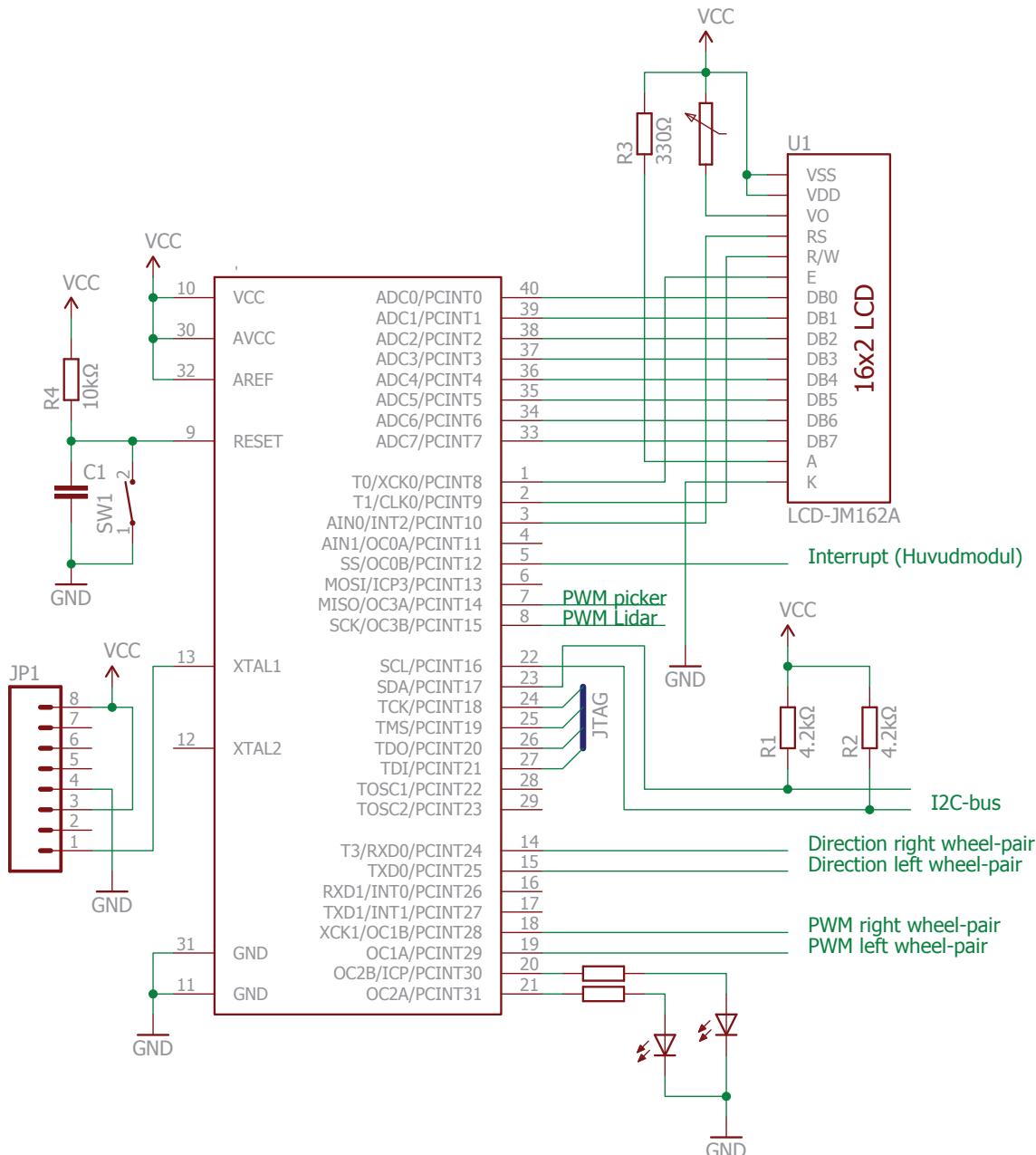
O Kopplingsschema



Figur 18: Kopplingsschema över sensormodul



Figur 19: Kopplingsschema över huvudmodul



Figur 20: Kopplingsschema över styrmodul

P Kodexempel

Nedan följer kodexempel från de olika modulerna.

P.1 Huvudmodul

```

1  ****
2  /*  findTarget - Sends a controller command according to search
3   algorithm
4
5   walls[i] = 0 open
6   1 wall
7   */
8  ****
9  uint8_t * findTarget()
10 {
11     if(walls[0] == 0){ //Rotate 90 degrees to the right
12         return rotateRight;
13     } else if(walls[1] == 0){ //One forward
14         return oneForward;
15     } else if (walls[2] == 0){ //Rotate 90 degrees to the left
16         return rotateLeft;
17     } else { //Rotate 180 degrees (clockwise)
18         return rotate180;
19     }
20 }
21 ****
22 /* hasFoundTarget - checks if target has been found
23
24 Returns: 0 not found
25         1 target in the module ahead
26         2 target found
27
28 */
29 ****
30 uint8_t hasFoundTarget(void)
31 {
32     straightAhead = sensorData[10]*128 + sensorData[12];
33
34     if((sensorData[16] == 1) && (straightAhead < oneModuleAhead)){
35         return 1;
36     } else if((target[0] != 0xFF) && (target[1] != 0xFF)){
37         return 2;
38     } else {
39         return 0;
40     }
41 }
42 ****
43 /* readSensors - checks in which directions there are walls
44

```

```

45
46     Updates map and walls accordingly
47     */
48 /***** ****
49 void readSensors()
{
50
51     walls[0] = 0;
52     walls[1] = 0;
53     walls[2] = 0;
54
55     straightAhead = sensorData[10]*128 + sensorData[12];
56
57
58 //Cardinal direction: north, east, south, west
59 if(direction == 0){
60     //Straight ahead, right, !!south open!!, left
61     if (straightAhead < oneModuleAhead){
62         map[position[0]][position[1]+1] = wallValue;
63         walls[1] = 1;
64     }
65
66     if (sensorData[2] < 245){
67         map[position[0]+1][position[1]] = wallValue;
68         walls[0] = 1;
69     }
70
71     if (sensorData[4] < 245){
72         map[position[0]-1][position[1]] = wallValue;
73         walls[2] = 1;
74     }
75
76 } else if(direction == 1) {
77     //Left, straight ahead, right, !!west open!!
78     if (sensorData[4] < 245){
79         map[position[0]][position[1]+1] = wallValue;
80         walls[2] = 1;
81
82         if (straightAhead < oneModuleAhead){
83             map[position[0]+1][position[1]] = wallValue;
84             walls[1] = 1;
85         }
86
87         if (sensorData[2] < 245){
88             map[position[0]][position[1]-1] = wallValue;
89             walls[0] = 1;
90         }
91     }
92
93 } else if(direction == 2) {
94     //!!north open!!, left, straight ahead, right
95     if (sensorData[4] < 245){
96         map[position[0]+1][position[1]] = wallValue;

```

```

97     walls[2] = 1;
98 }
99
100 if (straightAhead < oneModuleAhead){
101     map[position[0]][position[1]-1] = wallValue;
102     walls[1] = 1;
103 }
104
105 if (sensorData[2] < 245){
106     map[position[0]-1][position[1]] = wallValue;
107     walls[0] = 1;
108 } else if (countNrOfExploredPaths() <= 1){
109     nrOfUnexploredPaths = nrOfUnexploredPaths + 1;
110 }
111
112 } else if (direction == 3){
113     //Right, !east open!!, left, straight ahead
114     if (sensorData[2] < 245){
115         map[position[0]][position[1]+1] = wallValue;
116         walls[0] = 1;
117     }
118
119     if (sensorData[4] < 245){
120         map[position[0]][position[1]-1] = wallValue;
121         walls[2] = 1;
122     }
123
124     if (straightAhead < oneModuleAhead){
125         map[position[0]-1][position[1]] = wallValue;
126         walls[1] = 1;
127     }
128 }
129
130 //Send map
131 if (position[0] + 1 < dimx){
132     sendMapCoordinate(position[0] + 1, position[1]);
133 }
134 if (position[0] - 1 >= 0){
135     sendMapCoordinate(position[0] - 1, position[1]);
136 }
137 if (position[1] + 1 < dimy){
138     sendMapCoordinate(position[0], position[1] + 1);
139 }
140 if (position[0] - 1 >= 0){
141     sendMapCoordinate(position[0], position[1] - 1);
142 }
143
144 }

```

P.2 Sensormodul

```

1 //*****
2 /*  Interrupt: Infrared detector - detection of target.
3    Checks if target is found. Responds to any edge.
4    */
5 //*****
6 ISR (INT0_vect)
7 {
8     if(lookingForTarget == 1) {
9         if(!(EICRA & (1<<ISC00))) //If falling edge
10        {
11            counter = 0;
12            TCCROB |= (0<<CS02)|(0<<CS01)|(1<<CS00);
13            EICRA |= (1<<ISC00);
14
15        } else if(EICRA & (1<<ISC00)){
16            TCCROB &= ~(1<<CS00);
17            TCCROB &= ~(1<<CS01);
18            TCCROB &= ~(1<<CS02);
19            EICRA &= ~(1<<ISC00);
20
21            if(counter>20){
22                anyEdge = 1;
23                detection++;
24            }
25        }
26    }
27 }
28
29 //*****
30 /*  Interrupt: ADC conversion - Invokes interrupt once conversion is
31   finished
32   Transforms an analog input from the IR-sensors into digital output.
33   */
34 //*****
35 ISR(ADC_vect){ //IR-SENSOR
36     CLKPR = 0x06;
37
38     if(ADMUX == adc1){
39         Distance_1 = ADCH;
40         ADMUX = adc2;
41         //PORTB ^= (1<<PORTB0);
42     }
43     else if(ADMUX == adc2){
44         Distance_2 = ADCH;
45         ADMUX = adc3;
46     }
47     else if(ADMUX == adc3){
48         Distance_3 = ADCH;
49         ADMUX = adc4;
50     }

```

```

51     else if(ADMUX == adc4){
52         Distance_4 = ADCH;
53         ADMUX = adc5;
54     }
55     else{
56         if ((ADCH >= 160) && (lastWheelSensor == 0)){
57             wheel_counter = wheel_counter + 1;
58             lastWheelSensor = 245;
59         }
60         else if ((ADCH <= 60) && (lastWheelSensor == 245)){
61             wheel_counter = wheel_counter + 1;
62             lastWheelSensor = 0;
63         }
64         wheel_sensor = ADCH;
65         ADMUX = adc1;
66     }
67 }
68 */
69 /* Subprogram: Get median value - picks the median value of five.
70 It is given an array of values from LIDAR-Lite, returns the
71 median.
72 */
73 */
74 /*
75 uint16_t getMedianLIDAR(uint16_t arr[])
76 {
77     uint16_t i, j, swap;
78
79     for (i=0; i<4; i++)
80     {
81         for (j=0; j<4-i; j++)
82         {
83             if ( arr[j] > arr[j+1])
84             {
85                 swap = arr[j+1];
86                 arr[j+1] = arr[j];
87                 arr[j] = swap;
88             }
89         }
90     }
91     return arr[2];
92 }
```

P.3 Styrmodul

```

1 /*
2 /* autonomousRotate - autonomously rotate.
3
4 Rotate autonomously with respect to preferred accumulated angle
5 (preferredAccumulatedAngle) and call for new control command
```

```

6    when rotation is considered done.
7
8    Before autonomousRotate is called, preferredAccumulatedAngle should
9    be updated using the convertAngle function. convertAngle sets the
10   number of degrees and direction to rotate.
11
12  Note: labs = long abs
13          */
14 /*****
15 void autonomousRotate()
16 {
17     if(labs(accumulatedAngle) > labs(preferredAccumulatedAngle)){
18         if(adjustRotationMode == 1) {
19             currControlCommand = commandAdjust;
20         } else {
21             currControlCommand = stop;
22             stopWheels();
23             if (debugMode == 0){
24                 callMainInterrupt();
25             }
26         }
27     } else if (preferredAccumulatedAngle > 0){
28         rightWheelPair(preferredRotationSpeed, 1);
29         leftWheelPair(preferredRotationSpeed, 0);
30     } else {
31         rightWheelPair(preferredRotationSpeed, 0);
32         leftWheelPair(preferredRotationSpeed, 1);
33     }
34
35     accumulatedAngle += angularVelocity;
36 }
37
38 /*****
39 /*  autonomousAdjust - autonomously adjust angle to wall.
40
41 Autonomous adjust the angle to the wall or compensate the
42 preferred accumulated angle (preferredAccumulatedAngle)
43
44 Called either before or after rotation, depending on adjust
45 rotation mode (adjustRotationMode).
46
47 Before autonomousAdjust is called, autonomousAdjustMode should
48 be updated.
49          */
50 /*****
51 void autonomousAdjust()
52 {
53
54     if ((sideSensors[frontLeftIndex] != maxDistance) &&
55         (sideSensors[rearLeftIndex] != maxDistance)){
56
57         lastDistanceDifference = sideSensors[rearLeftIndex] -

```

```

58             sideSensors[frontLeftIndex];
59 } else if ((sideSensors[frontRightIndex] != maxDistance) &&
60             (sideSensors[rearRightIndex] != maxDistance)){
61     lastDistanceDifference = sideSensors[frontRightIndex] -
62                             sideSensors[rearRightIndex];
63 }
64
65 if(labs(lastDistanceDifference) <= preferredDistanceDifference) {
66     if(adjustRotationMode == 2) {
67         adjustRotationMode = 1;
68         currControlCommand = rotation;
69     } else {
70         currControlCommand = stop;
71         stopWheels();
72         if (debugMode == 0){
73             callMainInterrupt();
74         }
75     }
76 } else {
77     if (lastDistanceDifference < 0){
78         rightWheelPair(preferredAdjustSpeed, 1);
79         leftWheelPair(preferredAdjustSpeed, 0);
80     } else {
81         rightWheelPair(preferredAdjustSpeed, 0);
82         leftWheelPair(preferredAdjustSpeed, 1);
83     }
84 }
85 }
```

P.4 Datormodul

```

1 /**
2  * Creates a new log with a timestamp
3  *
4  * @author Isak Stroemberg
5  */
6 public void createNewLog() {
7     logEmpty = true;
8
9     // get time and date
10    DateFormat fileDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm
11 .ss");
12    DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
13
14    Date date = new Date();
15
16    String fileTimeDate = fileDateFormat.format(date); //2014/08/06
17      15:59:48
18    String timeDate = dateFormat.format(date); //2014/08/06 15:59:48
```

```
17 // create file if needed
18 lastLog = new File(filePath + fileName + fileTimeDate +
19   fileExtension);
20
21 if(!lastLog.exists()) {
22   try {
23     lastLog.createNewFile();
24
25     // write log header
26     FileWriter fileWriter = new FileWriter(lastLog.getAbsoluteFile(
27       (), true);
28
29     PrintWriter printWriter = new PrintWriter(fileWriter);
30
31     printWriter.println("=====");
32     printWriter.println(timeDate);
33     printWriter.println("=====\n");
34
35     printWriter.close();
36   } catch (IOException e) {
37     e.printStackTrace();
38   }
39 }
```

```
1  /**
2   * Sensor data observer
3   *
4   * @author Isak Stroemberg
5   */
6  @Override
7  public void update(Observable o, Object arg) {
8      // check if update is in sensorData
9      if(o instanceof model.SensorData) {
10          if(arg instanceof int[]) {
11              int[] sensorData = (int[]) arg;
12
13              int index = 0;
14              for(int value : sensorData) {
15                  sensorTable.setValueAt(value, index, 1);
16                  index++;
17              }
18          }
19      }
}
```

```
1  /**
2   * Update sensor values and notify
3   *
4   * @author Isak Stroemberg
5   */
6  public void update(int[] sensorValues) {
```

```
7     int accumulatedRotation = 0;
8     if(values != null) {
9         accumulatedRotation = values[values.length - 1];
10    }
11
12    values = sensorValues;
13
14    // take lidar average
15    for(int index = 0; index < oldLidarValues.length - 1; index++) {
16        oldLidarValues[index + 1] = oldLidarValues[index];
17    }
18    oldLidarValues[0] = values[4];
19
20    int lidarAverage = 0;
21    for(int value : oldLidarValues) {
22        lidarAverage += value;
23    }
24    lidarAverage /= oldLidarValues.length;
25
26    values[4] = lidarAverage;
27
28    values[values.length - 1] += accumulatedRotation;
29
30    setChanged();
31    notifyObservers(values);
32}
33}
```
