Teknisk dokumentation

Redaktör: Hannes Snögren

Version 0.1

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

HT2, 2014, Grupp 2
Linköpings Tekniska Högskola, ISY

Gruppdeltagare

Namn	**Ansvar**	**Telefon**	**E-post**
Pål Kastman	Projektledare	0703896295	palka285@student.liu.se
Hannes Snögren	Dokumentansvarig	0706265064	hansn314@student.liu.se
Alexander Yngve	Hårdvaruansvarig	0762749762	aleyn573@student.liu.se
Martin Söderén	Mjukvaruansvarig	0708163241	marso329@student.liu.se
Daniel Wassing	Leveransansvarig	0767741110	danwa223@student.liu.se
Dennis Ljung	Testansvarig	0708568148	denlj069@student.liu.se

Hemsida: http://github.com/ultralaserdeluxe/gloria

Kund: Tomas Svensson
Kontaktperson hos kund: Tomas Svensson
Kursansvarig: Tomas Svensson
Handledare: Peter Johansson

Innehåll

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2014-12-16	Första utkast	hansn314	

1 Inledning

Systemet Gloria är en lagerrobot som kan följa en bana, plocka upp paket vid särskilda stationer och sätta ned dem vid nästa tomma station.

Projektet utfördes som ett moment i kursen TSEA29 vid Linköpings Universitet under HT 2014[1]. Syftet med projektet var att ge gruppmedlemmarna övning i konstruktion och utveckling med mikrodatorer och erfarenhet i att jobba enligt en projektmodell, i det här fallet LIPS.

I detta dokument finns dokumenterat hur systemet är designat och fungerar. Syftet är dels att kunden skall kunna lösa uppkomna problem eller vidareutveckla systemet och dels att utomstående part i utbildningsyfte skall kunna förstå hur systemet fungerar.

2 Produkten

Systemet Gloria är en så kallad *lagerrobot*. Roboten kan autonomt röra sig längs en bana byggd enligt banreglerna som är specificerade i kravspecen. Roboten kan endast bära ett paket åt gången och ett paket kan endast sättas ner på en redan tom station . Om roboten upptäcker ett paket den bör plocka upp, stannar den vid stationen och övergår i ett manuellt läge för att låta användaren plocka upp paketet. När användaren signalerar att paketet är greppat övergår roboten i det autonoma läget och fortsätter följa banan till nästa tomma station där den autonomt sätter ner paketet och återigen fortsätter följa banan.

Om roboten inte bär på ett paket avslutar denne sin körning om den kommer till en stoppstation.

Bild på Gloria!

3 Systemet

Systemet består av fyra enheter. En PCenhet som styr och låter användaren styra systemet och läsa debugdata, en styrenhet som sköter drivningen av robotarm och motorer, en sensorenhet som kontinuerligt läser sensordata och en huvudenhet som tar emot kommandon från PCenheten och avgör vad roboten skall göra. Hur dessa enheter hänger ihop illustreras i figur 1.

PCenheten innefattar ett grafiskt gränssnitt där man kan se debugmeddelanden, sensordata, motorhastigheter och armens position. I det grafiska gränssnittet finns reglage för att bestämma om systemet är i sitt manuella eller autonoma läge. När systemet är i sitt manuella läge styrs det med två joysticks via det grafiska gränssnittet.

Huvudenheten innehåller robotens intelligens, den tar beslut om vad som ska göras beroende på om roboten är i sitt manuella eller autonoma läge. I det autonoma läget är det i huvudenheten all styrlogik finns som säger vart roboten är och vad den ska göra. Det är även där regleringen ligger som ser till att roboten följer banan.

Sensorenheten förser huvudenheten med sensordata.

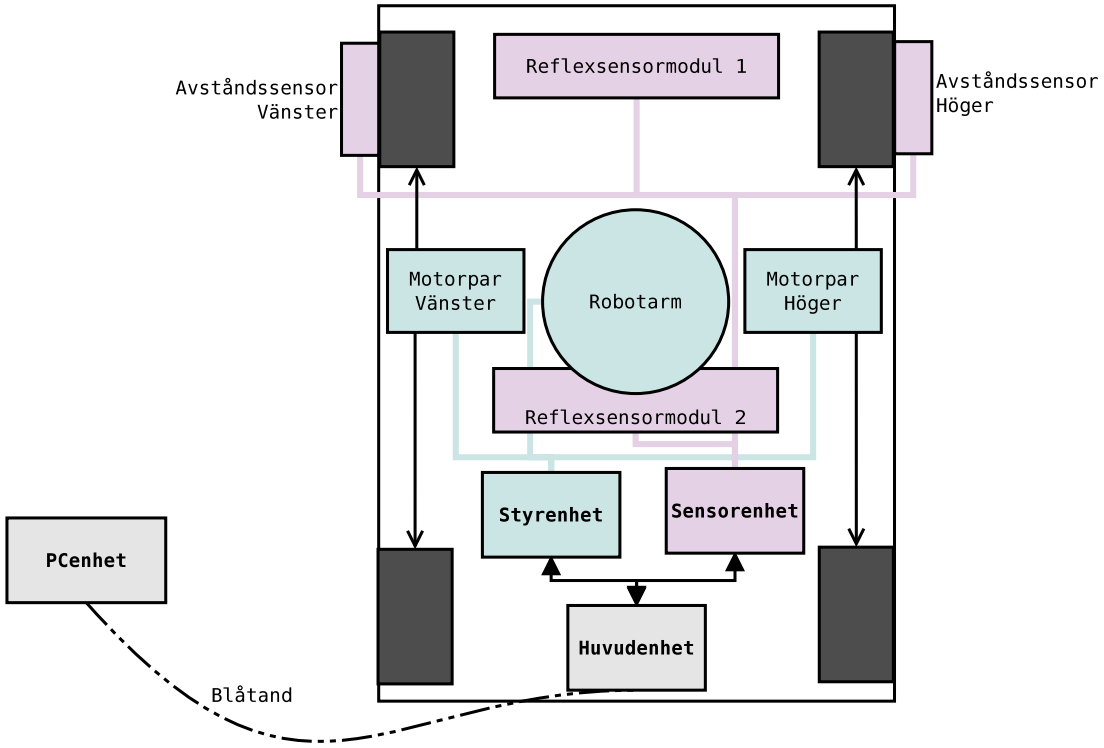
Styrenheten tar instruktioner från huvudenheten och ser till att motorer och servon utför dessa.

4 Kommunikation

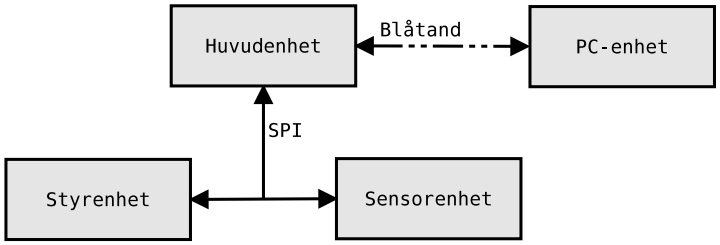
Figur 2 ger en översikt av vilka moduler som kommunicerar med varandra och på vilket sätt detta sker. Styrenheten och sensorenheten använder samma SPI-buss fast med olika Slave Select-pinnar. SPI-bussen körs på 20kHz.

4.1 PCenhet $\longleftrightarrow$ Huvudenhet

PC:n ansluter till huvudenheten över ett Personal Area Network (PAN) via Blåtand. Alternativt kan en ethernetkabel eller ett trådlöst nätverk användas för att åstadkomma samma sak. Vi sätter sedan upp en TCP/IP-anslutning för att överföra information.



Figur 1 – Översikt av systemet



Figur 2 – Översikt systemets kommunikationskanaler

Kommunikation sker på initiativ av PCenheten. PCenheten skickar ett kommando till huvudenheten, och i det fall att det är status som begärs, svarar huvudenheten. Kommandon ges i form av strängar, där en sträng kan innehålla godtyckligt många kommandon. Ett enskilt kommando ges på formen `kommando=argument1,argument2,...,argumentN`. Innehåller en sträng mer än ett kommando separeras varje kommando av ett semikolon.

När `status` begärs svarar huvudenheten på samma form som PCenheten med en sträng innehållandes relevant data separerade med semikolon. Tabell 1 listar vad och i vilken ordning huvudenheten returnerar till PCenheten.

4.1.1 Kommandon

4.2 Huvudenhet ⟷ Styrenhet

Huvudenheten kommunicerar med styrenheten över SPI. Storleken på en instruktion är antingen fyra eller sju bytes lång. För att förebygga synkroniseringsproblem skickas först två startbytes bestående av hexadecimalt `0xff` följt av längden på instruktionen (startbytesen och längdbyten räknas ej med). Därefter följer instruktionsbyten som består av två delar. De fyra högsta bitarna

Attribut	Argument	Beskrivning
lineSensor	11 heltal	Normaliserad sensordata för främre linjesensor
lineCalMax	*D*	Kalibreringsdata
lineCalMin	*D*	Kalibreringsdata
middleCalMax	*D*	Kalibreringsdata
middleCalMin	*D*	Kalibreringsdata
middleSensor	2 heltal	Normaliserad sensordata för centerad linjesensor
distance	2 heltal	Normaliserad sensordata för avståndssensorer
armPosition	X, Y, Z, P, W, G	Armens nuvarande position
errorCodes	Lista med strängar	Fel- och debugmeddelanden
latestCalibration	Datum	När sensorerna senast blev kalibrerade
autoMotor	Sant/Falskt	Huruvida roboten är i autonomt eller manuellt läge
regulator	2 heltal	Utsignalen från regulatortråden
error	Heltal	Det beräknade felet från regulatortråden
state		Vilket tillstånd roboten befinner sig i
hasPackage	Sant/Falskt	Huruvida roboten har ett paket eller inte

Tabell 1 – Figur över data som huvudenheten returnerar på ett status-kommando

Instruktion	Argument	Beskrivning
autoMotor	True/False	Sätter motorerna till autonomt eller manuellt läge
autoArm	True/False	Sätter armen i autonomt eller manuellt läge
hasPackage	True/False	Meddela roboten att vi har greppat ett paket
calibrateFloor		Kalibrera sensorer efter golv
calibrateTape		Kalibrera sensorer efter tejp
status		Returnera robotens status
start		Roboten påbörjar körning
armPosition	X,Y,Z,P,W,G	Förflytta armen till givna koordinater
motorSpeed	L,R	Sätt motorernas hastighet till givna hastigheter
halt		Roboten stannar
clearErrors		Rensa listan med debug och felmeddelanden

Tabell 2 – Kommandon från PCenhet till huvudenhet

anger vilken instruktion som ska utföras enligt tabell 3, och de fyra lägsta vilket servo eller motorpar instruktionen rör enligt tabell 4. I fallet att instruktionen är `Sätt register A till D` krävs dessutom två databytes.

Då ett motorpar adresseras anger den minsta biten i den första databyten vilken riktning motorparet skall röra sig. 0 anger framåt, 1 bakåt **Stämmer?**. Den andra databyten anger vilken hastighet vi vill att motorerna skall röra sig i. Notera att motorernas hastighet sätts med instruktionen `Sätt register A till D`.

De servon vi arbetar med har en upplösning på 10 bitar både för position och hastighet. Vi måste alltså ha två databytes när vi vill ändra någon egenskap hos dessa. De två minsta bitarna i den första databyten är de två högsta bitarna och därefter följer den andra databyten.

Tabell för att illustrera bitar hit och dit?

4.2.1 Instruktionslista

Instruktion	Argument	Beskrivning
0001	A, D	Sätt register A till D
0010	A	Utför givna kommandon för A
0011	A, D	Sätt servohastighet för A till D

Tabell 3 – Kommandon från huvudenhet till styrenhet

Adress	Beskrivning
0000	Höger hjulpar
0001	Vänster hjulpar
0010	Arm axel 1
0100	Arm axel 2
0110	Arm axel 3
1000	Arm axel 4
1011	Arm axel 5 (*gripklo*)
1100	Samtliga motorer
1101	Samtliga servon
1111	Samtliga motorer och servon

Tabell 4 – Adresser för adressering till styrenhet

4.3 Huvudenhet $\longleftrightarrow$ Sensorenhet

Huvudenheten kommunicerar med Sensorenheten över SPI. Protokollet mellan dessa moduler är väldigt primitivt då det endast finns en instruktion, `Returnera sensordata för A`. Instruktionen anges av de fyra högsta bitarna av instruktionsbyten enligt tabell 5. Huvudenhetens begäran är då bara på en enda databyte och Sensorenheten svarar omedelbart med de 8 mest signifikanta bitarna för den efterfrågade sensorn. Vilken sensor som enheten returnerar data för anges av de fyra minsta bitarna i instruktionsbyten enligt tabell 6.

4.3.1 Instruktionslista

Instruktion	Argument	Beskrivning
0000	A	Returnera sensordata för A

Tabell 5 – Instruktion från huvudenhet till sensorenhet

5 Teori

I projektet ingår främst två större teoretiska beräkningsmoment. Vi måste reglera utdatan till motorerna för att roboten skall följa linjen. Vidare vill vi kunna använda enklare koordinater för att tala om vart i rummet armen befinner sig, och vi behöver således kunna konvertera (x, y, z)-koordinater till vinklar för armens axlar.

Adress	Beskrivning
0000	Linjesensor 1
0001	Linjesensor 2
0010	Linjesensor 3
0011	Linjesensor 4
0100	Linjesensor 5
0101	Linjesensor 6
0110	Linjesensor 7
0111	Linjesensor 8
1000	Linjesensor 9
1001	Linjesensor 10
1010	Linjesensor 11
1011	Avståndssensor Höger
1100	Avståndssensor Vänster
1101	Linjesensor center Vänster
1101	Linjesensor center Höger

Tabell 6 – Adresser för instruktioner till sensorenhet

5.1 Reglering

För att följa banan måste vi reglera motorernas hastighet. Vi gör detta genom att beräkna ett fel och ändra motorernas hastighet proportionellt mot detta. Felet beräknas genom att vi i varje iteration beräknar vart tejpen befinner sig längs den främre linjesensorn. För att göra detta multiplicerar vi linjesensorvärdena med vikter, dividerar med summan av linjesensorvärdena och subtraherar sedan medianvärdet, vilket i det här fallet är 6. Vi får ekvation 1.

$$Q_{fel} = 6 - \frac{\sum_{i=1}^{11} S_n * n}{\sum_{i=1}^{11} S_n} \tag{1}$$

För att beräkna utsignalen använder vi felet Q_{fel} och förändringen jämfört med förra iterationen ΔQ_{fel} tillsammans med konstanterna P och D. Vi vet att regleringen körs med en uppdateringsfrekvens på 50Hz.

$$Q_{out} = (P * Q_{fel} + D * \Delta Q_{fel}) * freq \tag{2}$$

Q_{out} i ekvationen ovan är den utsignal som används för att beräkna motorernas nya hastigheter.

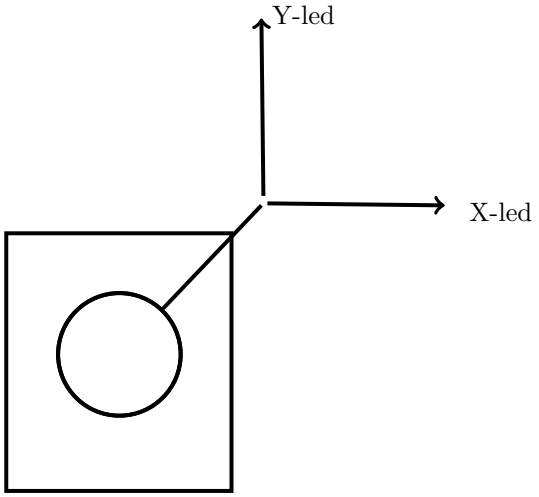
5.2 Styrning av arm

Omvandlingen från (x, y, z)-koordinater till vinklar för armens servon sker på huvudenheten. På huvudenheten finns en pythonmodul `arm.py` som implementerar beräkningarna som beskrivs nedan. Den används i tre steg.

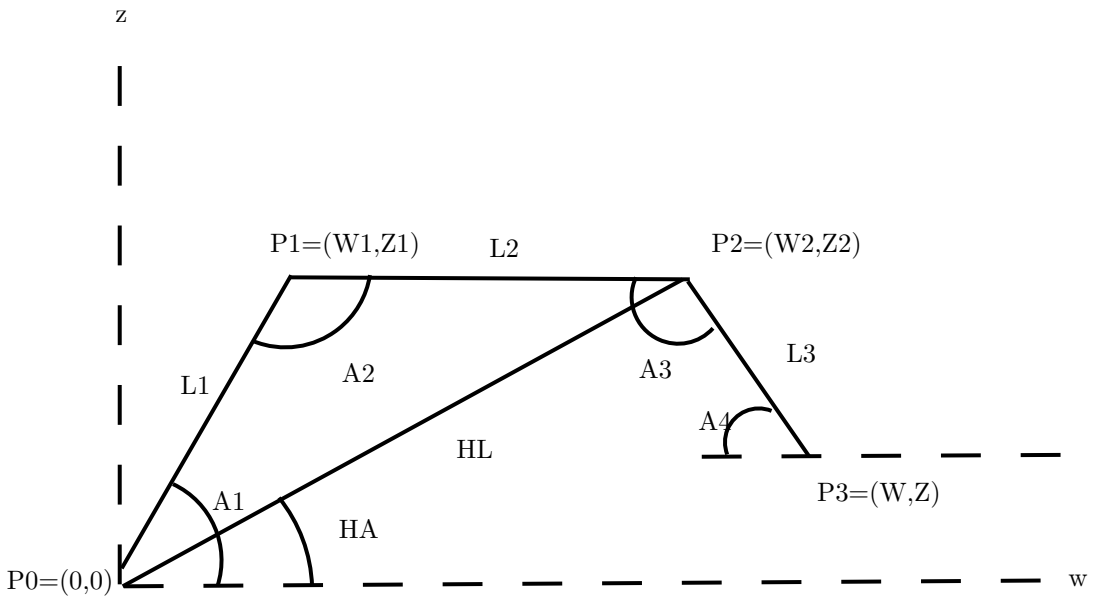
1. Skapa en instans av `robotArm` som är en klass i `arm.py`.
2. Använd funktionen `setAll` som tar in en lista som argument med följande innehåll `[x,y,z,gripperAngle,gripperRotationsOffset,gripper]`.
3. Använd funktionen `getServoValues` som returnerar en lista med alla servons vinklar i form av 10-bitars heltal.

GripperAngle är gripklons vinkel mot marken, gripperRotationsOffset är en offset på gripklons vinkel i förhållande till leden som gripklon är fäst i och gripper är hur mycket gripklon klämmer.

I listan som returneras är första värdet till armens bas, det andra är till $A1$, det tredje är till $A2$, det fjärde till $A3$, det femte till klons rotation och det sjätte till klons grepp.



Figur 3 – Armens koordinatsystem i förhållande till roboten. Upp är robotens färdriktning



Figur 4 – Armens vinklar

Omvandlingen sker på följande sätt där gripperRotationOffset och gripper ignoreras. Antag att roboten startar i $(X, Y, Z, GR) = (0, 0, 0, \pi/2)$. Där GR(GripRadian) är vilken vinkel gripklon har till z-axeln.
Vi omvandlar våra rum-koordinater till en vinkel och plankoordinater $X, Y, Z, GR \rightarrow A, W, Z, GR$. Där A är armens vinkel mot positiva x-axeln och W är armens utsträckning.

$$A = tan^{-1}(\frac{Y}{X})$$

$$W = \sqrt{X^2 + Y^2}$$

Genom denna omvandlingen så kan vi göra om detta till ett 2-dimensionellt problem istället för ett 3-dimensionellt problem. Vi får in X,Y,Z,GR och ställer in robotens vinkel mot x-axelns positiva

del och sedan ser vi till att roboten bara har rätt utsträckning och höjd.

$$P_4 = (W, Z)$$

$$P_0 = (0, 0)$$

$$W_2 = W - cos(GR) * L_3$$

$$Z_2 = Z - sin(GR) * L_3$$

$$H_L = \sqrt{W_2^2 + Z_2^2}$$

$$H_A = tan^{-1}(\frac{Z_2}{W_2})$$

Cosinussatsen ger:

$$A_1 = cos^{-1}(\frac{L_1^2 + H_L^2 - L_2^2}{2 * L_1 * H_L}) + H_A$$

$$W_1 = cos(A_1) * L_1$$

$$Z_1 = sin(A_1) * L_1$$

$$A_2 = tan^{-1}(\frac{Z_2 - Z_1}{W_2 - W_1}) - A_1$$

$$A_3 = GR - D_2 - D_3$$

Då är alla vinklar beräknade men servornas nollpunkter är inte på samma ställen som nollställena i beräkningarna ovanför. Så följande offset ställs in i grader:

- $A0 = A0 + 60$
- $A1 = 240 - A1$
- $A2 = 240 - A2$
- $A3 = 150 - A3$
- $A4 = 240 - A0 + GripperRotationOffset$ (klons rotation beror på basens rotation)
- $A5$ har ingen offset

Dessa vinklar är nu i grader. Dessa multipliceras nu med $\frac{1024}{360}$ för att omvandla grader till 10-bitars binära tal, avrundas nedåt till heltal och kan nu användas direkt av styrenheten.

6 PC-enhet

Syftet med PC-enheten är att sköta kommunikationen mellan PC och huvudenheten samt underlätta för användaren att styra systemet genom ett smart och tydligt grafiskt gränssnitt. Mjukvaran för att göra detta är uppdelad i två delar: En PC-modul som hanterar all data som skickas och tas emot samt ett grafiskt gränssnitt (GUI) mot användaren. Allt är skrivet i programspråket python.

6.1 Hårdvara

För att köra PCenheten krävs en PC med Python installerat. Under utvecklingen av systemet har Ubuntu använts och testats, och det är det som det finns instruktioner för i användarhandledningen. Vidare krävs en Blåtands-enhet för kommunikation med roboten.

6.2 PC-modul

PC-modulens uppgift är att skapa ett gränssnitt för kommunikation mellan PC och det övriga systemet. Rent kodmässigt är den strukturerad i form av en klass med olika metoder för att skicka och ta emot data från användaren och huvudenheten. Metoder som börjar på get läser olika data från huvudenheten (hastighet, armposition, debugdata) och metoder som börjar på set sätter och skickar kommando från GUI till huvudenheten. Klassens initiering tar in en sträng som är IP-adressen till huvudenheten. För att skicka data över nätverket används funktioner från det inbyggda paketet socket. Ett annat paket kallat select används för I/O verifiering.

6.3 GUI

Det grafiska gränssnittet ska hjälpa användaren att enkelt sköta och styra systemet. Det är implementerat via en egen klass som använder funktioner ur paketet tkinter för att rita upp och hantera olika grafiska objekt som fönster och knappar. Funktionerna kallar på lämpliga metoder i PC-modulen om kommando ska skickas eller data ska läsas och visas i fönstret. För att hantera joystick-indata och styrning används funktioner ur paketet pygame.

7 Huvudenhet

Huvudenheten har till uppgift att styra roboten. I huvudenheten sitter all systemets intelligens.

7.1 Hårdvara

Den hårdvara som ingår i huvudenheten består av en Beagleboard-xm. På denna sitter en Blåtandsdongel monterad för kommunikation med PCenheten. Denna är ansluten till styrenheten och sensorenheten med en flatkabel som innehåller SPI-busen.

7.2 Mjukvara

Mjukvaran är uppdelad i fyra trådar. En maintråd som har hand om all styrlogik, en PCtråd som hanterar all kommunikation med PC:n, en sensortråd som kontinuerligt uppdaterar huvudenhetens sensorvärden och en regulatortråd som har hand om regleringen så roboten kan följa banan utan problem. Alla dessa kommunicerar med varandra genom en global datastruktur som innehållar alla olika variabler som behöver delas. Kommandon från PC:n delas mellan pctråden och maintråden genom en kö.

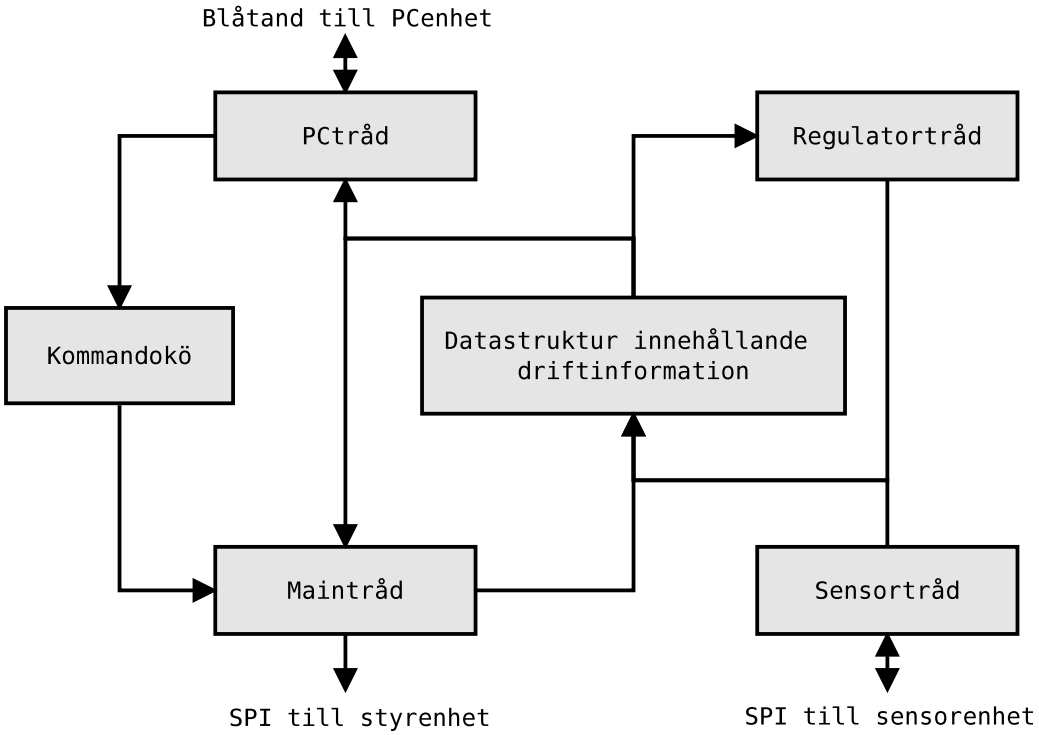
Figur 5 illustrerar dataflödet mellan huvudenhetens olika trådar. Värt att notera är att det finns en tråd dedikerad till vardera extern enhet som kommunicerar med huvudenheten.

7.2.1 Maintråd

Maintråden är den tråd som ser till att vår robot faktiskt gör något med den information och de kommandon den tar emot från andra enheter. Tråden avgör vilket tillstånd systemet befinner sig i, om roboten skall köras autonomt beroende på sensordata eller om den bara ska utföra kommandon mottagna från PCenheten.

Tillstånd

Systemet består av sex olika tillstånd, illustrerade i figur 6 och förklarade i tabell 7. Det som inte syns i figuren är att systemet kan gå till tillståndet `MANUAL` även från de tre tillstånden `STATION FRONT`, `STATION CENTER` och `STATION BOTH` och i så fall kommer systemet även återgå till detta tidigare tillstånd från `MANUAL` då `autoMotor = True`.



Figur 5 – Översikt över huvudenhetens trådar

Tillstånd	Beskrivning
`HALTED`	Autonom och manuell styrning avstängd.
`MANUAL`	Systemet styrs manuellt.
`LINE`	Systemet kör autonomt.
`STATION FRONT`	Systemet har detekterat en station vid de främre sensorerna.
`STATION CENTER`	Systemet har detekterat en station vid de centrerade sensorerna.
`STATION BOTH`	Systemet har detekterat stationer vid både de främre och de centrerade sensorerna.

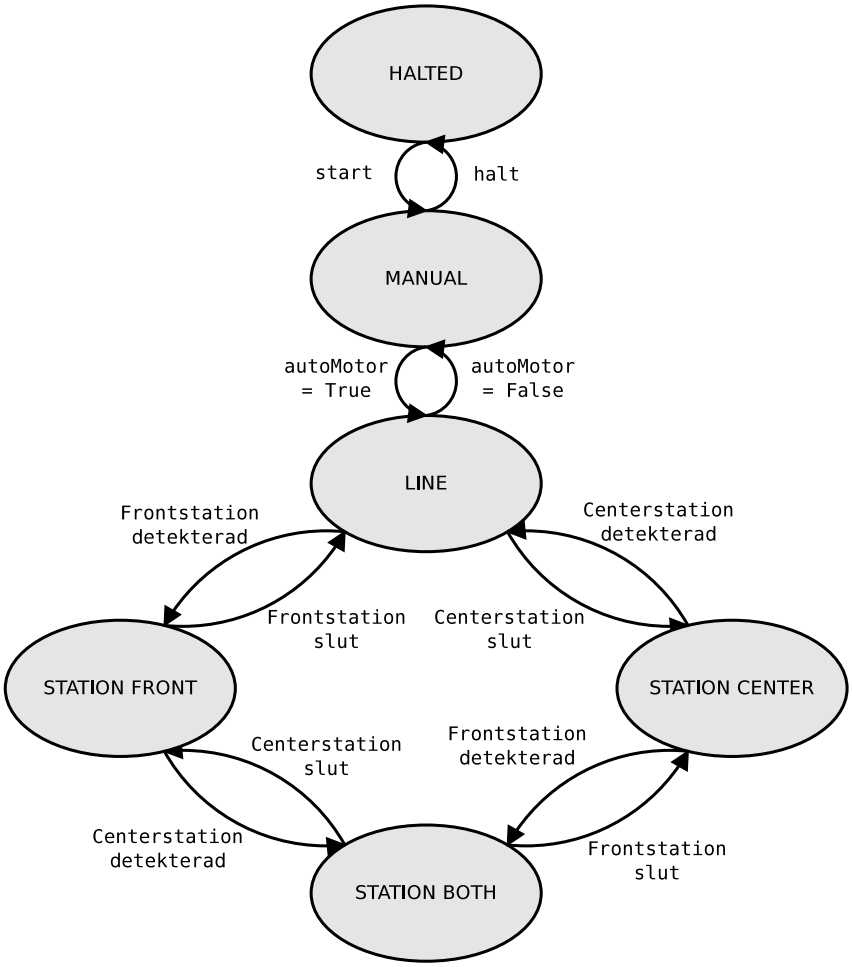
Tabell 7 – Maintrådens olika tillstånd

Detektion av stationer

Det finns många potentiella felkällor vid detektion av stationer. Vi skulle kunna få en störning på utsignalen från våra analoga sensorer, underlaget kanske inte är helt homogent och regleringen skulle kunna få oss att momentant stå snett i en korsning. För att motverka dessa felkällor har vi olika typer av filter.

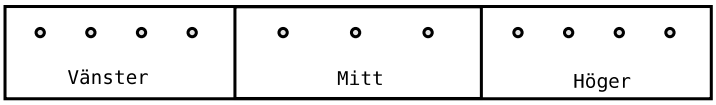
Vi filtrerar spikar från avståndssensorerna genom att vi lagrar de 10 senaste uppmätta värdena, viktar dessa med en binär skala ($D_n * 2^{-n}$, där D är uppmätta värden och n är index i listan där 0 är det senaste och 10 det äldsta uppmätta värdet) och resultatet säger vi är det uppmätta avståndet.

Det finns inte samma möjlighet att att filtrera utdata från linjesensorerna på samma sätt, då vi även kan få fel på grund av reglering eller att en tejp i banan inte är helt rak. För varje iteration av sensordata från linjesensorn har vi en rad om 11 data. Istället för att filtrera vardera av dessa 11 data för sig, accepterar vi att ett fåtal av dessa data är avvikande. Den främre linjesensorn delas upp i segment enligt figur 7 och för varje segment sätts ett krav för när detta segment kan betraktas detektera en faktisk tejp. För *Höger* och *Vänster* är kravet att tre av fyra sensorer är över tröskelvärdet. För *Mitt* är kravet endast att en sensor är över tröskelvärdet. Endast två av



Figur 6 – Översikt över maintrådens olika tillstånd

sensorerna på den centrerade linjesensorn används, den har alltså endast två segment bestående av en sensor var, *Center-höger* och *Center-vänster*.



Figur 7 – Linjesensorn är uppdelad i segment

För att inte upptäcka samma station flera gånger på grund av reglering eller sned bana och för att inte upptäcka en station i en korsning finns det även en mekanisk till. Varje iteration av sensordata sparas i en lista med de senaste sex iterationerna av sensorvärden. När vi sedan ska undersöka om det verkligen är en station vi är vid, går vi igenom alla sex iteration av sensordata. Om minst fyra av dessa rader av sensordata uppfyller kraven för en station, korsning eller ett avbrott i banan säger vi att så är fallet.

En station kännetecknas av att den ena av *Höger* och *Vänster* detekterar en tejp samtidigt som den inte gör det och vi också har tejp i *Mitt*-segmentet. Skulle alla segment detektera en tejp har vi en korsande väg i banan som skall ignoreras och skulle inget segment detektera tejp har vi ett avbrott i banan.

Vi gör på motsvarande sätt för att detektera att en station är rakt under roboten. Där kräver vi att minst en sensor på *Mitt*-segmentet på frontsensorn ska vara över tröskelvärdet i minst fyra

av sex iterationer för att försäkra oss om att vi är på en någorlunda rak linje. Vidare kräver vi att de centrerade sensorerna ska vara över tröskelvärdet i minst fyra av de sex lagrade iterationerna.

7.2.2 PCtråd

PCtråden sätter upp en socket när den skapas och väntar på en inkommande anslutning från en PC. När den får en anslutning så väntar den på ett kommando.

Oftast tas flera kommandon emot samtidigt och PCtråden gör om dessa till en lista av kommandon som den går igenom och behandlar ett efter ett. Får man till exempel in två kommandon om sätta motorhastigheten så ser tillvägagångssättet ut på följande sätt:

1. ";motorSpeed=speed1,speed2;motorspeed=speed3;speed4;tas emot
2. detta görs om till [[motorspeed",[speed1",speed2"]],[motorSpeed",[speed3",speed4"]]]
3. det första kommandot behandlas genom att argumenten omvandlas till heltal, motorernas hastighet uppdateras i dictionaryt och kommandot läggs till i kön så maintråden vet att den ska skicka ut kommandot.
4. nästa kommando behandlas på samma sätt men om speed1=speed3 och speed2=speed4 så ignoreras kommandot då det inte påverkar hastigheterna.

7.2.3 Sensortråd

Sensortråden hämtar data från Sensorenheten med en given uppdateringsfrekvens. De hämtade sensorvärdena normaliseras och lagras i den globala datastrukturen. Normaliseringen utförs på så sätt att sensordatan jämförs med de värden vi fått genom kalibrering, som i ekvation 3. Den normaliserade sensordatan är alltså hur stor del av det använda intervallet som används.

$$D_{norm} = \frac{D_{in} - D_{min}}{D_{max} - D_{min}} \tag{3}$$

7.2.4 Regulatortråd

Regulatortråden använder data från linjesensorerna för att beräkna hur roboten behöver röra sig för att följa linjen. Detta görs sedan om till motsvarande motorhastigheter och lagras i den globala datastrukturen.

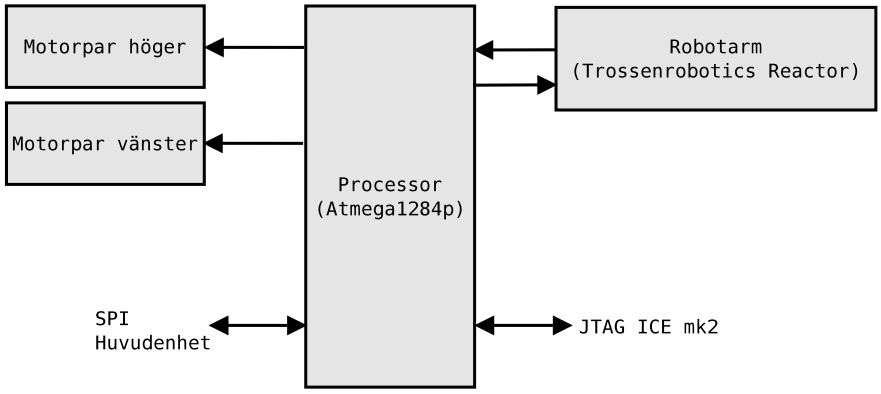
Regleringen utförs i en egen tråd för att säkerhetsställa att beräkningarna sker med jämna tidsintervall. Det är dock värt att notera att detta tidsintervall är beroende av det tidsintervall med vilket sensortråden uppdaterar sensordata eftersom det är meningslöst att reglera på gamla sensorvärden.

8 Styrenhet

Styrenheten har till uppgift att ta kommandon från huvudenheten och producera utsignaler som motorer och servon kan förstå.

8.1 Hårdvara

Styrenheten består av en Atmega1284p, en tri-state-krets[12], två motorpar och en robotarm Trossenrobotics Reactor[4]. Styrenheten är ansluten till huvudenheten via samma 20-pinnars flatkabel som sensorenheten. Figur 8 illustrerar grovt hur de är ihopkopplade. Bilaga A visar mer detaljerat hur det är kopplat.



Figur 8 – Översikt av styrenheten

8.1.1 Motorer

Systemet har sammanlagt fyra motorer, uppdelat i två motorpar. De är anslutna till Styrenheten med en 10-pinnars flatkabel. Motorparen levererades med intelligenta H-bryggor[6] och behöver förses med en PWM-signal var, som bestämmer motorernas hastighet och en riktningssignal var där 1 är i framriktningen och 0 backriktningen.

8.1.2 Robotarm

Robotarmen är en Trossenrobotics Reactor och består av åtta servon av typ AX-12a[5]. Dessa är fördelade på fem axlar och en gripklo. De kommunicerar med Styrenheten över UART med en baudrate på 1MB/s, där både in och utsignal går via samma sladd. Denna är därför ansluten till två tristates för att förebygga att något oväntat skrivs eller läses från bussen.

8.1.3 Processor

Styrenheten drivs av en enkretsdator av typ Atmega1284p. Den kan anslutas till en PC via JTAG för att programmeras eller felsökas. Den har en klockfrekvens på 16MHz och tar emot kommandon från Huvudenheten, tolkar dessa och ser till att motorerna och robotarmen utför det som Huvudenheten har sagt. Figur 9 visar hur enkretsdatorn är ansluten till övrig hårdvara.

8.2 Mjukvara

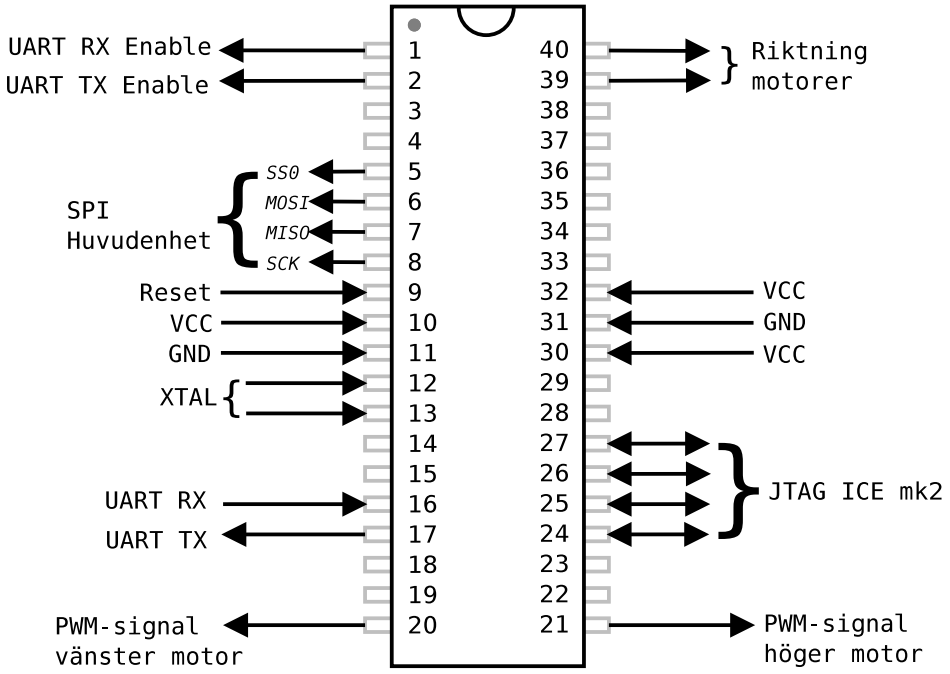
Styrenhetens mjukvara består av två moment. För det första tas kommandon emot via SPI från huvudenheten. Vi utnyttjar att Atmega1284p har hårdvarustöd för SPI och triggar ett interrupt när en byte tas emot.

Centralt i mjukvaran är den kommandokö där mottagna databytes hamnar. Vi lägger alltid till inkomna databytes till det sista kommando-objektet i kön. Om vi får in en databyte och det sista objektet är fullt eller listan tom, skapas ett nytt kommando-objekt och läggs till kön. Vi vet att ett kommando-objekt är fullt genom att jämföra den förväntade längden (se 4.2) med hur många byte kommandot består av.

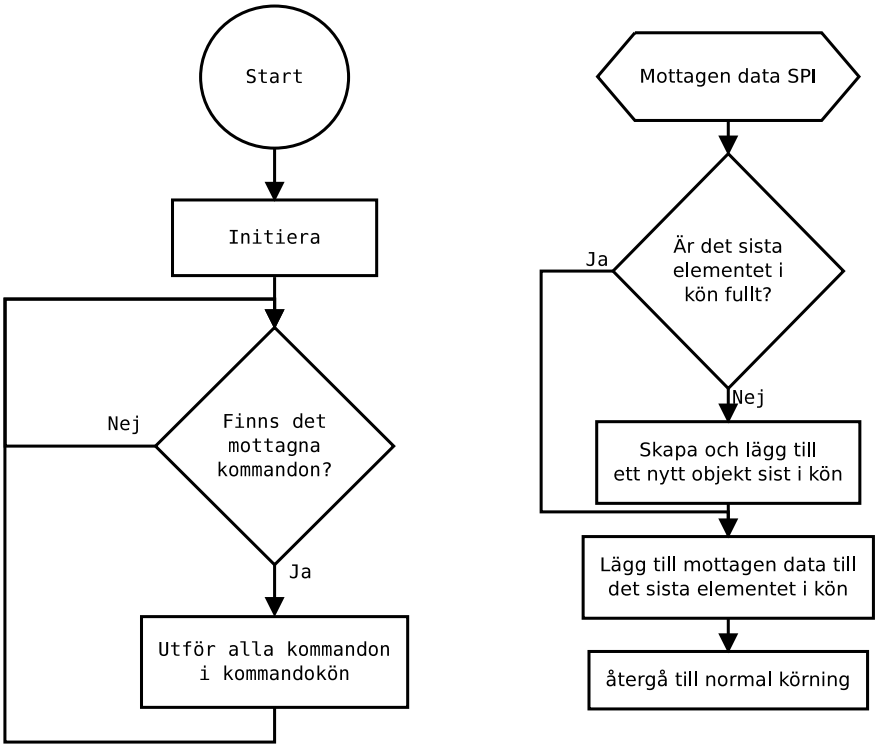
På andra sidan kommandokön har vi den del av mjukvaran som tolkar och utför inkomna kommandon. Den kollar kontinuerligt om kommandokön är tom. Är den inte det kollar vi om det första kommandoobjektet är fullt. Är det fullt mottaget tolkar vi och utför det.

Styrenheten kommunicerar med robotarmen över UART. Atmega1284p har hårdvarustöd för USART, vilket används genom att vi lägger till data till en buffert och väntar på att det skickas.

Motorernashastighet kontrolleras genom en PWM-signal till vardera motorpar. Vi utnyttjar att Atmega1284p har hårdvarustöd för två PWM-signaler som styrs av en timer var.



Figur 9 – Schema över hur enkretssdatorn i styrenheten är ansluten till övrig hårdvara.



Figur 10 – T.v. Styrenhetens mainloop. T.h. Hur styrenheten hanterar avbrott över SPI.

Styrenheten har en intern datatyp för typ av enhet den förväntas styra. Om det mottagna kommandot är ett av typen `Sätt register A till D` lagras detta i den adresserade hårdvarans datatyp. I det fall att det är armen som adresseras skickas den mottagna datan till rätt register

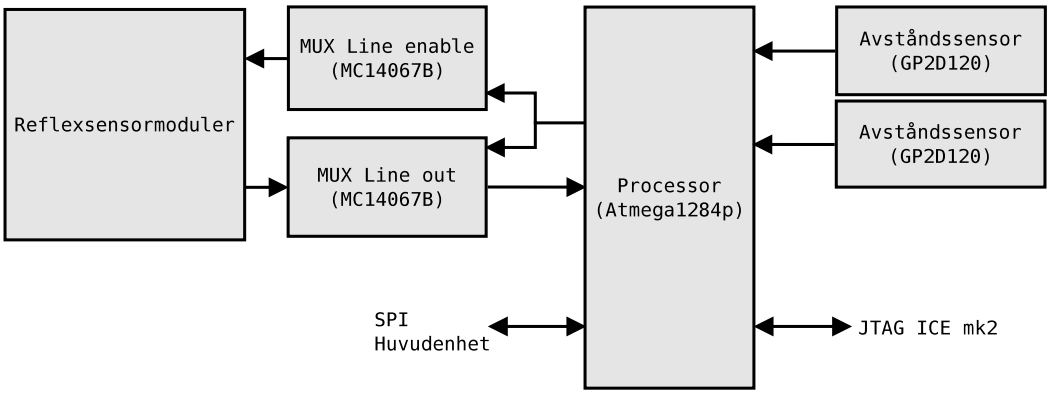
med ett `WRITE`-kommando [5]. Det innebär att armen inte realiserar de mottagna värden förrän den får ett `ACTION`-kommando. När styrenheten mottager ett `Action`-kommando från huvudenheten realiserar den tidigare mottagna kommandon för den adresserade hårdvaran. Detta görs genom att skicka `ACTION`-kommandon till armen respektive sätter de timers som styr PWM-signalerna till motorparen.

9 Sensorenhet

Sensorenheten har till uppgift att förse huvudenheten med sensordata. Sensordatan den returnerar är obehandlad rådata.

9.1 Hårdvara

Sensorenheten består av en Atmega1284p, två reflexsensormoduler och två avståndssensorer. Reflexsensormodulerna är kopplade via två 16-1 muxar till enkretsdatorn. Den ena muxen används för att styra en konstant hög enable-signal till rätt reflexsensor och den andra för att välja rätt utsignal. Båda muxarna styrs av samma styrsignal från enkretsdatorn. Avståndssensorerna är kopplade direkt till enkretsdatorn. Enkretsdatorn är ansluten till huvudenheten med en 20-pinnars flatkabel över vilken de kommunicerar via SPI. Figur 11 illustrerar hur enheten är organiserad i grova drag. Bilaga A visar mer detaljerat hur det är kopplat.



Figur 11 – Översikt av sensorenheten

9.1.1 Reflexsensormodul

Reflexsensormodulerna består av 11 reflexsensorer var. De ansluts med en 22-pinnars flatkabel vardera där respektive kabel har en enable- och en utsignal för vardera reflexsensor enligt databladet[10]. De matas med 5V och ger en analog insignal mellan 0V och 5V beroende på hur mycket ljus som reflekteras. De ger låg utspänning då underlaget reflekterar mycket ljus, och hög utspänning då lite ljus reflekteras.

Den ena reflexsensormodulen används för att detektera vart på banan vi är för att kunna reglera styrningen, endast de två yttersta sensorerna används på den andra modulen för att detektera när en uppplocknings-/nedsättningsstation befinner sig mitt under roboten, de övriga sensorerna används inte.

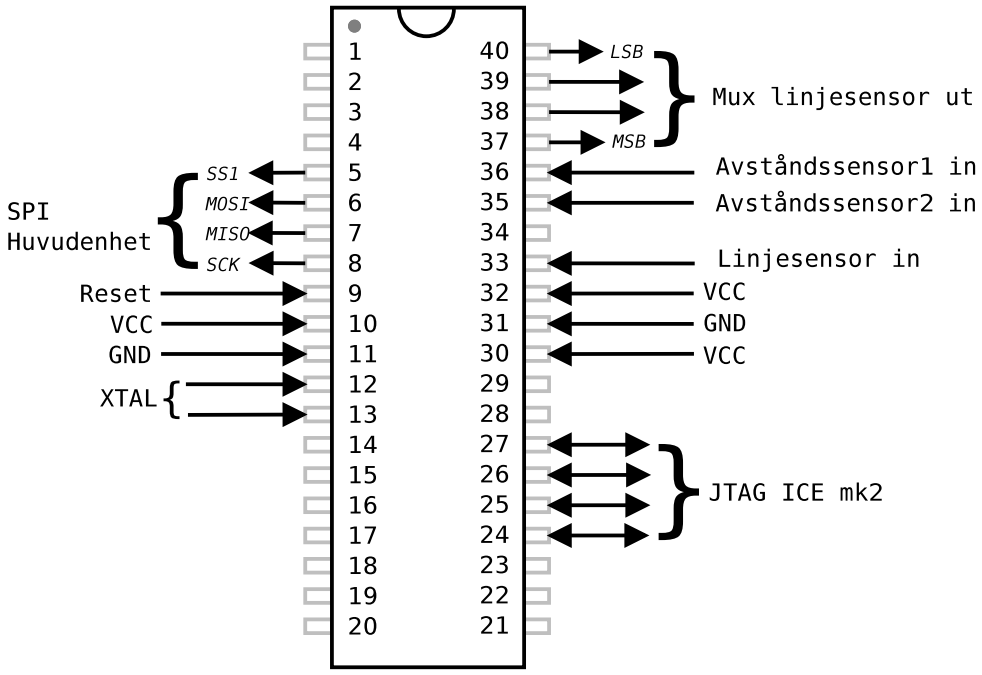
9.1.2 Avståndssensorer

Avståndssensorerna är av typen GP2D120 vilken använder ljus för att detektera avståndet. Dess utsignal är en analog spänning mellan 0V och 3.2V som går hög ju närmare ett objekt befinner sig.

Avståndssensorerna sitter på vardera sida av roboten och används för att detektera huruvida det står ett paket vid en station eller inte.

9.1.3 Processor

Den centrala komponenten i Sensorenheten är enkretsdatorn Atmega1284p. Den kör en klockfrekvens på 16MHz och utför den faktiska läsningen av sensorerna och vidarebefodrar dessa värden till Huvudenheten. Figur 12 visar hur de olika enheterna är kopplade till enkretsdatorn.



Figur 12 – Schema över hur enkretssdatorn i sensorenheten är ansluten till övrig hårdvara.

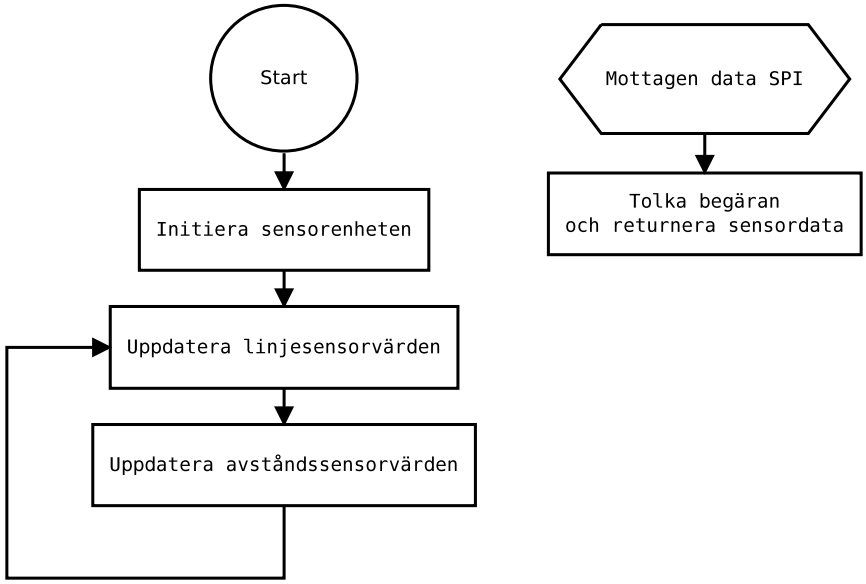
9.2 Mjukvara

All mjukvara på sensorenheten är skriven i *C* och finns på enkretsdatorn. Den består av två delar, illustrerat i figur 13.

För det första körs en mainloop där sensordata kontinuerligt uppdateras. Först itererar vi igenom Reflexsensorerna genom att först styra om enable-signalen till den aktuella sensorn och därefter utföra en AD-omvandling på den signal vi får tillbaks och sedan gå till nästa. Därefter utför vi i tur och ordning en AD-omvandling på insignalerna från avståndssensorerna. Alla värden sparas i en global datastruktur.

För det andra tar sensorenheten emot begäran om sensordata från huvudenheten över SPI. När en sådan förfrågan inkommer triggas ett avbrott i vilket sensorenheten svarar med det aktuella sensorvärdet. Då inget i avläsningen av sensorerna är tidskritiskt behöver vi inte oroa oss för när dessa avbrott kommer.

Då Atmega1284p har hårdvarustöd för SPI på så sätt att den triggar ett interrupt då den har mottagit en byte. Det är i detta avbrott vi utför ovan nämnda procedur för kommunikation med huvudenheten.



Figur 13 – Till vänster sensorenhetens mainloop, till höger den avbrottsrutin som körs vid mottagen data över SPI.

10 Slutsatser

Vi har byggt en robot som kan utföra en rad uppgifter både autonomt och manuellt. Även om systemet fungerar felfritt så skulle många förbättrningar kunna göras.

Mjukvarumässigt kan en hel del kod optimeras för att systemet ska flyta på bättre, speciellt de delar där styrning av armen sker. På

beagleboarden återfinnes också kod för detta som kan göras bättre. Filtreringsfunktioner för sensorvärdena kan förbättras för att undvika felaktiga värden. All kod kan struktureras bättre för ökad läsbarhet.

Ur ett hårdvarumässigt perspektiv skulle fler och mer precisa sensorer (mer specifikt avståndssensorerna) kunna förbättra systemet.

Om vi hade haft mer tid hade vi önskat utöka funktionaliteten hos roboten. En autonom upplockning av paket hade varit önskvärd och hade gjort robotens autonoma läge helt komplett.

Som sagt fungerar systemet felfritt men med ovan nämnda förbättringar i mjukvaran samt hårdvaran kan produkten bli ännu mer attraktiv på marknaden.

Referenser

[1] http://www.isy.liu.se/edu/kurs/TSEA29/, information hämtad 2014-09-**tidigt?**.

[2] http://www.commsys.isy.liu.se/sv/student/kurser/TATA62/lips, information hämtad 2014-09-25.

[3] LIPS, Studentlitteratur, Tomas Svensson, Christian Krysander **Gör rätt**

[4] http://www.trossenrobotics.com/p/phantomx-ax-12-reactor-robot-arm.aspx, information hämtad **hittepå**.

[5] http://support.robotis.com/en/techsupport_eng.htm#product/dynamixel/ax_series/dxl_ax_actuator.htm, information hämtad 2014-10-24.

[6] https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/pwm-motorstyrning.pdf, information hämtad **hittepå**

[7] http://www.atmel.com/images/doc8059.pdf, information hämtad **hittepådatum**.

[8] http://www.sharpsma.com/webfm_send/1205, information hämtad **Hittepådatum**.

[9] http://www.ubuntu.com, **Hur refererar vi ubuntu?**

[10] https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/reflexsensormodul.pdf, information hämtad **hittepådatum**.

[11] https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/4067b.pdf, information hämtad **hittepådatum**.

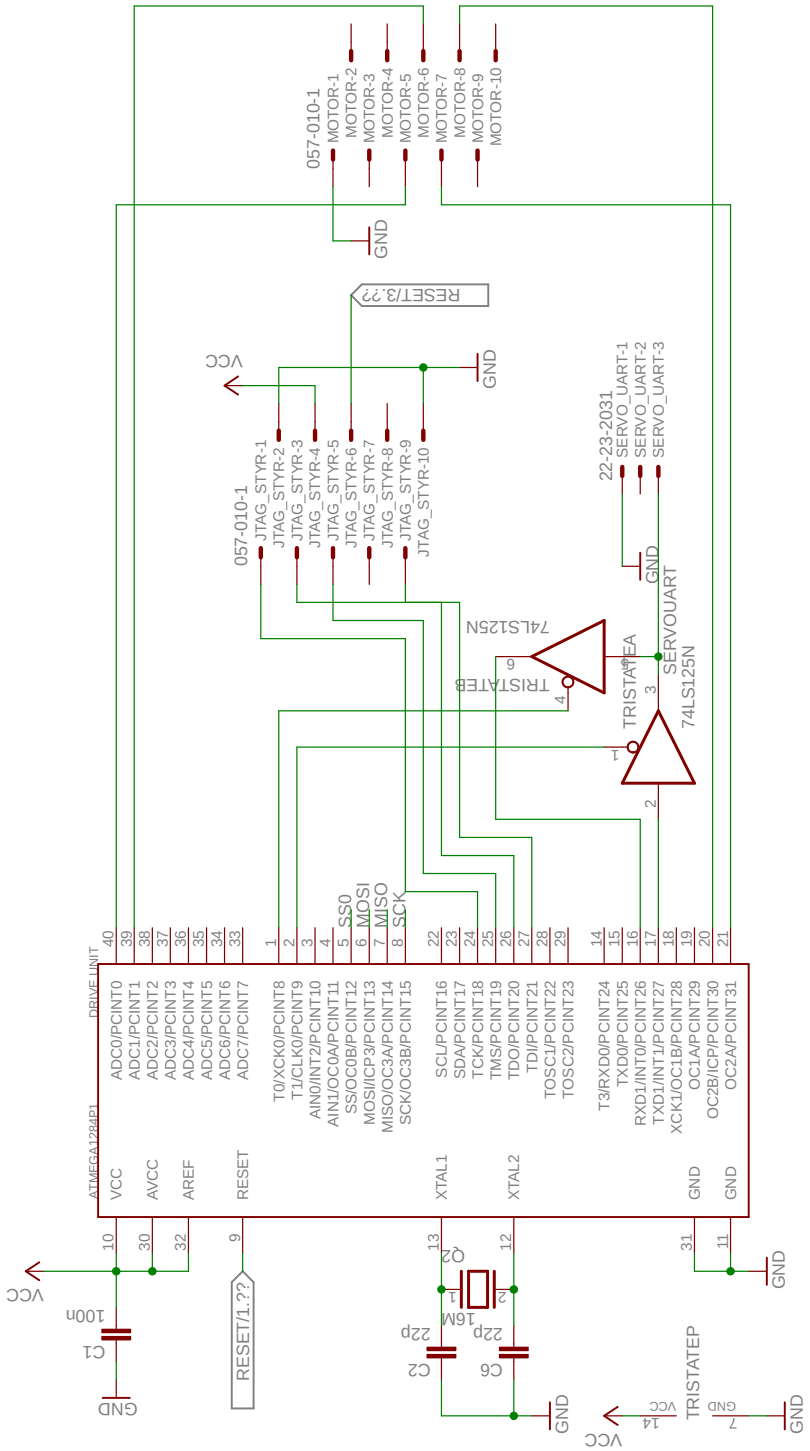
[12] https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/sn74ls125.pdf, information hämtad **hittepådatum**.

[13] http://www.ti.com/lit/ds/symlink/txb0104.pdf, information hämtad **hittepådatum**.

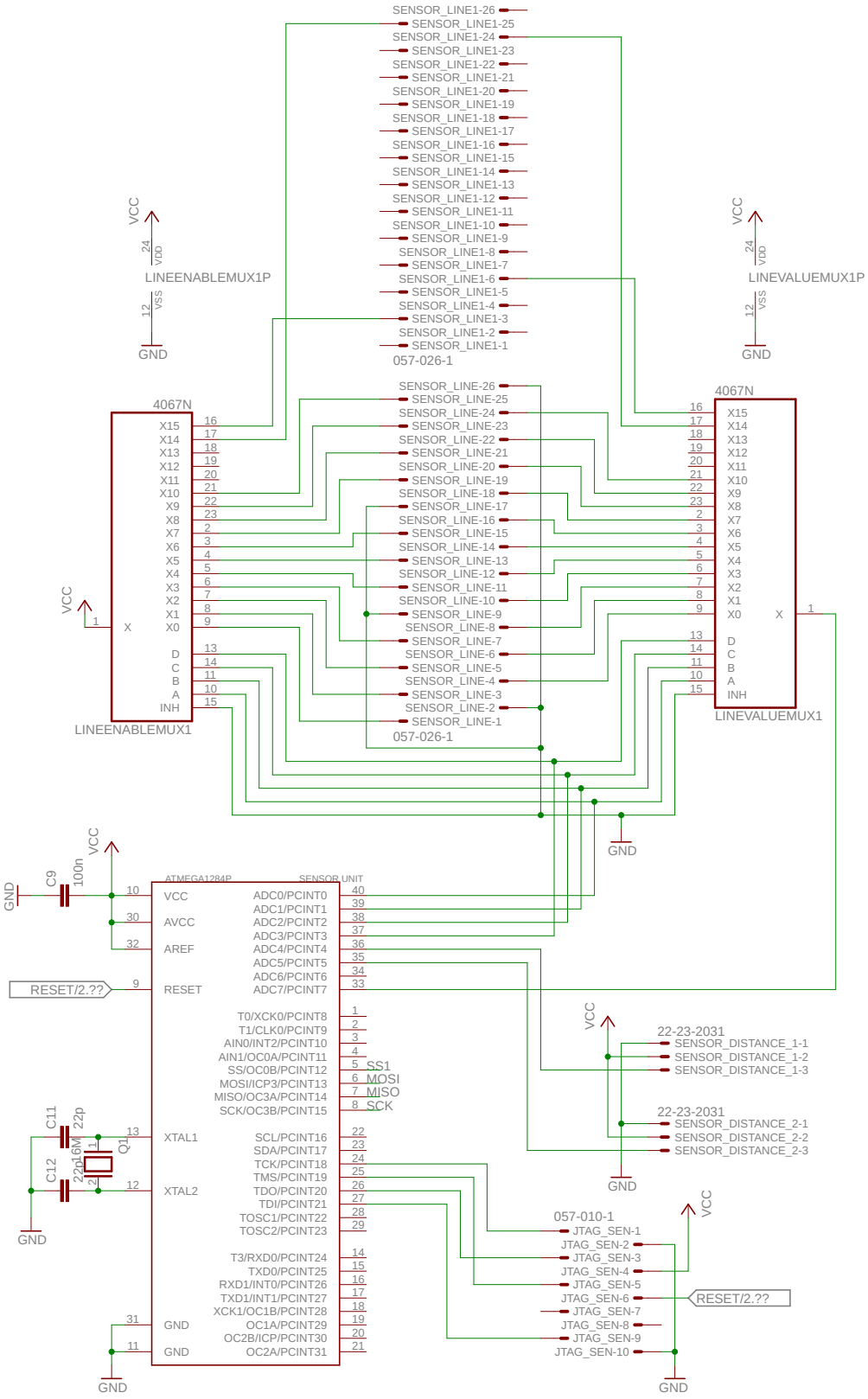
Resurs/datablad för Beagleboard?
Datablad för kristall/oscillator
Datablad för tryckknapp

Bilaga A Kopplingsschema

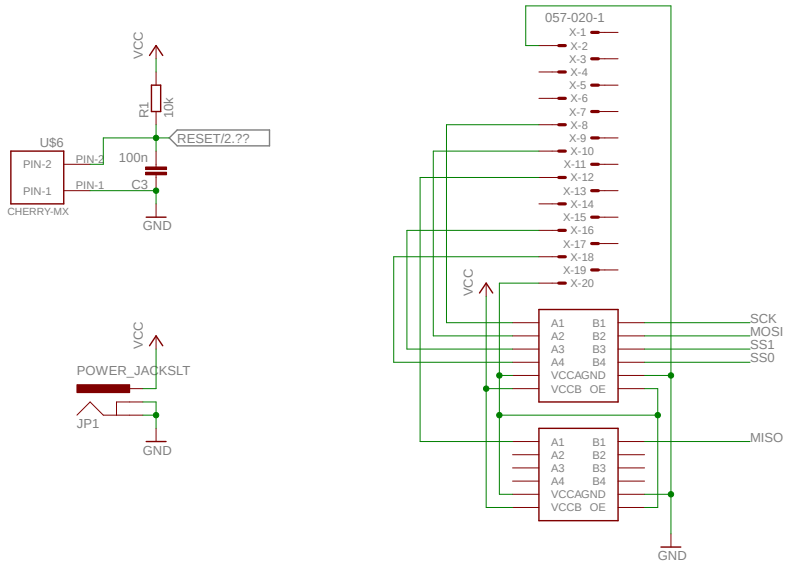
På nästkommande sidor följer de kopplingsscheman som beskriver hur hårdvaran i Gloria hänger ihop.



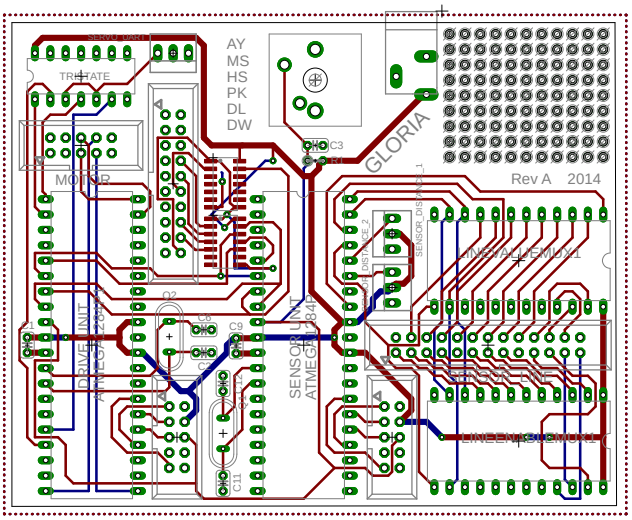
Figur 14 – Kopplingsschema styrenhet



Figur 15 – Kopplingsschema sensorenhet



Figur 16 – Kopplingsschema level-shifters och reset-knapp



Figur 17 – Hur PCBn ser ut