Designspecifikation

Victor Tranell

Version 1.0

Status

Granskad		
Godkänd		

M/S Sea++

Projektgrupp 2 , HT15
Linköpings tekniska högskola, ISY

Namn	Ansvar	Telefon	E-post
Anton Rooth		070 369 01 40	antro937@student.liu.se
Erik Rönmark		076 818 78 26	eriro331@student.liu.se
Michael Sörsäter	Dokumentansvarig (DOK)	076 142 70 99	micso554@student.liu.se
Mikael Ångman		073 843 15 00	mikan972@student.liu.se
Peter Tullstedt		073 714 45 66	pettu298@student.liu.se
Victor Tranell	Projektledare (PL)	073 680 71 09	victr593@student.liu.se

E-postlista för hela gruppen: tsea29.grupp2@gmail.com
Hemsida: https://github.com/nullacid/grupp2robot

Kund: Instutitionen för systemteknik, Linköpings Universitet, 581 83 LINKÖPING,
telefon 013-28 10 00, fax: 013-13 92 82
Kontaktperson hos kund: Tomas Svensson, 013-28 13 68, tomas.svensson@liu.se

Kursansvarig: Tomas Svensson, B-huset, rum 3B:528, 013-28 13 68, tomas@isy.liu.se
Handledare: Peter Johansson, B-Huset, rum 3D: 541, 013-28 13 45, peter.a.johansson@liu.se

Innehåll

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2015-10-02	Första versionen	Victor Tranell	
0.2	2015-10-23	Andra versionen	Victor Tranell	
1.0	2015-10-29	Första versionen	Victor Tranell	

1. Inledning

Denna designspecifikation har till uppgift att noggrant delge hur kartroboten M/S Sea++ ska konstrueras samt hur kontrollmjukvaran ska implementeras. Roboten ska helt autonomt kunna kartlägga ett rum och kontinuerligt skicka tillbaka sensor- och kartdata till en extern dator vars programvara ska visa datan grafiskt. Roboten ska även ha ett läge där den kan styras från den externa datorn likt en radiostyrd bil.

2. Översikt av systemet

Systemet är uppbyggt av tre moduler som är monterade på ett chassi. Utöver det används en extern dator.

- Den externa datorn ska med ett grafiskt gränssnitt visa sensordata samt realtidsuppritning av kartan. Den externa datorn ska kommunicera med kommunikationsmodulen via Bluetooth.
- Styrmodulen ansvarar för all styrlogik, tar emot kommandon och data från övriga moduler och styr hjulen utifrån detta.
- Sensormodulen samlar in sensordata som bearbetas för att få värden som de andra modulerna kan hämta.

2.1. Gränssnitt

Kommunikationen i systemet kommer att ske seriellt mellan alla moduler. Denna seriella kommunikation kommer att följa ett protokoll som visas i Bild 1. Receivern är den modul som i just den situationen tar emot ett kommando. Receivermodulen kommer att i sin vanliga algoritm att se om det finns en byte att hämta i dess buffer.

Receiverns algoritm börjar med att den tar emot byten från sin buffer och samtidigt meddelar sendern att den är redo att ta emot mer. Om det finns fler bytes att ta emot så väntar receivern på att det finns ny data att hämta, och hämtar datan när den finns. Sedan meddelar den sendern att den är redo att ta emot mer data, ifall det finns fler bytes att hämta så upprepas processen. Sendern börjar med att skicka en byte till receiverns buffer och väntar därefter på att receivern är redo att ta emot mer. Om det inte finns fler bytes att skicka kan sendern här hoppa ut ur senderalgoritmen, annars så skickar den nästa byte och repeterar.

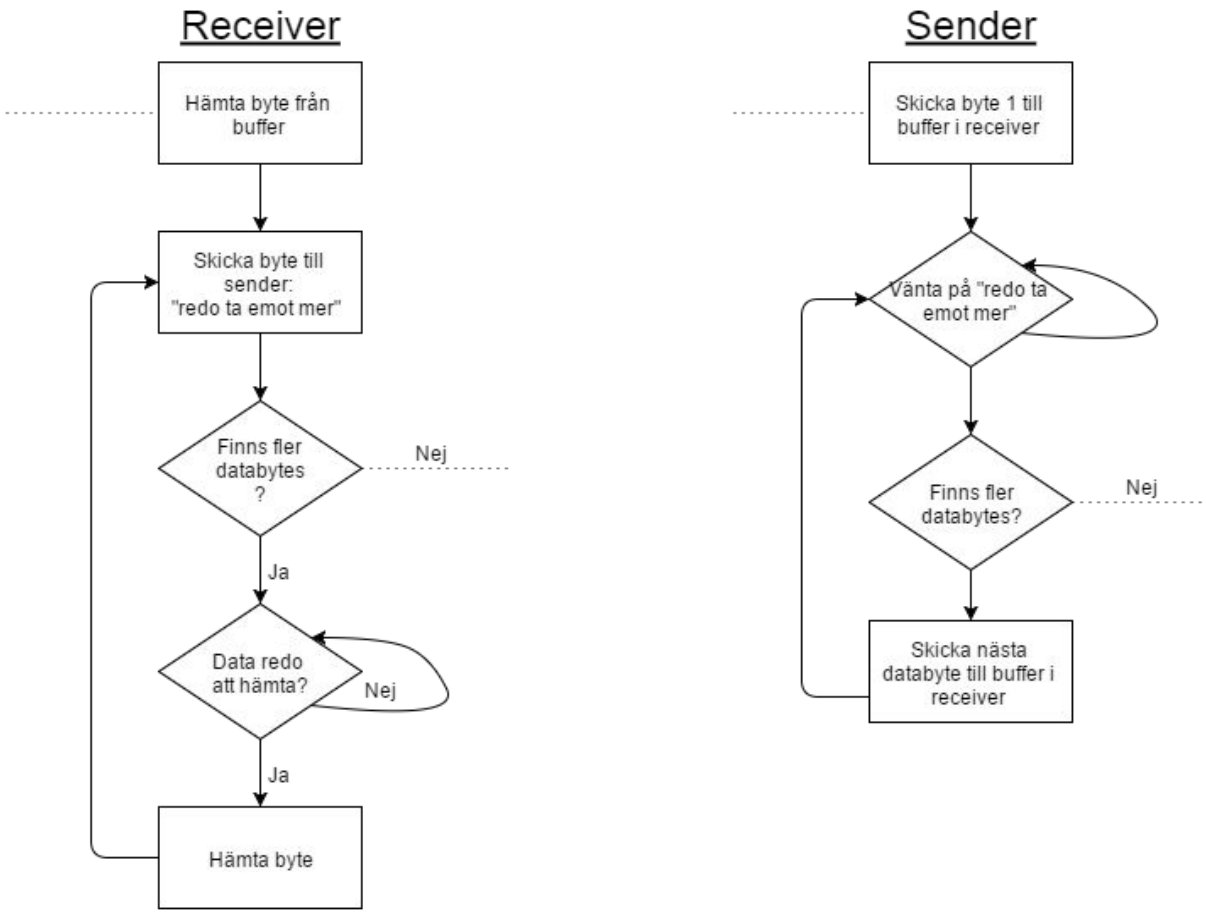

Bild 1. Flödesschema över sender-receiver förhållandet i systemets kommunikationsgränssnitt.

Kommunikationen mellan den externa datorn och kommunikationsmodulen sker via Bluetooth. Mellan kommunikationsmodulen och styrmodulen samt mellan styrmodulen och sensormodulen sker kommunikationen med USART. Kommunikationen kommer fungera så att när kommunikation sker nedåt i systemet så skickas ett funktionskommando, se Bild 2. När kommunikationen sker uppåt i systemet så skickas databytes med bl.a. sensordata, vilket enbart kommer att ske på förfrågan uppifrån. I funktionskommandot så bestämmer de 2 mest signifikanta bitarna längden i bytes på dataströngen som ska skickas tillbaka. De kommer att vara 0 när ingen data ska hämtas (styrkommandon).

Bild 2. Översikt av bitarnas betydelse i ett funktionskommando.

2.1.1. Felhantering

Bit 5 i funktionskommandot kommer att vara en paritetsbit för att kontrollera kommandots intergritet. Sändaren kommer att summera ettorna i kommandot och sätta paritetsbiten till 1 eller 0 om summan blir udda respektive jämn.

Om paritetsbiten inte stämmer så skickar mottagaren tillbaka ett felmeddelande “1111 1111” till sändaren, som då avbryter sitt väntande på databytes och upprepar sitt kommando.

Databytesen kommer inte att ha någon paritetsbit eftersom datan som hämtas ändå kommer att hämtas så pass ofta. Däremot så kommer databytesen för kartdatat att ha en paritetsbit i den mest signifikatna biten i varje byte eftersom datan bara kommer att skickas en gång per ruta. Felaktig data skulle då ge ett felaktigt resultat i den grafiska representationen som inte kommer att korrigeras under körningen. Felaktig paritet i en byte gör att mottagaren begär datan igen med hjälp av begär temp-funktionen, för att få den senast skickade kartdatan.

2.1.2. Kommandon

I Bild 3 visas hur ett kommando skickas genom systemet och vilken information som skickas i vilket steg. Funktionskommandon skickas bara nedåt i systemet och datakommandon skickas bara uppåt i systemet. När en modul har skickat ned ett kommando för att hämta data så kommer den att vänta på att datan returneras innan den kan fortsätta med sina andra sysslor.

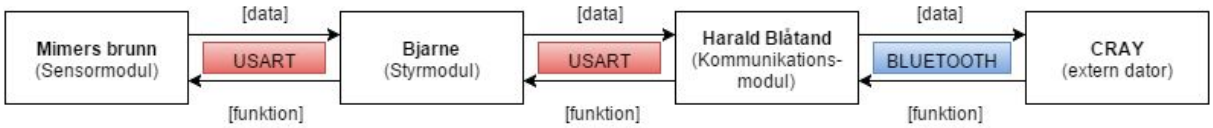

Bild 3. Hämtning av sensordata.

När data ska hämtas kommer sendern som frågade nedåt att vänta på att få tillbaka data nedifrån. Eftersom den har kvar funktionskommandot som den skickade nedåt vet den även hur många bytes den ska vänta på innan all data har hämtats. Den modul som är ansvarig för originalförfrågan kommer av samma anledning även att veta vad datan som tas emot hör till. I tabellerna nedan listas de olika funktionskommandona som används.

Tabell 1 - meddelandedata för styrningen

Tangentbordshändelse	**OP-kod (hexadecimal)**
Pil upp trycks ner	00
Pil upp släpps	01
Pil vänster trycks ner	02
Pil vänster släpps	03
Pil ner trycks ner	04
Pil ner släpps	05
Pil höger trycks ner	06

Pil höger släpps	07

Tabell 2 - meddelandedata för hämtning av data.

Datatyp att hämta	**Beskrivning**	**Data (hex), svarets storlek (antal bytes)**	**OP**
Sensordata LIDAR	Avstånd i cm till vägg framåt	88, 2	08
Sensordata IR höger fram	Avstånd i cm till vägg på höger sida (främre sensor)	49, 1	09
Sensordata IR höger bak	Avstånd i cm till vägg på höger sida (bakre sensor)	4A, 1	0A
Sensordata IR vänster fram	Avstånd i cm till vägg på vänster sida (främre sensor)	4B, 1	0B
Sensordata IR vänster bak	Avstånd i cm till vägg på vänster sida (bakre sensor)	4C, 1	0C
Sensordata Gyro	Vinkelhastighet	8D, 2	0D
Sensordata Reflexsensor	Svart eller Vit markering	4E, 1	0E
Tokenvärde LIDAR	Antalet tomma rutor innan vägg	4F, 1	0F
Tokenvärde parallell höger	Kvot mellan de högra IR-sensorerna	50, 1	10
Tokenvärde parallell vänster	Kvot mellan de vänstra IR-sensorerna	51, 1	11
Tokenvärde Gyro	Antalet grader, mod 90°	52, 1	12
Tokenvärde vägg höger fram	Vägg eller inte vägg, vägg mer än 40 cm bort	53, 1	13
Tokenvärde vägg höger bak	Vägg eller inte vägg, vägg mer än 40 cm bort	54, 1	14
Tokenvärde vägg vänster fram	Vägg eller inte vägg, vägg mer än 40 cm bort	55, 1	15
Tokenvärde vägg vänster bak	Vägg eller inte vägg, vägg mer än 40 cm bort	56, 1	16
Tokenvärde Reflexsensor	Totalt tillryggalagd sträcka i cm	97, 2	17
Förändringskö karta dequeue	x, y och info om tilen. Se Bild 11	98, 2	18

Senaste styrdata	Senaste styrbeslut	59, 1	19
Systemets position i kartan	x och y	9A, 2	1A
Position i algoritm	Vad roboten just nu utför (följa vägg, utforska unexplored, hitta hem, o.s.v)	5B, 1	1B
Begär tempKartdata	Begär den senast begärda tilen på nytt.	9C, 2	1C

2.1.3. SPI

SPI kommer användas i sensormodulen, mellan sensorenheten(master) och gyrot(slave).

SPI står för Serial Peripheral Interface. Det finns en masterenhet och en slavenhet och de båda enheterna innehåller varsitt skiftregister. I SPI används dessa olika signaler:

- SS, slave select, används för att välja vilken slavenhet som ska användas
- MOSI, Master Output Slave Input, Datalinje där masterenheten skickar information till slavenheten
- MISO, Master Input Slave Output, Datalinje där slavenheten matar masterenheten med information
- SCLK, Klocka för synkronisering

I vår implementation behöver vi aldrig ha mer en en slavenhet per masterenhet, vår implementation kommer se ut som bild 4

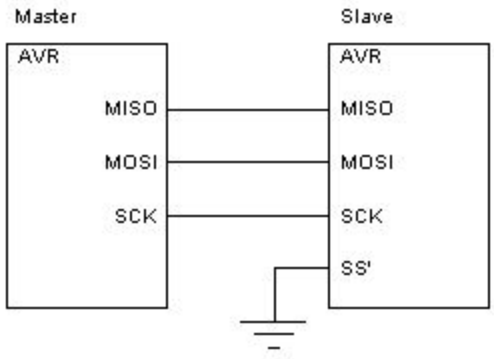

Bild 4. Vår implementation av SPI, 1 till 1 ratio

2.1.4. Bluetooth

Bluetooth kommer att användas mellan kommunikationsmodulen och den externa datorn. Läs mer om implementationen i kapitel 4.

2.1.5. TWI (I^2C)

ATmega16-kortet har stöd för I^2C-protokollet, här kallat TWI (Two wired interface). TWI ska användas för hämta information från LIDAR, se bild 5:

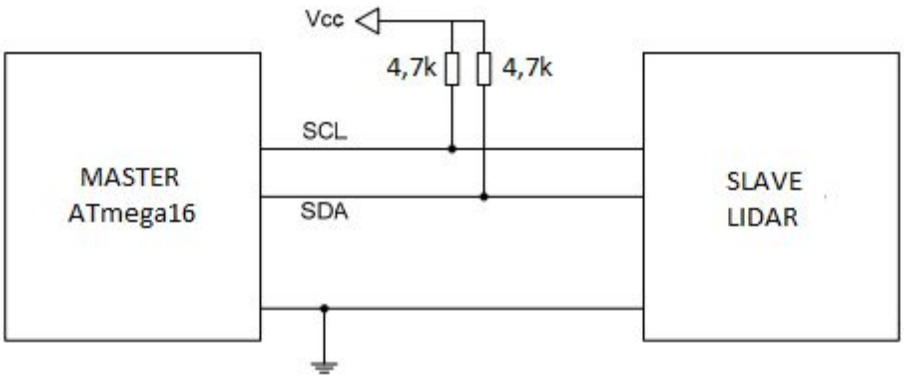

Bild 5. Vår implementation av TWI.

För att SCL och SDA ska vara inaktivt låga, används två pullupresistorer på 4.7 kΩ. SCL är klockan och SDA är datalinjen. För att få ut information från slavenoden skickar masternoden en startsignal, ett adresspaket (7 bitar, adressen till slavnoden) och 1 bit för att bestämma om det är läsa eller skriva den ska göra. I vårt fall kommer masternoden endast behöva läsa data från slavenoden. När masternoden läst av data skickas en slutbit.

2.1.6. USART

ATmega1284 har inbyggt stöd för USART i form av två register och följande pinnar:

- RxD: Reciever
- TxD Transmitter
- XCK Klocka

Det finns tre lägen som ATmega-kortet kan köra i:

- Asynkront normalt
- Asynkront dubbel hastighet
- Synkront

I vår fall kommer vi köra i det synkrona läget och vi har därför en master-slave relation där masterenheten styr klockan (XCK).

2.1.7. Implementationsstrategi

Vi har huvudsakligen tänkt bygga systemet modulvis utifrån modellen “utifrån-och-in” där vi kommer montera sensorer och kort och sedan testa småfunktionaliteter. Styrmodulen kommer däremot byggas “inifrån-och-ut” genom att börja med att abstrahera styrlogiken för att sedan lägga på mer funktionalitet.

De olika moduelerna kommer testas kontinuerligt samtidigt som mer och mer funktionalitet byggs på. Hela systemet kommer testas när varje modul är fungerande var för sig.

3. PC - CRAY™ Super Computer

CRAY uppfyller tre funktioner:

- Hämta data.
- Presentera data.
- Skicka styrkommandon.

Det externa programmet kommer att skrivas i Python 3.1 på grund av att alla de externa biblioteken som används finns till Python 3.1. All kommunikation med systemet kommer att ske genom Bluetooth. Bluetoothkommunikationen ska implementeras med hjälp av pythonbiblioteket pybluez som redan har implementerade funktioner för Bluetoothhantering i python. För att rita upp GUIt och hantera knapptryck så ska pythonbiblioteket pygame användas.

3.1. Bluetooth → Kommunikationsmodul

CRAY ska kommunicera med M/S Sea++ genom Bluetooth. Pybluez innehåller funktioner för det mesta som behövs för Bluetoothkommunikation, så utöver det så behöver bara gränssnittet som beskrivits i 2.1 implementeras.

3.2. Implementation av funktioner

Med pygame är det enkelt att kontrollera nedtryckningar och uppsläppningar av knappar, vilket passar bra för att styra systemet. Nedtryckning av en knapp ska sätta igång en motor och sedan när knappen släpps upp stängs motorn av. Det innebär att det måste finnas funktioner för både nedtryckning och uppsläppning av de tangenter som ska användas (piltangenter). Dessa funktioner ska skicka vidare detta kommando via bluetooth enligt Tabell 1.

Hämtning av data kommer att ske under en loop som hämtar in data med hjälp av de kommandon som finns i Tabell 2. När datan sedan kommer tillbaka så sätter CRAY datan till rätt variabel så att den kan ritas ut på skärmen.

3.3. GUI

CRAYs GUI ska presentera sensordata, styrdata och kartan från M/S Sea++. Grafiken ska implementeras med hjälp av biblioteket pygame, som gör det väldigt enkelt att implementera enkel 2D-grafik. Ett exempel på hur GUIt kan komma att se ut finns i Bild 6, där kartan tar upp vänstra halvan av skärmen och sensordatan är uppskriven på högersidan.

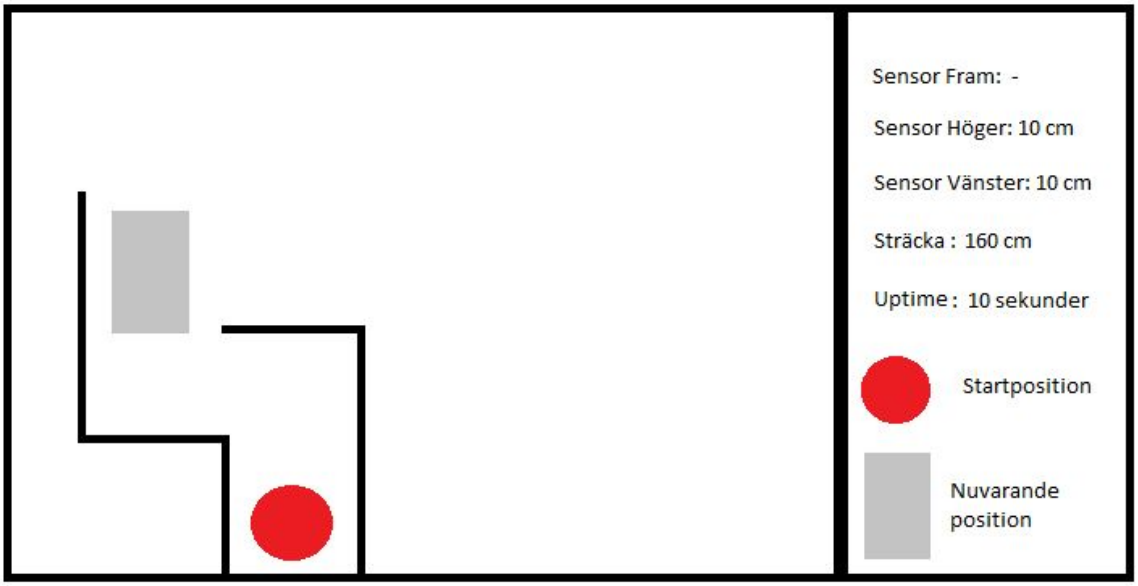

Bild 6. Exempel på ett GUI för CRAY.

3.3.1. Karta

Kartan i systemet kommer att representeras av en 32 x 32 array med värden för om varje ruta är outforskad, öppen eller en vägg. Den datan kommer att hämtas kontinuerligt av CRAY och kan enkelt representeras grafiskt genom att den har x-värde, y-värde och rutans tillstånd. Själva kartan kan egentligen representeras med en 16 x 16 array, men eftersom robotens startposition i kartan är okänd så behövs en extra stor karta för att underlätta uppritning/positionering. När hela ytterkanten är funnen så skulle den grafiska representationen av kartan eventuellt kunna skalas om till denna 16 x 16 array istället för att visa mycket tomrum runt kartan. Eftersom systemets position i kartan finns lagrat i systemet så är det enkelt att rita ut den på den grafiska representationen av kartan också.

4. Kommunikationsmodul - Harald Blåtand

Harald Blåtand ska hantera kommunikationen mellan systemet och den externa datorn. Modulen kommer ha en dator av typen ATmega1284. Modulen kommer i aktivt läge att vänta på ett kommando från den externa datorn. När den tar emot ett kommando skickar den vidare det via USART till styrmodulen. Om kommandot var att hämta data väntar den på en respons och skickar sedan vidare datan upp till den externa datorn. Efter det lyssnar den igen efter ett kommando att ta emot från den externa datorn. Om datan som den ska få tillbaka av styrmodulen innehåller fler bytes väntar kommunikationsmodulen tills den fått alla bytes innan den skickar vidare datan till den externa datorn. En tydligare bild av kommunikationsmodulens arbetsgång visas i Bild 7.

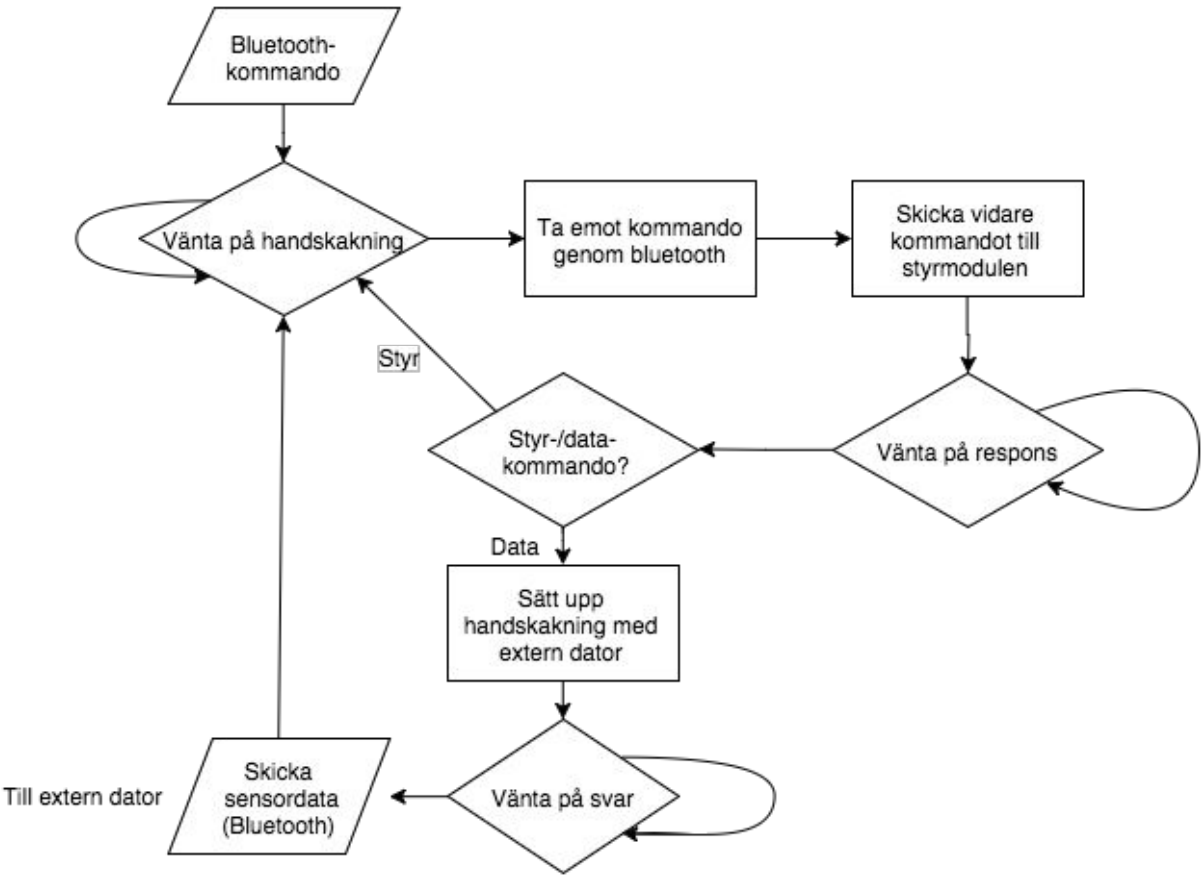

Bild 7. Flödesschema över kommunikationsmodulens arbetsgång

4.1. Firefly (Bluetooth)

Kommunikationen från PC till kommunikationsenheten kommer ske med Bluetooth. PCn kommer skicka en två byte stor sträng som innehåller en headerbyte med metadata och en databyte med funktionens namn (se kapitel 2.1). Fireflymodulen kopplas in till kommunikationsmodulen direkt på USART-ingångarna.

4.2. USART → Styrmodul

Kommunikationen mellan kommunikationsmodulen och styrmodulen kommer ske med USART. I det här fallet kommer kommunikationsmodulen vara master och styrmodulen slave vilket innebär att kommunikationsmodulen levererar klockpuls (XCK) till styrmodulen.

5. Styrmodul - Bjarne

Bjarne har till uppgift att styra och kontrollera motorerna som styr hjulen. Bjarne består av en AVR av typen ATmega1284. AVRen är ansluten till sensormodulen via USART och till kommunikationsmodulen via USART. Instruktioner om styrning eller datahämtning kommer hämtas från kommunikationsmodulen. Bjarne kommer också köra algoritmen för autonomt läge. Tre knappar kommer monteras på systemet. En knapp kommer bestämma om roboten arbetar autonomt eller styrs från datorn. En knapp används för att starta det autonoma läget. Dessutom kommer en resetknapp implementeras.

5.1. Motordrivning

Styrenheten innehåller två motorpar kopplade till höger respektive vänster hjulpar vilket gör roboten differentialstyrd. Motorerna kopplas in till styrmodulen via en IDC-kontakt, se Bild 8.

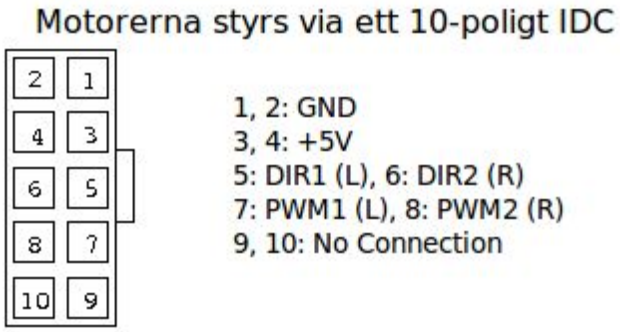

Bild 8. De olika pinnarna till motorstyrningen

Dessa motorer styrs via två signaler var; DIR och PWM. DIR styr motorenas riktning och PWM är en pulsbreddsmodulerad signal som kontrollerar motorernas hastighet, se Bild 9. PWM fungerar genom att fyrkantsvågor kontinuerligt skickas, i vårt fall från styrenheten till motorerna. Genom att slå av och på spänningen varierar man effekten hos motorerna, tätare pulser ger högre hastigheter.

Lämplig frekvens för PWM är maximalt 2 kHz. För att generera denna signal används en 8-bits räknare inkluderad i ATMega1284. Denna räknare jämför sitt nuvarande tal med ett referenstal som programmeraren bestämmer. När dessa två tal är lika sätts utsignalen till logisk 0. Räknaren fortsätter sedan räkna uppåt tills maxtalen nås och den börjar om igen. När räknaren börjar om sätts även utsignalen till logisk 1:a, vilket alltså resulterar i en fyrkantsvåg där referenstalen bestämmer förhållandet mellan $t_{hög}$ och $t_{låg}$.

PWM-signalen styrs via två olika register; TCNT0 och OCR0. TCNT0 styr räknarens funktion och OCR0 är referenstalet. PWM-signalens frekvens styrs genom att räknarens egna frekvens delas med ett heltal (prescaling). Uttrycket för PWM-frekvensen ser ut på följande sätt:

$$F = CLK/(N * 256)$$

Där CLK är datorns klockfrekvens (20 MHz) och N är heltalet. I vårt fall väljs N till 64 vilket ger en frekvens på 1,221 kHz.

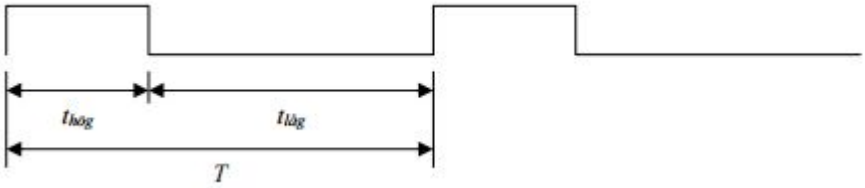

Bild 9. Större $t_{hög}$ ger större hastighet på motorerna.

5.2. Karta

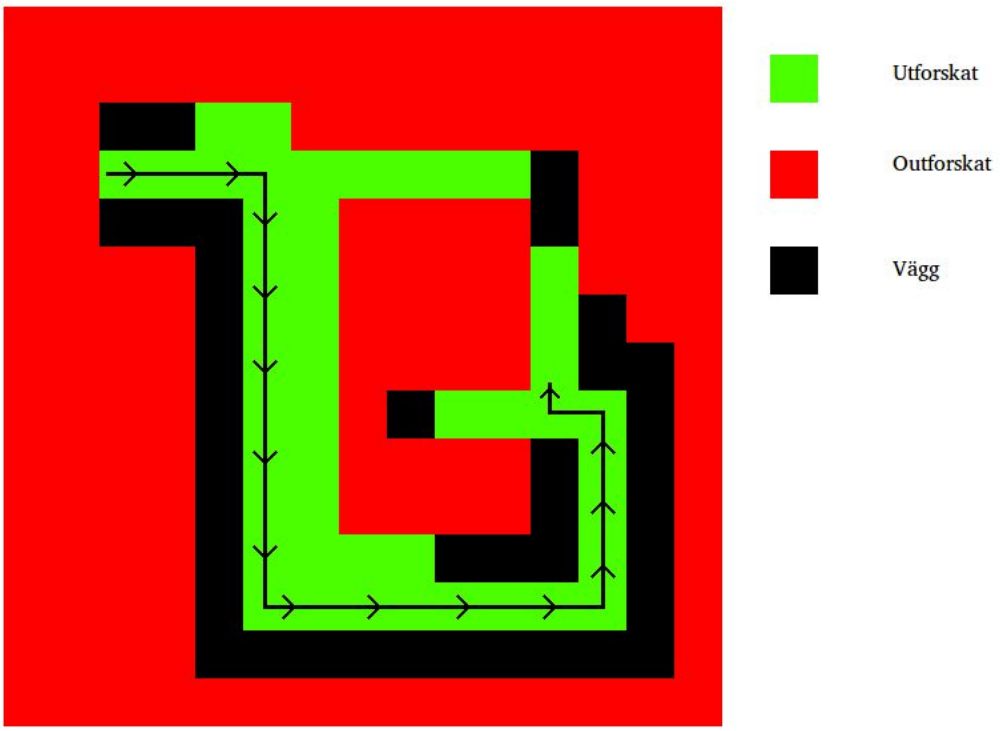

Bild 10. Representation av karta

Roboten kommer ha representation av den insamlade kartdatan. Representationen är en 2D-array vilken är 32x32 rutor där varje ruta representerar varje möjlig position för roboten. Roboten utgår från att den startar i mitten av den (16,16, 0-indexerad). Initialt är varje ruta markerad som outforskad (rött). Allt eftersom roboten upptäcker sin omgivning markerar den olika rutor i kartan som utforskad (grönt) eller vägg (svart), se Bild 10.

Direkt vid uppstart kommer roboten kunna markera utforskat/vägg mot sidorna och även framåt. Medan roboten åker runt kommer fler rutor ändra status från outforskad till vägg/golv. Då roboten håller sig längs högra kanten kommer den tillslut komma tillbaka till startpositionen. Då kommer den börja söka efter outforskade rutor och kartlägga dem.

När ny kartinformation upptäcks uppdateras kartan med den nya informationen. Samtidigt läggs den nya kartinformationen till i en förändringskö. Där lagras 2 bytes med information om koordinaterna och tillståndet rutan är i. Se Bild 11 för hur datan är placerad i de 2 bytesen. När den externa datorn vill uppdatera sin karta så frågar den efter en uppdatering från förändringskön, så att den inte behöver ladda över hela kartan varje gång.

Datan kommer även att ha en paritetsbit i varje byte för att garantera att datan inte blir felaktig, då det skulle ge felaktig information till användaren.

Eftersom det finns ett krav att datan ska kunna skickas igen ifall den är felaktig, så kommer styrmodulen att behöva lagra den senaste kartdatan som skickats, för att kunna skicka samma data igen utan att behöva vänta på ett svar om att datan var korrekt. Vid felaktig data så kommer kommunikationsmodulen att begära tempdatan istället för vanlig kartdata.

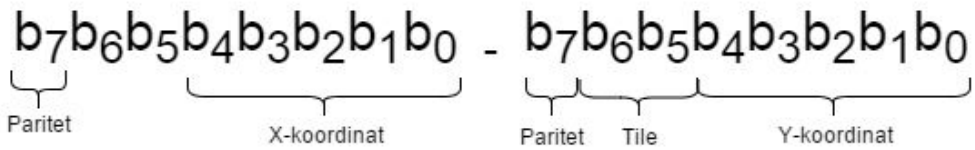

Bild 11. En förändring i kartan lagras med 2 bytes.

5.3. Autonoma läget

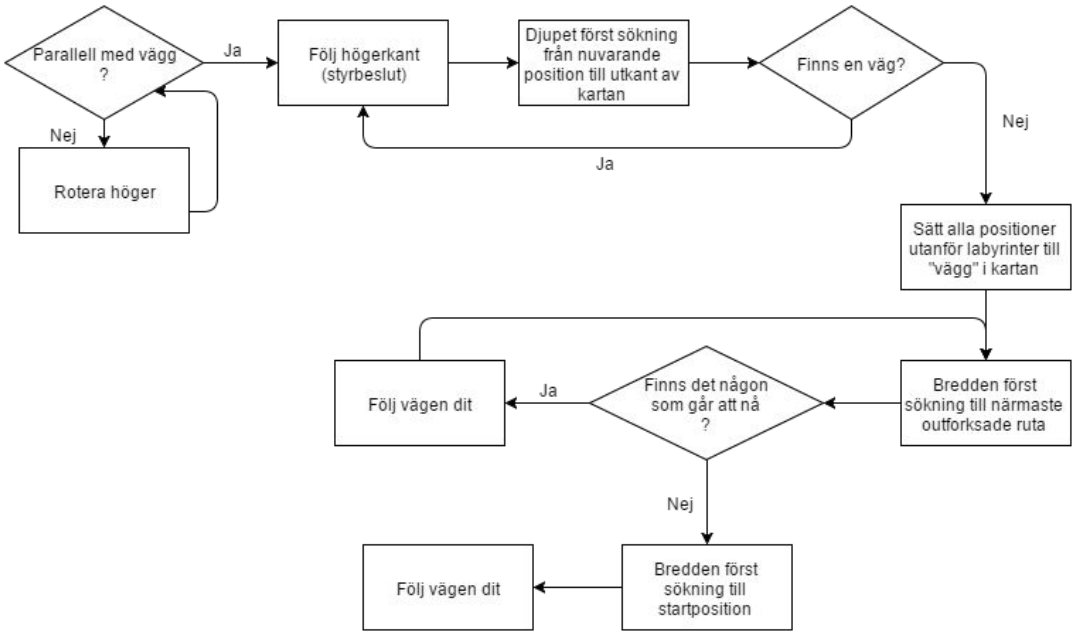

Bild 12. Flödesschema för styralgoritmen.

När systemet sätts igång är första steget att ställa sig parallellt med en vägg. “Följ högerkant” är en algoritm som går ut på att systemet hittar en vägg och följer den åt höger (följer högerkanten), se Bild 12. Samtidigt som roboten åker igenom rummet sparar den ner data om väggar i form av ett rutnät med information om vad som finns i varje ruta (vägg, öppen, outforskad). I varje iteration då kartdatat sparas görs en bredden-först-sökning för att hitta outforskade rutor. Finns det en outforskad ruta som vi kan ta oss till åker vi till närmsta sådan ruta. När det inte längre finns outforskade rutor är alla strukturer (yttervägar och ev köksö) hittade.

När alla strukturer är funna letar systemet sig tillbaka till startpositionen m.h.a. bredden-först-sökning. I Bild 12 visas ett flödesschema för hela styralgoritmen; de rutor som heter “Följ högerkant (styrbeslut)” innebär att systemet hela tiden väljer ett håll den ska åka

sträckan 40 cm (en ruta i rutnätet) med prioriteten höger, framåt, vänster, bakåt (se Bild 13). Svängarna i styrbeslutet är multipler av 90° och systemet använder ett gyro för att bedöma dessa vinklar (den sensordatan kommer att hämtas in under styrbeslutsteget).

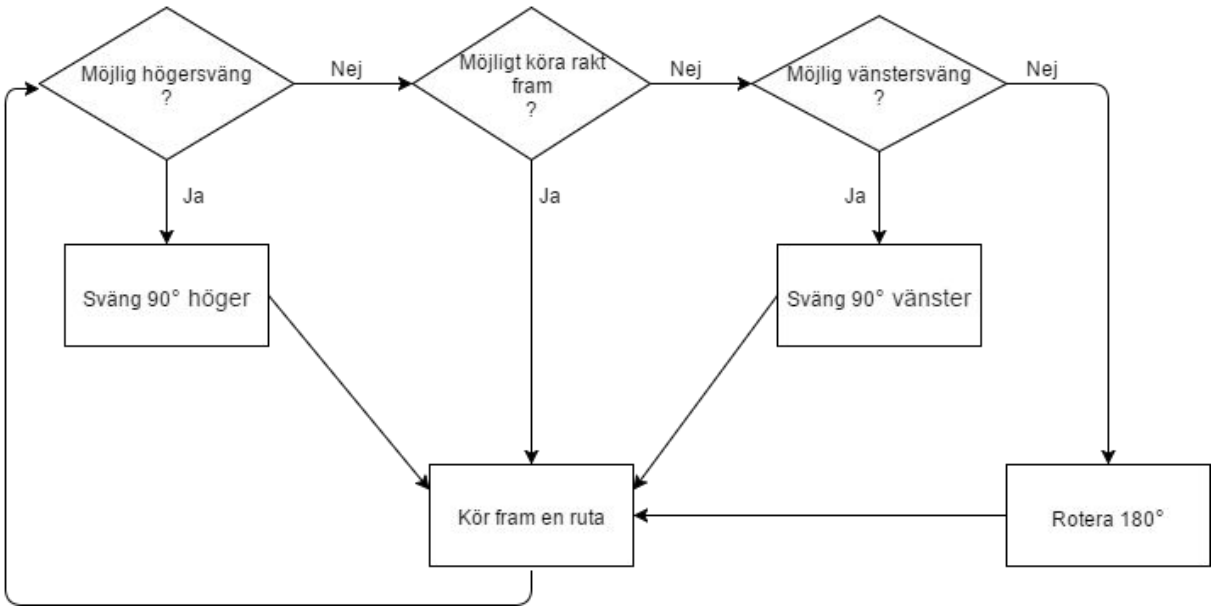

Bild 13. Flödesschema över styrbeslut för "följ högerkanten".

5.4. Reglering

Styrenheten är ansvarig för att i mjukvaran implementera reglering av styrningen så att roboten rör sig med mjuka och jämna rörelser när den åker längst med en vägg. För att få till detta behöver roboten kontrollera att den åker parallellt med väggen och justera styrriktningen ifall systemet inte är parallellt med väggen. Från sensorerna i sensormodulen kommer tokendata finnas som bedömer ifall systemet står parallellt med väggen, se avsnitt 6.1.2. Denna tokendata kommer ha några olika amplituder för olika amplitud på svängen som behöver göras för att korrigera felet. Styrmodulen hämtar också sensordata för tillryggalagd sträcka från sensormodulen för att bedöma när systemet åkt 40 cm, då ska det göra ett nytt styrbeslut. Bild 14 beskriver denna del av den autonoma styralgortimen.

Regleringens uppdateringstakt kommer bestämmas av sensorernas hastighet. De långsammaste sensorerna är IR-sensorerna som uppdateras var 50:e millisekund. Utifrån detta bestämdes uppdateringsfrekvensen till 25Hz.

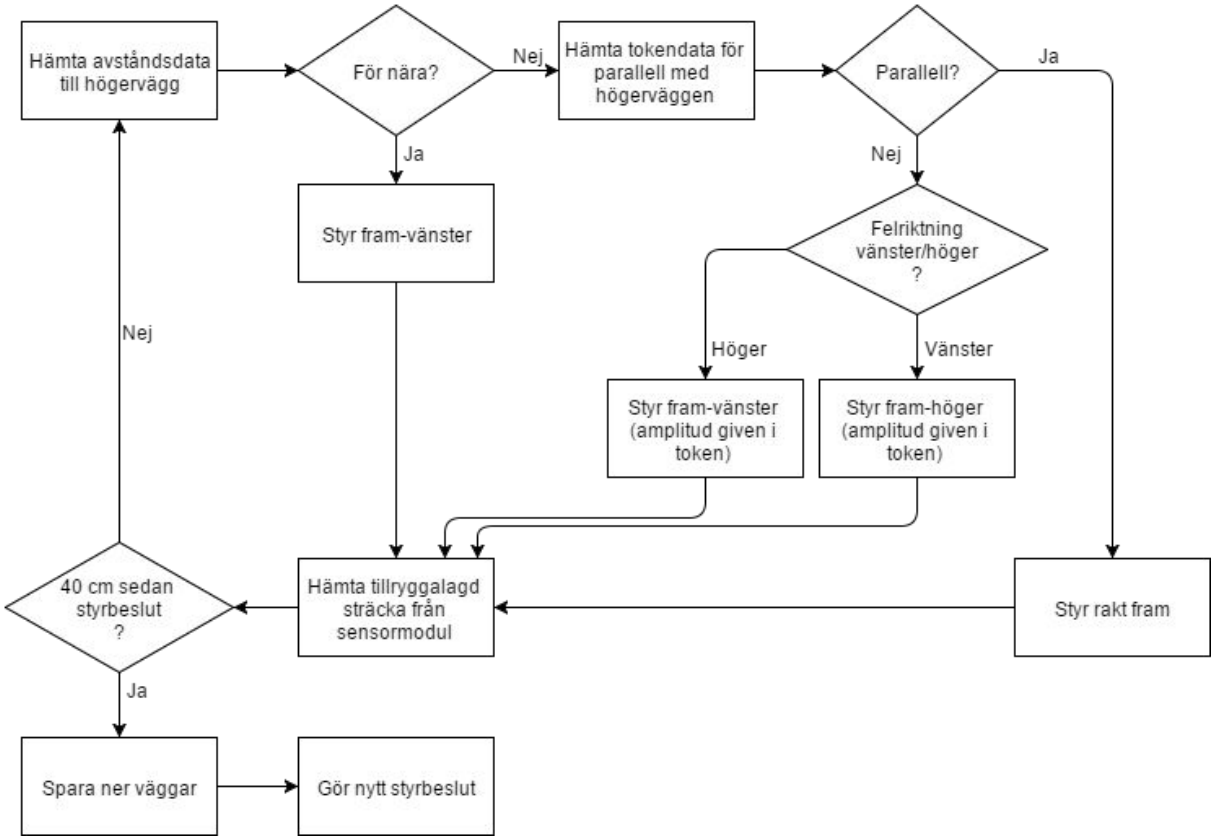

Bild 14. Flödesschema över justeringsalgoritmen.

5.5. Styrlogik

För att enkelt kunna översätta inkommande styrkommandon till motorrörelse kommer ett simpelt gränssnitt implementeras. Detta gränssnitt tar fyra argument som indata; spd_left, spd_right, dir_left, dir_right. Båda spd-argumenten är heltal mellan 0 och 100 vilka sedan mappas mot 0-240 och används sedan som referenstal till PWM-signalen. DIR är true/false-värden och styr vänster respektive höger motors riktning.

- För att åka rakt framåt (100,100,1,1)
- För att svänga vänster (100,100,0,1)

Styrkommandon som inkommer är av typen “vänster pil trycktes ner”, “uppåtpilen släpptes” vilket tillåter roboten att svänga och köra framåt samtidigt.

5.6. Knappar

Tre knappar kommer monteras på systemet. En knapp kommer bestämma om roboten arbetar autonomt eller styrs från datorn. En knapp används för att starta det autonoma läget. Dessutom kommer en resetknapp implementeras som kommer vara låg tills en stund efter systemstart. Detta kommer implementeras enligt bild 15. Föreslagna komponentvärden är 22 nF på kondensatorn och 4,7k Ohm.

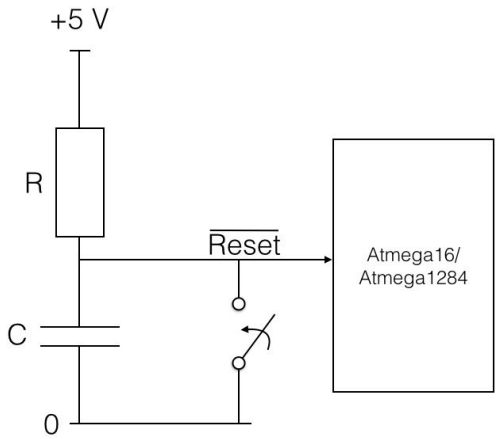

Bild 15: Bild på implementationen av resetkanppen.

6. Sensormodul - Mimers brunn

Mimers brunn har till uppgift att samla in data från de olika sensorerna och filtrera/omvandla datat till ren form. Sensorenheten kommer bestå av ett ATmega16-kort.

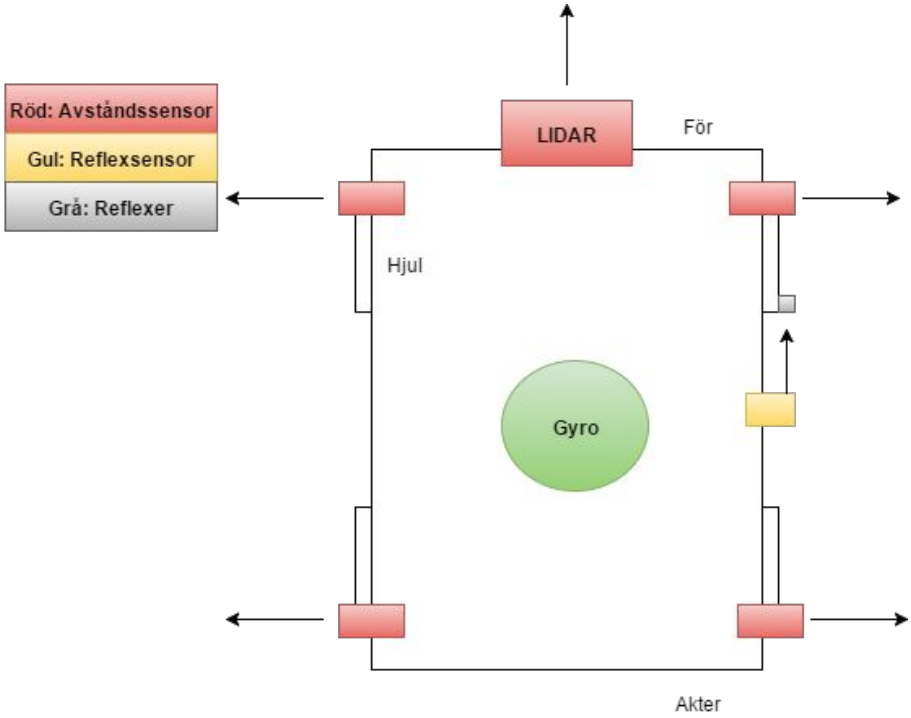

Bild 16. Sensorerna och deras placering på roboten

6.1. Sensorer

I modulen används fyra olika typer av sensorer, se Bild 16:

- Odens öga, sensor som mäter avståndet föröver
- Hugin och Munin, sensorer som mäter avståndet till väggarna
- Sextanten, gyroskop som mäter rotationen av roboten
- Hel, reflexsensor för att mäta tillryggalagd sträcka

6.1.1. Odens öga (LIDAR)

För att mäta avstånd fratmåt ska en LIDAR lite v2 lasersensor användas. Den ska monteras på robotens framsida. Avståndsdatan från LIDAR lite v2 kommer att skickas via TWI (Atmels version av I^2C) för mer information se avsnitt 2.1.4. Datan skickas utan filtrering direkt till sensorenheten.

6.1.2. Hugin och Munin (IR-sensorer)

Avståndet till väggarna ska mätas med hjälp av 4 st IR sensorer av typ GP2Y0A21. Sensorerna ska monteras så att två är åt vänster och två åt höger. Dessa sensorer lämpar sig för avståndsmätning mellan 10 och 80 cm. Sensorerna matas med 5 volt till Vcc och de skickar ut en signal på V_0 utgången. Sensorerna har en uppskattat uppdateringsfrekvens på 39 ms. Dåliga mätvärden kan erhållas om sensorerna blir utsatta för direkt ljus, därför kommer vi tillverka höljen som sensorerna ska sitta i för att skärma av dem.

För att få ut ett mått på hur parallell roboten är med väggen räknar vi ut det med formeln (S1/S2)-1. S1 är sensorn främre och S2 den bakre. Ett tal nära noll är "bra parallell" och ett stort tal ej är parallell. Detta tal får trimmas fram men borde lämpligt vara max ±0.1. Om talet är positivt betyder det att roboten är snett bort från väggen och om talet är negativt att den är sned mot väggen.

6.1.3. Gyro

För att kunna veta när man svängt 90 grader, utan att behöva bry sig om eventuellt olika slitage på motorerna och sliring etc, används ett gyroskop MLX90609. MLX90609 skickar ut datan till processorn via SPI, se avsnitt 2.1.1 för mer information om detta.

För att svänga 90 grader så samplar vi data från gyrot med bestämd frekvens. Vi summerar bidragen och jämför summan av vinkelhastigheterna med ett innan uppmätt värde som motsvarar 90 grader.

6.1.4. Hel (Reflexsensor)

För att kunna mäta tillryggalagd sträcka kommer en reflexsensor monteras vid ett framhjul. På hjulet kommer markeringar vara utsatta så reflexsensorn kan upptäcka ljusförändringar och på så sätt mäta hur långt roboten har åkt. Utsignalen kommer variera mellan 0 och +5V beroende på hur mycket ljus hjulen reflekterar.

Vid avståndsberäkning kommer vi försöka åka parallellt med en vägg och göra mätningar med hjälp av Odens öga. Vår förhoppning är att detta kommer räcka för avståndsberäkning och vi kommer därför inte börja implementera Hel utan istället försöka med enbart Odens öga. Om detta inte fungerar kommer vi komplettera med Hel.

6.2. Förstärkning, filtrering, formatering (FFF) av IR

Vi har valt att uppdatera värdena från sensorna 20 gånger per sekund, alltså efter 50 ms.

Ifall det visar sig att IR sensorerna ger dåliga värden kan signalen förstärkas innan den går igenom filtret. Isåfall görs detta genom att det från IR-sensorernas V_0 utgångar skickas signaler genom en ickeinverterande förstärkarkrets byggd av opAmp LM324 och två resistorer, se Bild 17. V_0 skickas genom operationförstärkarens Vin och vidare till Vut. Operationförstärkaren matas med 5 volt till Vcc.

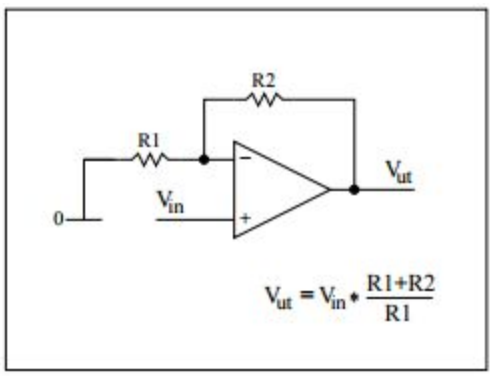

Bild 17. Ickeinverterad förstärkarkrets.

Signalen från Vut skickas genom ett lågpassfilter bestående av följande komponenter: 1 resistor á 18 kohm och 1 kondendsator á 100 nF, detta ger en brytfrekvens på ungefär 88.42 Hz. Bild 18 visar kopplingsschemat över lågpassfiltret.

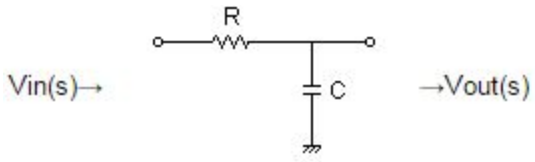

Bild 18. Lågpassfilter.

Den filtrerade signalen ska sedan vidare genom ATmega16-kortets A/D-omvandlare.

I fallet med signalen från IR-sensorerna behöver processorn omvandla den icke-linjära signalen från volt till cm. Detta görs genom att vi skapar en map med voltvärden och dess motsvarande avstånd och använder den för omvandlingen. För att försöka undvika felvärden kommer en medelvärdesberäkning utföras.

6.3. Sensorenhetens implementation

Sensorernas värden kommer läsas ut kontinuerligt av sensormodulen i ordning. Detta gör så att att begärd data kan sändas direkt vid begäran från styrmodulen. Detta sker via USART (eftersom SPI är upptaget av gyrot) se avsnitt 2.1.4 för mer information om USART. De olika datavariablerna listas i Tabell 2.

6.4. Tokenfiering

IR-sensorernas data görs om till två bitar, antingen 10 om vägg närmast, 01 om vägg ett steg bort, 00 annars. Bitarna läggs i en byte. För att skapa token om hur parallellt roboten står tas värdet för främre sensorn delat på det för bakre sensorn vilket ger en kvot. Är kvoten tillräckligt nära 1 anses systemet vara parallelt med väggen.

Reflexsensorns data kontrolleras och tokenvärdet plussas med 1 ifall värdet ändrades.

Gyroskopets data omvandlas till hur många grader som svängts genom att de uppmätta vinkelhastigheterna summeras och jämförs med en referensmap. Summationen tas sedan modulo 90 grader.

Laser-sensorns data görs om till antalet tomma rutor innan vägg.

7. Komponentlista

Tabell 3. Komponentlista

Namn	Antal	Beskrivning
Allmänt		
Chassi	1	4-hjuligt Terminatorchassi till roboten
Virkort	3	Ett virkort per modul
Tryckknappar	1	Avstudsad tryckknappsmodul för att starta/stoppa det autonoma läget
Tryckknapp	1	För att växla mellan det radiostyrda och autonoma läget
Kommunikation		
AVR-ATmega1284	1	Mikrokontroller
Firefly kort	1	För Bluetoothkommunikationen
Sensorer		
AVR-ATMega16	1	Mikrokontroller
LIDAR	1	För att mäta avståndet rakt framåt från roboten
Reflexsensor	1	För att mäta åkt sträcka genom att mäta ljusskillnader på ett hjul. Se Bild Sensor
IR sensor	4	För att mäta avståndet till väggarna
OPAMP	4	Förstärka signalen från IR-sensorerna
Lågpassfilter	4	Filtrera signalen från OPAMP
GYRO	1	Mäter förändring i rotation, för att kunna svänga korrekt
Styrning		
AVR-ATmega1284	1	Mikrokontroller

8. Referenser

8.1. Datablad

1. Datablad PWM:
   https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/pwm-motorstyrning.pdf
2. Guide PWM
   http://brigadr starts counting up, TCNT0 increases. Until
   TCNTa-diverse-4u.blogspot.se/2013/06/pwm-configuration-for-atmega16.html
3. PyBluez:
   https://github.com/karulis/pybluez
4. Pygame:
   http://www.pygame.org/wiki/about
5. Datablad Firefly:
   https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/firefly.pdf

Appendix A - Kopplingschema