Teknisk dokumentation för kartrobot

Patrik Sletmo

Version 1.0

Status

STATUS	Patrik Sletmo	2016-12-DD

Projektidentitet

Grupp 3, 16/HT, KarToffel
Linköpings tekniska högskola, ISY

Namn	Ansvar	Telefon	E-post
Patrik Sletmo	Projektledare	070 783 57 61	patsl736@student.liu.se
Rebecca Lindblom	Utvecklare	073 436 40 79	rebli156@student.liu.se
Matildha Sjöstedt	Utvecklare	070 515 84 11	matsj696@student.liu.se
Sebastian Callh	Utvecklare	073 820 46 64	sebca553@student.liu.se
Anton Dalgren	Utvecklare	076 836 51 56	antda685@student.liu.se
Matilda Dahlström	Utvecklare	070 636 33 52	matda715@student.liu.se

Hemsida: https://github.com/SebastianCallh/kartoffel-tsea29

Kund: Mattias Krysander, 013 - 28 2198 , matkr@isy.liu.se

Kursansvarig: Tomas Svensson, 3B 528, +46 (0)13 28 1368, tomas.svensson@liu.se
Handledare: Anders Nilsson, 3B 512, +46 (0)13 28 2635, anders.p.nilsson@liu.se

Innehållsförteckning

Dokumenthistorik

Version	Datum	Utförda ändringar	Utförd av	Granskad
VERSION	2016-12-DD	Första version	Grupp 3	Patrik Sletmo

1 Inledning

1.1 Bakgrund

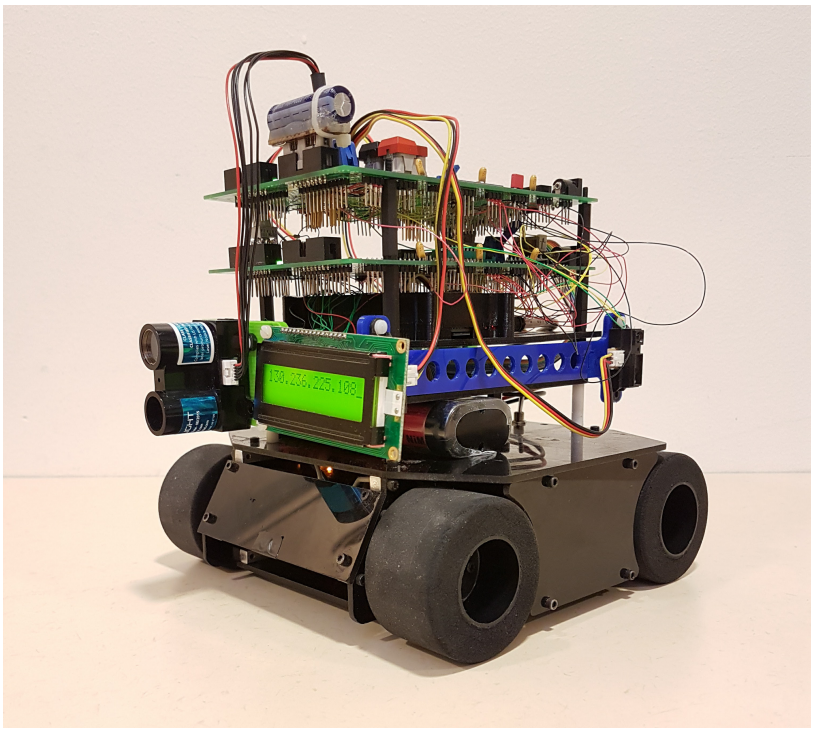
Som en del av utbildningen till civilingenjör i datateknik på Tekniska Högskolan vid Linköpings Universitet har vår projektgrupp i kursen Konstruktion med mikrodatorer (TSEA29) arbetat mot beställaren Mattias Krysander för att tillverka en robot vars syfte är att kartlägga en sluten bana. Den färdiga produkten tävlar mot andra gruppers robotar och redovisas i samband med kursens avslutande.

1.2 Syfte

Det här dokumentet skall ge en ingående beskrivning av kartrobot-systemet från ett tekniskt perspektiv. Dokumentet ska fungera som konstruktionsunderlag för någon som vill bygga en kopia av den robot som beskrivs. Det ska från instruktioner i dokumentet även vara möjligt att underhålla och felsöka både mjuk- och hårdvaran i systemet.

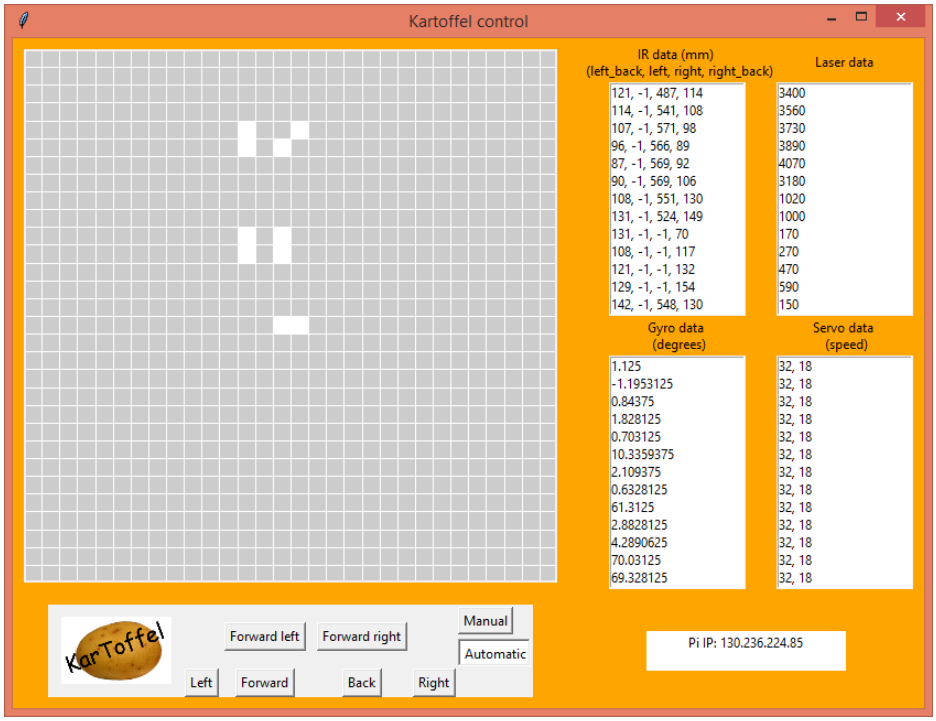
2 Produkten

Systemet består av en fyrhjulig robot samt en mjukvaruklient som kommunicerar med varandra via Bluetooth. Systemets syfte är att roboten ska kunna kartlägga sin omgivning helt autonomnt och förmedla kartläggningen till mjukvaruklienten. För att göra det så är den utrustad med diverse IR-sensorer, en lasersensor och ett gyroskop för att kunna navigera och mäta avstånd. Se figur 1 för en bild på konstruktionen.



Figur 1: En bild på roboten.

Mjukvaruklienten är som nämnt ansluten via Bluetooth, vilket låter den rendera kartläggningen i realtid. Den har även ett gränssnitt för att kunna fjärrstyra roboten. Se figur 2 för en bild på mjukvaruklientens gränssnitt.



Figur 2: En bild på mjukvaruklienten.

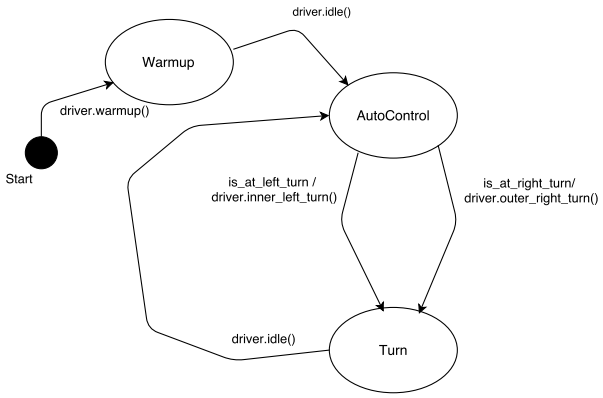
3 Teori

Den övergripande strategin för att kratlägga rummet är att alltid färdas längst väggen på höger sida och på så sätt kartlägga rummets ytterväggar. För det krävs en regleringsalgoritm för att följa väggen och logik för navigering samt kartläggning.

3.1 Navigering

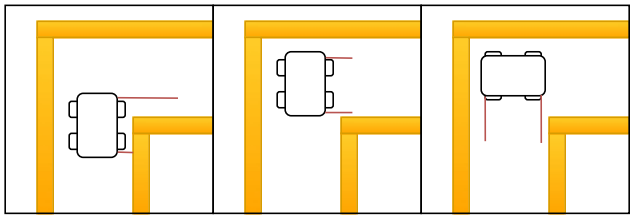
Navigeringslogiken implementeras som en tillståndsmaskin vars transitioner styrs av indata från robotens sensorer. Den antar att omgivningen följer banspecifikationen i bilaga A.

Efter en kort uppvärmningsperiod för att hinna läsa in alla sensorer så börjar roboten följa väggen på sin högra sida och håller ett lämplig avstånd till den med hjälp av en modifierad PID-reglering (se avsnitt 3.3). Om roboten upptäcker att väggen på höger sida tar slut så innebär det att väggen har svängt av bort från roboten och att den behöver göra en yttersväng till höger för att följa efter. Skulle roboten upptäcka en vägg framför sig betyder det istället att rummet svängt av till vänster, och att roboten behöver göra in innersväng åt vänster. En återvändsgränd består ur robotens perspektiv som två stycken innersvängar till vänster. Ett flödesdiagram över robotens navigeringstillstånd kan ses i figur 3.



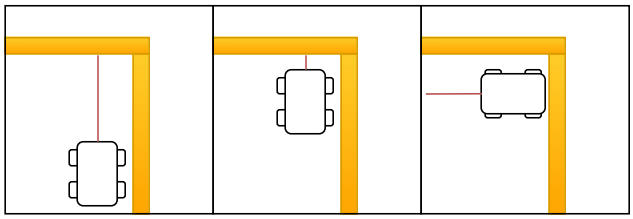
Figur 3: Ett flödesdiagram på navigatorns tillstånd.

Flödesdiagramet i 3 är all logik som behövs för att roboten ska kunna följa en vägg. De olika momenten som dyker upp under kartläggningen illustreras och förklaras i figurerna 4, 5, 6.



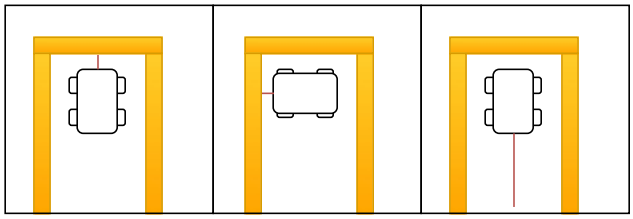
Figur 4: En illustration på hur roboten hanterar en yttersväng.

Det första momentet som roboten behöver kunna genomföra är en yttersväng till höger, vilket illustreras i 4. För att detektera en sväng så använder den sina två IR-sensorer monterade fram respektive bak på höger sida. När den första IR-sensorn inte längre detekterar en vägg vet roboten att den snart ska rotera, och när den bakre gör samma sak utförst en rotation 90 grader åt höger.



Figur 5: En illustration på hur roboten hanterar en innersväng.

För att utföra en innersväng använder sig roboten av lasersensorn monterad framtill och mäter avståndet till väggen. När avstånder underskrider ett visst tröskelvärde så vet roboten att den ska rotera 90 grader till vänster. Sekvensen illustreras i figur 5. Värt att notera är att yttersväng till höger tar prioritetet över att göra en innersväng åt vänster. Det vill säga att även om lasersensorns mätvärde underskrider tröskelvärdet så kommer det att ignoreras ifall robotens främre högra IR-sensor meddelar att den är på väg mot en högersväng.



Figur 6: En illustration på hur roboten hanterar en återvändsgränd.

När roboten tar sig förbi en återvändsgränd faller den i själva verket in i samma rutin som när den utför an innersväng åt vänster upprepat två gånger. Roboten roterar när den närmar sig slutet på återvändsgränden, och direkt efter rotationen upptäcks en vägg framför den och den roterar ytterligare 90 grader för att sedan fortsätta ut ur återvändsgränden. En illustration ses i figur 6.

3.2 Kartläggning

3.2.1 Positionsbestämning

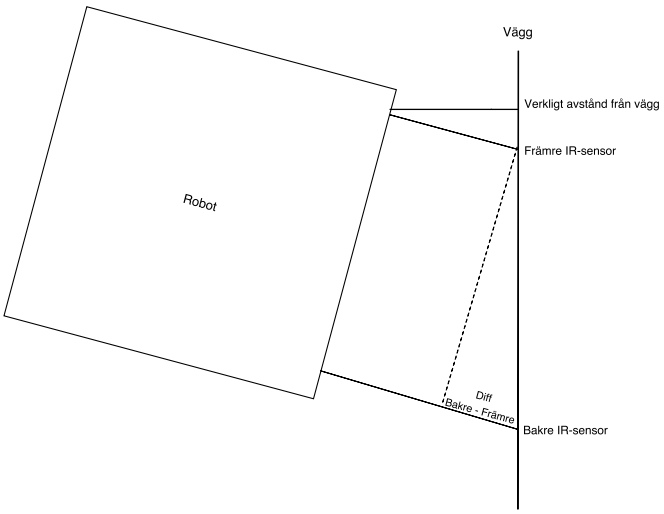
Under kartläggningen av rummet så räknar roboten sektioner som den har passerat och utnyttjar det för att bestämma sina koordinater när en sektion har passerats. Robotens start koordinater är alltid i origo och startriktiningen är alltid ”norrut”, alltså positivt längs Y-axeln. När roboten enligt navigeringsalgoritmen i avsnitt 3.1 bestämmer sig för att ta en sväng och rotera så kommer kartläggningen att spara den nuvarande positionen. Detta genom att vid varje inläsning av laserdata har roboten sparat detta i en lista och tagit ut skillnaden mellan det högsta och lägsta avstånd som lasern har uppmätt och dividerat med storleken på en ruta i rutnätet. Då fås antalet rutor som roboten har förflyttat sig i en viss riktning och på så sätt vet roboten sin nuvarande position. Positions bestämnigen är även den baserad på en tillståndsmaskin. Den är baserad på två tillstånd, väntar och mäter, där väntar är det tillstånd som roboten befinner sig i när den gör en 90 graders rotation. Därefter går roboten in i tillståndet mäter där en ny sektion initieras och mätdata sparas varje varv i huvudloopen.
Under positionsbestämningen så kommer roboten att hålla utkik efter väggar på vänstersidan för att kunna lokalisera eventuella väggar på en köksö. Även dessa är uppbygda av sektioner då roboten säger att den här sektionen kan vara en eventuell köksö och sparar undan koordinaterna när den inte längre ser väggen på vänster sida. Om väggen som roboten såg till vänster redan ligger i den besökta vägen så kommer den inte att sparas som en del av en potentiell köksö. Om roboten passerar en koordinat som ligger i listan över koordinaterna för den potentiella köksön så kommer dessa att tas bort och således definieras dessa koordinater som del av en yttervägg.

3.2.2 Bestämning av köksö

När roboten har åkt ett varv längs ytterkanterna kommer den att gå in i ett läge där den letar efter köksön. Då kommer den att kolla i listan över potentiella köksöar och hitta minst en koordinat som representerar en vägg på köksön. Roboten kommer att åka längs ytterkanterna tills den kommer till den här koordinaten och svänger in till köksön för att sedan ta ett varv runt den för att bestämma vilka koordinater som köksön befinner sig på. Detta sker på samma sätt som beskrivs ovanför i avsnitt 3.2.1. När roboten har tagit sig ett varv runt köksön fortsätter den att åka längs ytterväggarna och tillbaka till garaget för att slutföra uppdraget.

3.3 Reglering

Roboten reglerar längs en vägg på sin högra sida med hjälp av två IR-sensorer, en vid robotens främre del och en vid robotens bakre del. Ett önskat värde är bestämt som det avstånd som roboten ska hålla ifrån väggen den reglerar emot. Med hjälp av det önskade värdet tillsammans med värdena ifrån de två IR-sensorenra räknas ett distansfel ut. Distansfelet räknas ut genom att dra bort främre IR-sensorns värde ifrån det önskade avståndet. Sedan läggs ett värde proportionellt mot differensen mellan bakre och främre sensorn till för att efterlikna det verkliga avståndet från väggen (se figur 7). Detta då värdet ifrån IR-sensorerna ökar mer än det verkliga värdet desto mer vinklad ifrån väggen roboten står.
Med hjälp av distansfelet och differensen utförs en enklare PID-reglering som istället för I- och D-delen använder sig av differensen och returnerar ett regleringsvärde u. Regleringsvärdet räknas ut enligt formeln $u[n] = Kp*e[n]+Ka*d$, där Kp och Ka är konstanter, $e[n]$ är distansfelet vid tid n, och d är differensen mellan värdena på den bakre och främre IR-sensorn. Regleringsvärdet adderas respektive subtraheras sedan på en standard-hastighet och skickas till de båda hjulparen för att räta upp roboten och få den att följa väggen och hålla ett önskat avstånd.



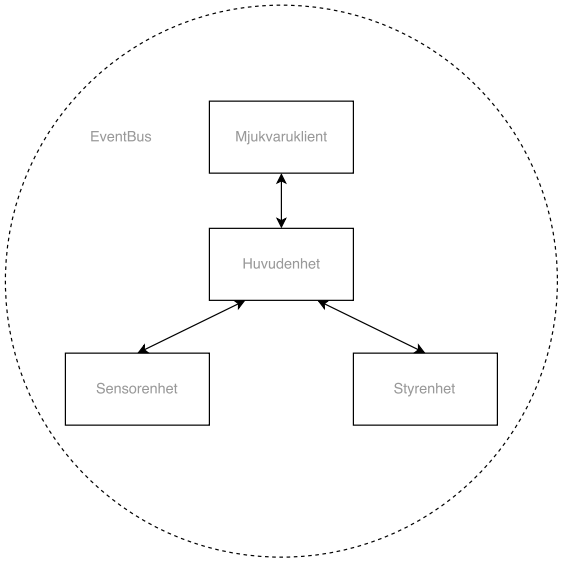
Figur 7: En illustration över hur roboten reglerar.

3.4 Kommunikation

Systemet byggs upp av ett antal separata moduler som behöver samarbeta för att utföra robotens funktion då dess unika funktioner är alldeles för specifika. Kommunikationen som möjliggör detta samarbete sker både via Bluetooth och I2C beroende på modulernas placering i systemet. Då det kan bli förvirrande att hantera dataflödet på två olika protokoll ligger det ett abstraktionslager som kallas EventBus ovanpå dessa överföringsmedier. Detta abstraktionslager samt de underliggande implementationerna beskrivs mer ingående i följande delavsnitt.

3.4.1 EventBus

För att underlätta programmering av logikenheten använder sig systemet av en distribuerad eventbuss kallad EventBus som tillhandahåller funktioner för att skicka data till en annan enhet på bussen och för att lyssna på mottagna meddelanden. Eventbussen är delad mellan sensorenheten, styrenheten, huvudenheten samt mjukvaruklienten men eftersom bussen saknar funktionalitet för att delegera data vidare till andra enheter kan data inte skickas direkt från t.ex. sensorenheten till styrenheten, se figur 8. Eventbussen saknar fysiskt representation i den mening som t.ex. I2C-bussen har utan är bara en abstraktion som med hjälp av I2C-bussen och Bluetooth-anslutningen realiserar dess distribuerade funktion.

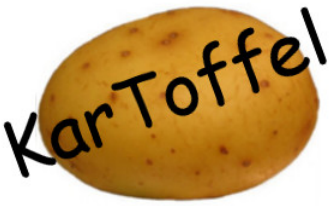


Figur 8: Översikt över eventbussens möjliga kommunikationsriktningar

Tanken med eventbussen är att programmeraren ska slippa tänka på att ett funktionsanrop exekveras på en annan enhet eller varifrån data kommer, så länge det läggs åtanke på att funktionerna exekveras asynkront. All mottagen data behöver manuellt behandlas i ett periodiskt funktionsanrop innan de "lyssnande" funktionsanropen körs. För att få en tydligare och mer deterministisk körning av huvudenhetens program läses all mottagen data in i början av varje upprepning av huvudloopen. Data ut från huvudenheten skickas till skillnad från inläsningen direkt när anropet sker för att säkerställa att data kommer fram i andra änden så tidigt som möjligt.

3.4.2 Bluetooth

TODO:TEXT OM BLUETOOTH. UPPDATERA SÅ DEN REFLEKTERAR VÅR IMPLEMENTATION. UPPDATERA BILD

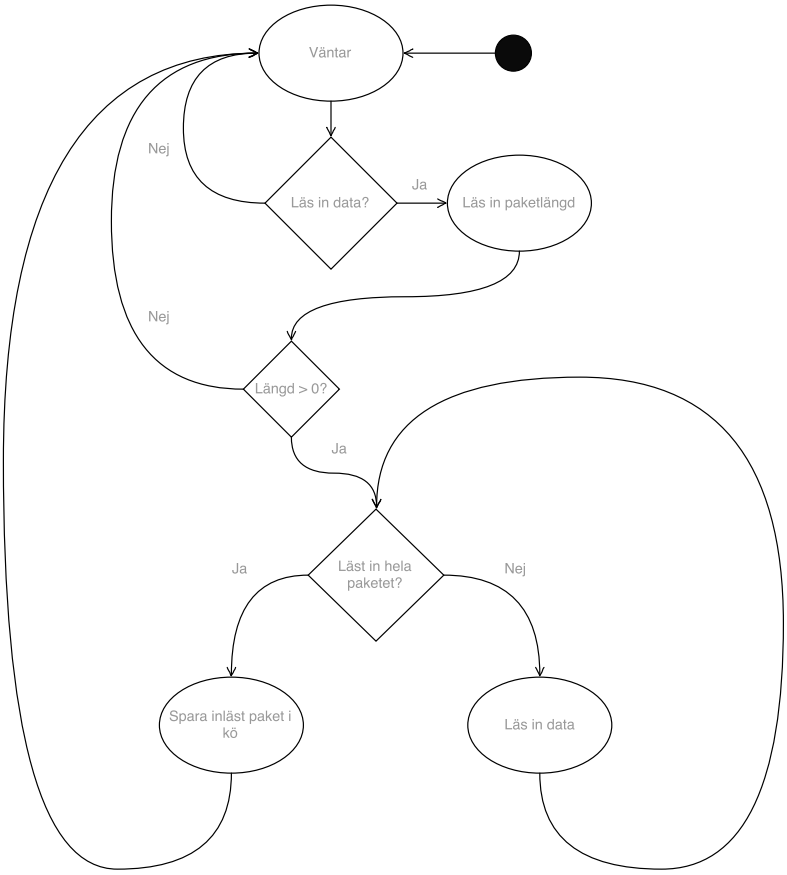


Figur 9: Flödesschema över Bluetooth-kommunikationen i huvudenheten.

Bluetooth kommunikationen i huvudenheten fungerar på så vis att huvudprogrammet kommunicerar via det allmänna gränssnittet EventBus till en bluetoothserver. Servern körs i en separat process än huvudprogrammet på huvudenheten och kommer ständigt kolla om det finns ny data att ta emot respektive ny data att skicka (se figur 9), tills den får ett omstart kommando. Kommandon till huvudenheten samlas i en inkö som EventBus hämtar ifrån, på samma sätt samlas kommandon som skickas till EventBus - med destination bluetoothklienten - i en utkö där servern hämtar ifrån. Dessa in- och utköer gör det även möjligt för processen som servern ligger i och huvudprocessen där huvudprogrammet och EventBus ligger i att kommunicera.

3.4.3 I2C

För att kunna kommunicera med olika moduler och sensorer på själva roboten används en I2C-buss som delas mellan huvudenheten, sensorenheten, styrenheten, laser-sensorn, gyroskopet samt accelerometern där huvudenheten agerar master och alla andra enheter slavar. Eftersom kraven som ställs på kommunikationen mellan modulerna förutsätter att data kan skickas i form av anrop räcker inte I2C-bussens protokoll till utan skrivningen och inläsningen till/från adresser behöver utökas med funktionalitet för att skicka faktiska datapaket. Det påliggande protokollet implementerar köer av paket för båda datariktningarna på AVR:erna som pollas och fylls på från huvudenheten. Varje paket innehåller ett antal bytes för dess data där den första byten innehåller paketets id. Eftersom I2C endast tillåter läsning eller skrivning av en enda byte innehåller varje paket on-the-wire också en inledande byte som specificerar dess längd, vilket ger en maxlängd av 255 bytes. När ett paket lästs in läggs det i en intern kö som hanteras av den modulspecifika koden, se figur 10.

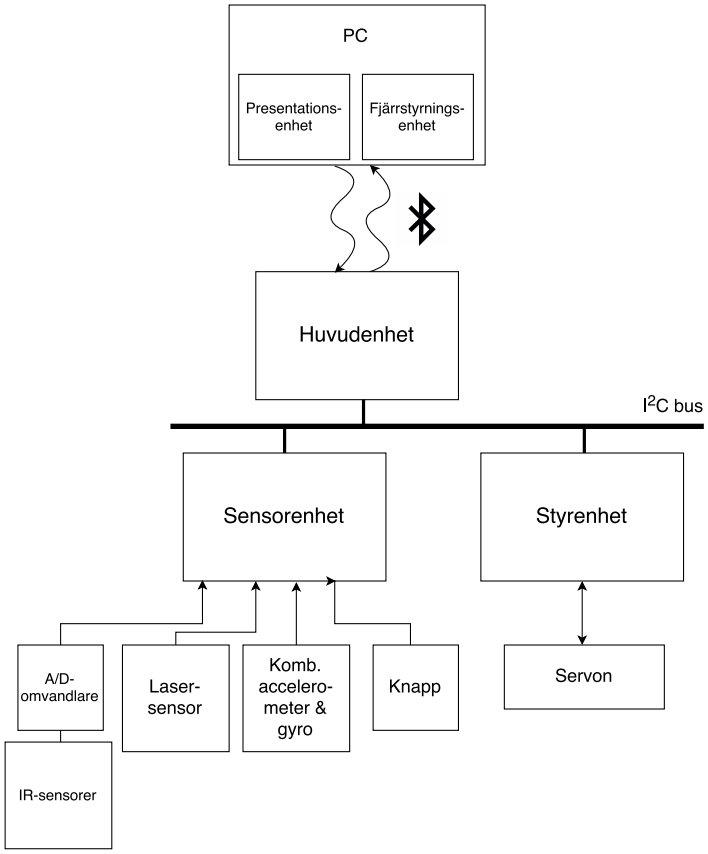


Figur 10: Flödesschema över I2C-kommunikationen

Alla inkopplade enheter på I2C-bussen använder sig inte av det protokoll som beskrivs ovan utan fungerar på det vanliga sättet genom läsning och skrivning till adresser. Dessa enheter är laser-sensorn, gyroskopet samt accelerometern och adresseras direkt från huvudenheten istället för att passera via sensorenheten som alla andra sensorer till följd av begränsningar som följer från single-master implementationen. Läsning från dessa sensorer sker periodvist i huvudenhetens programloop och sparas undan så att olika delar av programmet kan komma åt det senaste datat utan att behöva läsa multipla gånger.

4 Systemet

TODO:Text om systemet i helhet



Figur 11: En bild på systemet övergripande konstruktion.

4.1 Felsökning av mjukvara

Systemet är redan från början av projektet designat för att vara lätt att felsöka. Det finns enkla gränssnitt både in och ut från robotens alla processorer som gör att data kan manipuleras och inspekteras utan att behöva pausa programmet eller spara undan data i filer.

4.1.1 Felsökning på AVR

Eftersom AVR:erna i sitt grundutförande inte kan ge någon avancerad utdata finns möjligheten att på roboten koppla varje AVR direkt till en dators USB-port med hjälp av en seriell anslutningskabel. Genom att öppna seriellporten

i t.ex. Putty (Windows) eller screen (UNIX) kan data skickas fram och tillbaka mellan datorn och mikroprocessorn, se listing 1. I den nuvarande konfigurationen stöds bara möjligheten att skicka data från roboten till datorn men gränssnittet stödjer även den motsatta datariktningen. Implementationen bygger på att ersätta stdout-strömmen så utskrift till dator kan göras med det bekanta gränssnitt som definieras i stdio.h, se listing 2 för ett komplett exempel.

```
$ screen /dev/tty.usbserial
```

Listing 1: Anslutning via screen

```
#include "common/debug.h"

int measuredValue = 42;
printf("Measured value: %d\n", measuredValue);
```

Listing 2: Exempel av utskrift

4.1.2 Felsökning på Huvudenhet

Huvudenheten består av en Raspberry PI och kör kod skriven i Python, vilket gör att dess felsökningsmöjligheter är nästintill så bra som det går. Enklast är att ansluta sig till huvudenheten med en SSH-uppkoppling över Eduroam men det går även att koppla in tangentbord, mus och skärm direkt till roboten om så önskas. För att ansluta till enheten via SSH så loggar man in med användaren "pi" på den IP-adress som visas på robotens display och anger användarens lösenord. Notera att displayens innehåll uppdateras varje minut-omslag och det kan dröja ett par minuter innan roboten kopplar upp sig mot nätverket första gången.

4.2 Felsökning av hårdvara

På grund av varierande spänningsnivåer så inträffar det ibland fel i hårdvaran där mjukvaran har svårigheter att korrigera för felet på egen hand. De vanligaste felen och dess lösningar beskrivs i detta avsnitt.

4.2.1 Låg batterispänning

I stort sätt alla fel som uppkommer i hårdvaran beror på en för låg batterispänning. Det finns inget sätt att kontrollera robotens batterispänning från mjukvaruklienten eller huvudenhetens terminal utan batteriet måste avmonteras från roboten för att därefter testas med en multimeter. Lättast sättet att upptäcka att batteriet har för låg spänning är genom att observera robotens beteende. Vanliga symptom är:

- Trötta servon
- Sena svängar i hörn

- Annorlunda reglering
- Fel mätvärden
- Bruten anslutning över WiFi

Ifall något av dessa symptom observeras bör batteriet omedelbart bytas mot ett nyladdat för att undvika skador på robot, omgivning eller människor.

4.2.2 Felsökning av I2C

Ibland händer det att I2C-bussen tappar bort någon enhet eller helt enkelt slutar fungera, särskilt vid låga batterinivåer. Det går att verifiera att I2C-bussen fungerar som den ska genom att köra ett kommando från huvudenhetens terminal, se listing 3 för kommando och förväntad utdata för en hälsosam buss. Om bussen är trasig så kommer huvudenhetens program att sluta fungera och skriva ut ett felmeddelande i loggen. I de flesta fall går det att lösa problemet genom att bryta och sedan slå på spänningsmatningen till roboten men om batterinivån är låg kan batteriet behöva laddas.

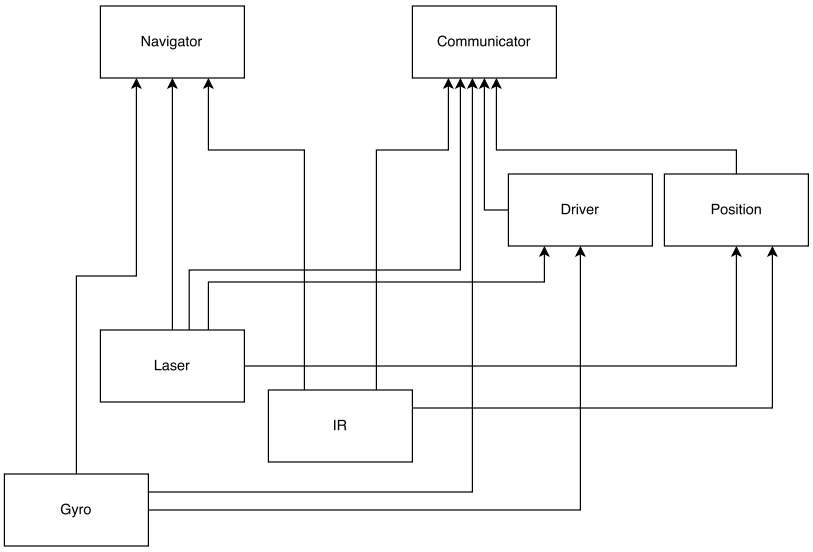
```
$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- 19 -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: 30 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- 62 -- -- -- -- -- -- -- -- 6b -- -- -- --
70: -- -- -- -- -- -- -- 77
```

Listing 3: Kommando och förväntad utdata för en fungerande I2C-buss

5 Modulerna

5.1 Huvudenhet

Huvudenheten realiseras med en Raspberry PI, på vilken operativsystemet Raspbian körs. Den representerar robotens beslutsfattande organ och sköter navigationsbeslut, kartläggning, kommunikation med mjukvaruklienten samt agerar mästare på robotens I2C-buss. Robotens logik är skriven i programmeringsspråket Python och är moduluppbyggd. En huvudloop injicerar alla beroenden i programets klasser enligt diagramet i figur 12 och uppdaterar relevanta objekt varje iteration. En beskrivning av mjukvarumodulerna finns nedan.



Figur 12: Ett diagram över logikenhetens mjukvarumoduler.

5.1.0.1 Navigator

Navigator-klassen sköter all logik för förflyttning. Givet viss indata från sensorerna fattar den beslut för att klarar av uppdraget. Se 3.1 för en beskrivning av navigeringsalgoritmen som används.

5.1.0.2 Communicator

Communicator tar emot begäran från mjukvarumodulen och innehåller all logik för att leverera korrekt svar. TODO:Team Bluetooth kanske vill knyta an det här till det de skriver om Bluetooth

5.1.0.3 Position

5.1.0.4 Driver

Driver innehåller metoder för att utföra standardiserade förflyttningar, t.ex. en inre högersväng eller en yttre vänstersväng. Vid ett anrop till en metod så ges Driver en uppgift som består av en instruktion, en jämförelsefunktion och ett värde till funktionen som Driver använder för att veta när dess uppgift är klar. Till exempel kan en uppgift (i ord) bestå av ”vänstersväng, mät i grader, 90”, vilket skulle få roboten att svänga vänster, till en skillnad i 90 grader mäts upp från gyroskopet.

5.1.0.5 IR

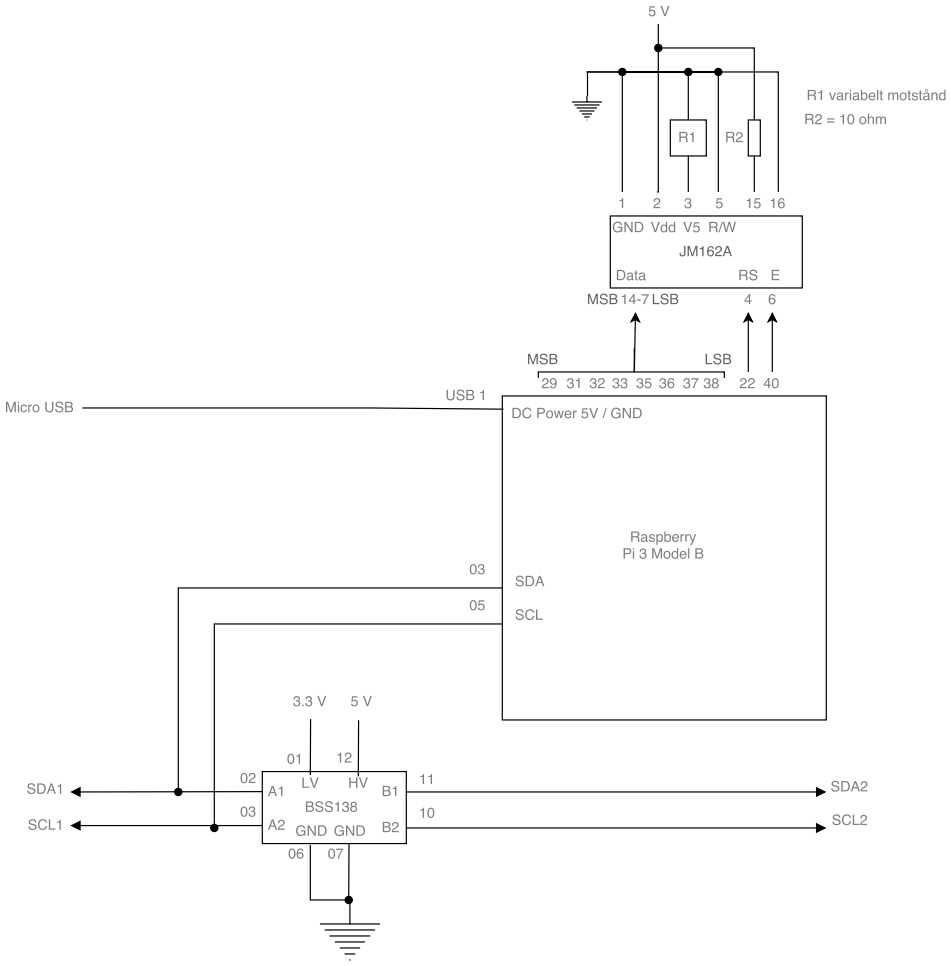
5.1.0.6 Laser

Laser-modulen är endast en Python-abstraktion för anrop till lasersensorn över I2C-bussen. Den läser av mätvärden och konverterar de till Python-heltal som sedan kan tolkas av all övrig kod.

5.1.0.7 Gyro

Gyro-modulen fyller samma funktion som Laser-modulen, och agerar mest som ett abstraktionslager.

5.1.1 Kopplingsschema



Figur 13: Kopplingsschema för huvudenheten.

Huvudenheten består av en Raspberry Pi 3 och en LCD-display av modellen JM162A. Utöver kopplingen till LCD:n så kopplas två pins till I2C-bussen. Eftersom Raspberry Pi drivs av 3.3 V matspänning och de övriga atmega-processorerna i roboten drivs av 5 V så behövs vi nivåskifta bussen så att de kan kommunicera med varandra, vilket görs med nivåskiftaren BSS138.

5.1.2 Komponenter

Komponent	Antal
Raspberry PI 3	1
JM162A	1

5.1.3 Resurser

Port	Antal	Används
GPIO pins	40	12

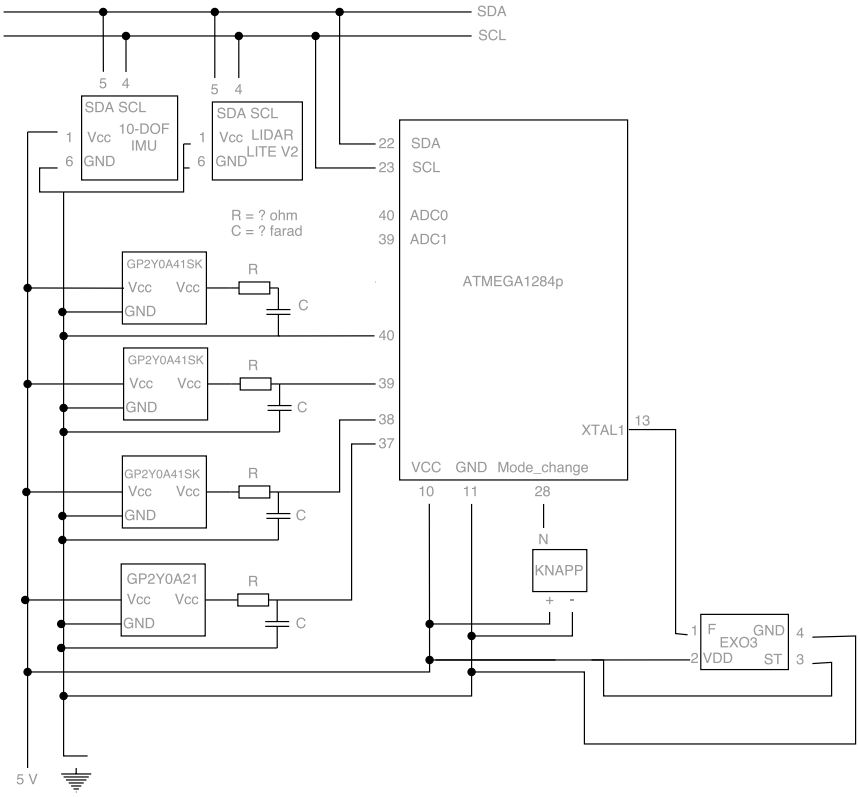
5.2 Sensorenhet

Sensorenheten har i uppgift att läsa in värden från robotens sensorer och rapportera värdena till huvudenheten. Den består av fyra IR-sensorer, ett gyroskop, en lasersensor och har sin egna beräkningsenhet, en ATMega 1284, vilket låter den arbeta asynkront från andra enheter.

Den ser på så sätt alltid till att ha data tillgänglig närhelst huvudenheten begär den. För IR-sensorerna sker det genom att processorn låter AD-omvandlaren köras för en sensor i taget och sparar sedan undan avståndet i minnet, medan gyroskop och laser håller koll på sina egna värden och är kopplade direkt på I2C-bussen. Huvudenheten kan sedan fråga sensorenheten efter ny IR-mätdata, eller fråga gyroskopet/lasern direkt varpå den får sensordatan levererad över I2C.

Avläsningarna från IR-sensorerna är analoga, och konverteras till digitalt med hjälp av ATMegans interna AD-omvandlare. För att sedan omvandla mätdatan till en approximation i millimeter används en tabell sparad i ATMegans minne.

5.2.1 Kopplingsschema



Figur 14: Kopplingsschema för sensorenheten.

Som ses i figur 14 är IR-sensorerna kopplade till ATMegan via var sitt lågpassfilter för att reducera brus. Längst upp i kopplingsschemat ses den gemensamma I2C-bussen, dit gyroskop och laser är anslutna. Notera att EXO3-komponenten är delad mellan sensorenheten och styrenheten trots att den illustreras som separata komponenter i kopplingsschemat.

5.2.2 Komponenter

Komponent	Antal
ATMega 1284	1
LIDAR-Lite v2	1
Knapp	1
GP2Y0A41SK IR-Sensor	3
GP2Y0A21 IR-Sensor	1
Adafruit 10-DOF IMU	1
EXO3	1 (delad)

Tabell 1: Tabell över de komponenter som sensorenheten består av.

5.2.3 Resurser

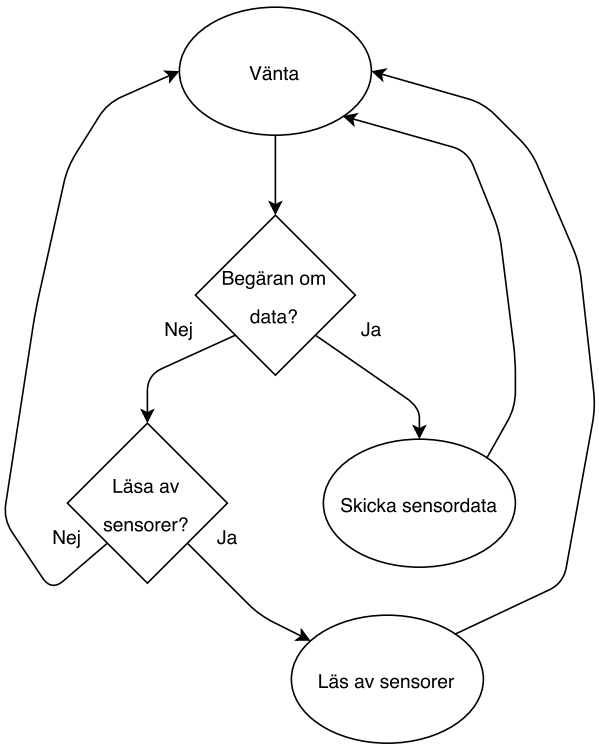
TODO:TABELLEN NEDAN EJ KORRIGERAD

Port	Antal	Krävs
SDA	1	1
SCL	1	1
PCINT	24	1
A/D	8	2
RESET	1	1
USART	4	2
JTAG	1	1
CLK	1	1

Tabell 2: Tabell över tillgängliga portar på processorn.

Den maximala hastigheten som sensordata kan läsas in i är begränsad av sensorernas rapporteringsfrekvens som är betydligt långsammare än en ATmega 1284:s klockfrekvens, därför lönar det sig inte med en snabbare processor för sensorenhetens syfte. Det i princip enda minnet sensorenheten använder sig av är för temporärlagring av alla IR-sensorers data, vilket endast motsvarar fyra heltal. Utöver sensorinläsningen kommer enheten svara på kommandon över huvudbussen som inte heller kräver några höga hastigheter eller minneskrav.

5.2.4 Programflöde



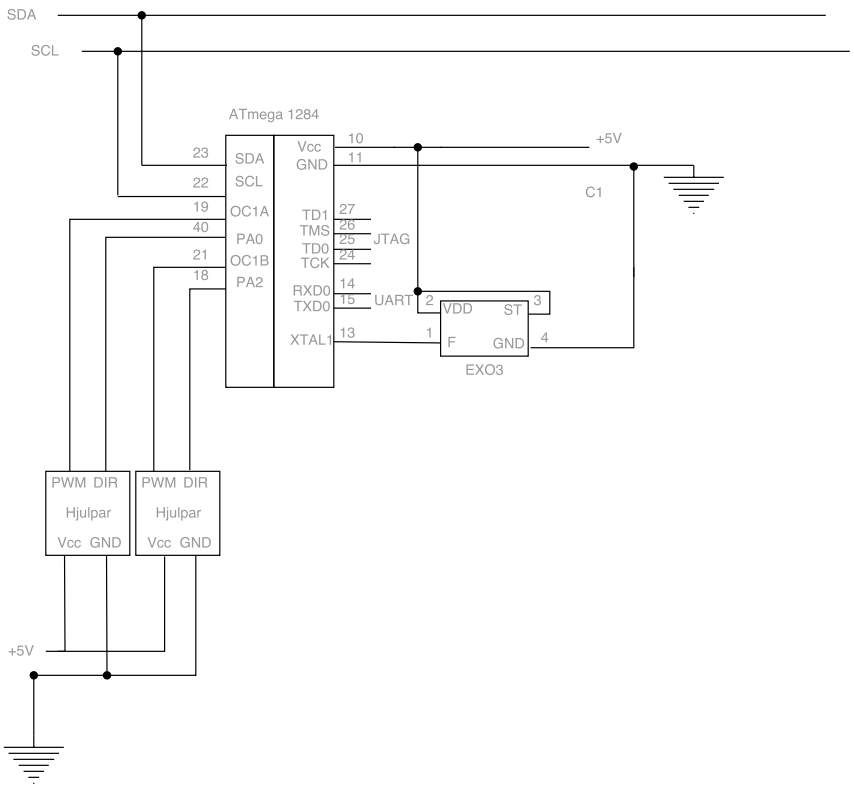
Figur 15: Ett flödesdiagram över sensorenhetens tillstånd

För att kunna tillhandahålla sensordata från IR-sensorerna när huvudenheten ber den om det så läser sensorenheten in data med jämna mellanrum och sparar den lokalt för att sedan skicka datan på kommando från huvudenheten. Om den fått en begerän om sensordata från huvudenheten så tar det prioritet att leverera den över att läsa in ny data. Ett flödesdiagram över dess beteende ses i 15.

5.3 Styrenhet

Styrenhetens uppgift är att kontrollera robotens förflyttning genom att ge korrekta signaler till dess hjulservon. Hjulen styrs parvis, det vill säga vänster hjulpar styrs med en signal och höger hjulpar med en annan. Hastigheten på hjulen styrs genom PWM (**TODO: FÖRKLARA PWM KORTFATTAT**) medan hastigheter internt är definerade som heltal. Det är styrenhetens uppgift att översätta heltalshastigheter till korrekt pulskvot.

5.3.1 Kopplingsschema



Figur 16: Kopplingsschema för styrenheten.

Styrenheten klockas av en extern EXO3-klocka som säkerställer en synkroniserad klockfrekvens bland robotens processorer, och är kopplad till de två hjulparen som sitter på robotens kropp. Likt sensorenheten så kommunicerar även styrenheten över I2C-bussen.

5.3.2 Komponenter

Komponent	Antal
ATMega 1284	1
Terminator (bas för fyrhjulingsrobot)	1
EXO3	1 (delad)

5.3.3 Resurser

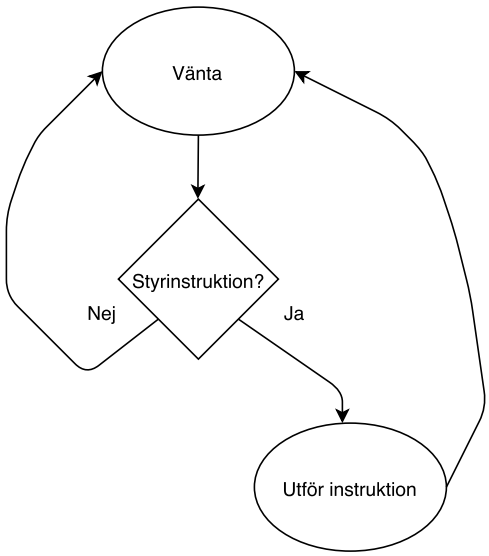
TODO:TABELL NEDAN EJ KORRIGERAD

Port	Antal	Krävs
SDA	1	1
SCL	1	1
RESET	1	1
IO	30	5
USART	4	4
JTAG	1	1
CLK	1	1

Tabell 3: Tabell över tillgängliga portar på processorn.

Omvandlingen av kommandon från huvudenheten och utskickning av signaler till servona uppskattas ej kräva mer prestanda eller minne än styrenhetens ATMega 1284 processor klarar av.

5.3.4 Programflöde



Figur 17: Ett flödesschema över styrenhetens tillstånd

Styrenheten står hela tiden och väntar på instruktioner från huvudenheten, och så snart den får några utför den dem så snabbt den kan. En instruktion i det här sammamhanget är att sätta nya hastigheter på de båda hjulparen. Ett flödesschema över styrenhetens beteende kan ses i figur 17.

5.4 Presentationsenhet

TODO: Innehåll här

5.4.1 Programflöde

TODO: Innehåll här med

5.5 Fjärrstyrningsenhet

TODO: Innehåll här också

5.5.1 Programflöde

TODO: Innehåll här med också

6 Slutsatser

6.1 Navigeringsalgoritm

Vår navigeringsalgoritm är i nuläget begränsad till att röra sig i ett rutnät och får således bara röra sig i vinklar som är multiplar av 90 grader. Begränsningen beror på att roboten inte har någon bra logik för att bestämma sin position utan att följa en vägg. Om roboten skulle kunna bestämma sin position när den navigerade i godtycklig riktning och på öppna ytor hade den totala sträckan som roboten behövt färdas minimeras.

6.2 Kartläggning

Roboten kartläggningsalgoritm kartlägger sin omgivning genom att beräkna raksträckor den åkt när den följt en vägg och spara start- och slut-koordinaterna. Om roboten kunde observera sin omgivning t.ex. med en svepande laser och beräkna var väggar fanns utifrån den datan så skulle den ej behöva färdas längs med alla väggar för att kartlägga dem. Det här förbättringsförslaget går hand i hand med förbättringarna föreslagna i avsnitt 6.1.

7 Referenser

A Banspecifikation

B Programkod