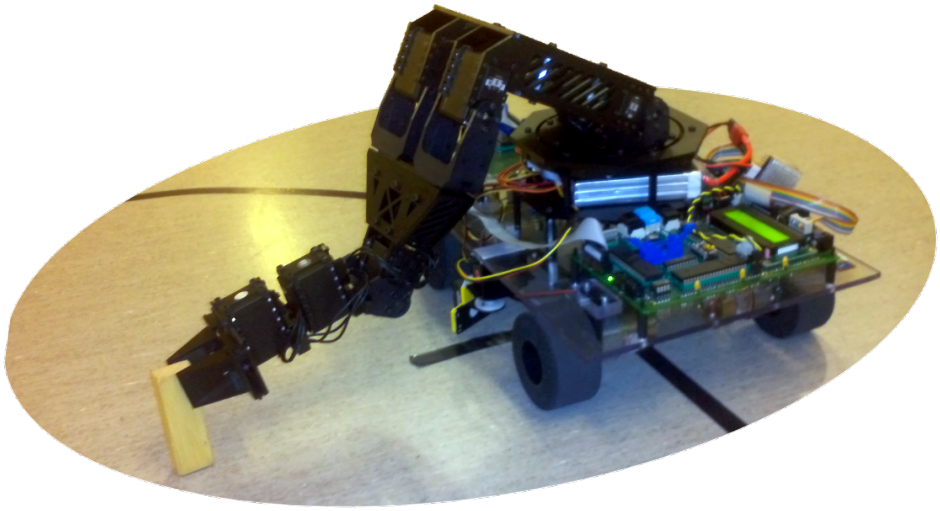
Teknisk dokumentation
Lagerrobot



Redaktör: Karl Linderhed

Version 1.0

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

Grupp 1, VT14, Lagerrobot
Linköpings tekniska högskola, ISY

Namn	**Ansvar**	**Telefon**	**E-post**
Karl Linderhed	Projektledare (PL)	073-679 59 59	karli315@student.liu.se
Patrik Nyberg	Dokumentansvarig (DOK)	073 -049 59 90	patny205@student.liu.se
Johan Lind		070-897 58 24	johli887@student.liu.se
Erik Nybom		070-022 47 85	eriny778@student.liu.se
Andreas Runfalk		070-564 23 79	andru411@student.liu.se
Philip Nilsson		073-528 48 86	phini326@student.liu.se
Lucas Nilsson		073-059 42 94	lucni395@student.liu.se

E-postlista för hela gruppen: tsea56-2014-grupp-1@googlegroups.com
Kontaktperson hos kund: Tomas Svensson, 013-28 13 68, tomass@isy.liu.se
Kursansvarig: Tomas Svensson, 013-28 13 68, 3B:528, tomass@isy.liu.se
Handledare: Anders Nilsson, 3B:512, 013-28 26 35, anders.p.nilsson@liu.se

Innehåll

1 Inledning **1**

2 Produktbeskrivning **1**

3 Systemöversikt **1**
3.1 Delsystemsöversikt . . . 2
3.2 Sensorer och sensorplacering . . . 3

4 Gemensamma funktioner **3**
4.1 Protokoll för seriell kommunikation mellan robot och PC . . . 3
4.2 USART . . . 4
4.3 Intern buss . . . 5
4.3.1 Protokoll . . . 5
4.3.2 Implementation . . . 5
4.4 Utmatning på LCD-skärm . . . 6

5 Kommunikationsenhet **7**
5.1 LCD-gränssnitt . . . 7
5.2 Seriell kommunikation över blåtand . . . 9

6 Sensorenhet **10**
6.1 Sensorer . . . 10
6.1.1 Avståndssensorer . . . 10
6.1.2 Linjesensor . . . 11
6.1.3 RFID-läsare . . . 13
6.2 Sidoskanner . . . 13
6.2.1 Koordinatberäkning . . . 14
6.3 Linjeföljning . . . 15
6.3.1 Korsning, avbrott och plockstationsdetektering . . . 15
6.3.2 Tyngdpunktsberäkning . . . 15

7 Chassienhet **16**
7.1 Funktion . . . 17
7.2 Översiktlig beskrivning av programmet . . . 17
7.2.1 Följa linje samt motorkontroll . . . 18
7.2.2 Identifiering av plockstationer . . . 19
7.2.3 Styrkommando . . . 19
7.2.4 Autonomt läge och start-/stoppkommando . . . 19

8 Armenhet **20**
8.1 Funktion . . . 20
8.1.1 Manuellt läge . . . 21
8.1.2 Autonomt läge . . . 21
8.1.3 Inverterad kinematik . . . 21
8.2 Översiktlig beskrivning av programmet . . . 23

9 Programvara för persondator **24**

Referenser **25**

A Kopplingsschema **26**
A.1 Sensorenhet 26
A.2 Kommunikationsenhet, Armenhet, Chassienhet 27

B Utdrag från programlistning **28**

C ID:n till buss **38**
C.1 Delsystem sensor 38
C.2 Delsystem chassi 39
C.3 Delsystem kommunikationsenhet 40
C.4 Delsystem arm 41

D Protokoll för blåtand **42**

Dokumenthistorik

Version	**Datum**	**Utförda förändringar**	**Utförda av**	**Granskad**
1.0	2014-05-26	Första version.	Alla	Alla

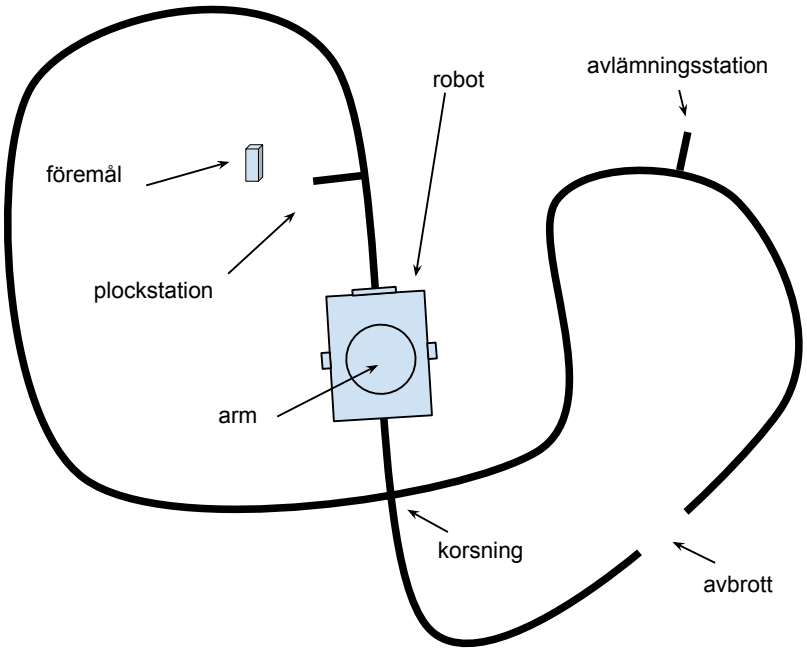
1 Inledning

Vi har i vårat kandidatarbetsprojekt konstruerat en robot. Detta dokument tjänar till att vara ett underlag för någon som vill konstruera en liknande produkt och beskriver i detalj hur systemet fungerar.

2 Produktbeskrivning

Produkten är en robot bestående av ett chassi av plexiglas med fyra hjul, sensorer, en arm och styrelektronik. Utöver detta finns programvara för övervakning och manövrering av roboten.

Roboten är tänkt att användas i ett lager där den autonomt eller fjärrstyrt hämtar och lämnar lagervaror. Lagerarbetet utförs genom att följa en svart linje längs marken där stationer är markerade med ett svart sträck, vinkelrätt mot banan, i den riktning stationen befinner sig. Vid stationerna finns en RFID-tagg för identifiering av stationstypen, upplockning eller avlämning.

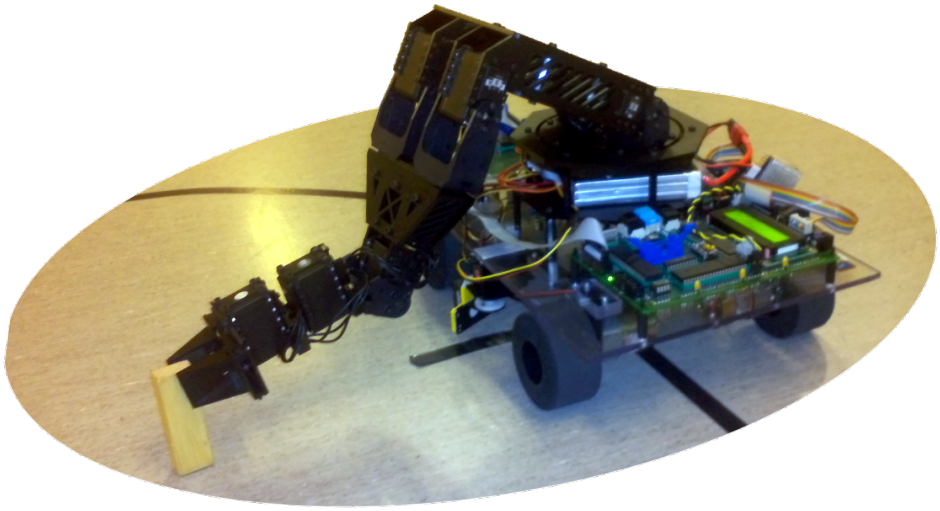


Figur 1: Översiktlig bild av bana med robot

3 Systemöversikt

Roboten är uppbyggd av fyra olika delsystem kallade chassi, sensor, arm och kommunikation där var och en av dem har sin specifika uppgift att sköta. Förutom robotens fyra delsystem har programvara för persondatorer utvecklats för övervakning och styrning. Roboten kan både

operera i ett fjärrstyrt, fullständigt manuellt, läge och i ett autonomt läge där möjligheten att styra vissa delar manuellt fortfarande ges.



Figur 2: Den färdiga roboten håller på att plocka upp ett föremål i ett ”lagerutrymme”.

3.1 Delsystemsöversikt

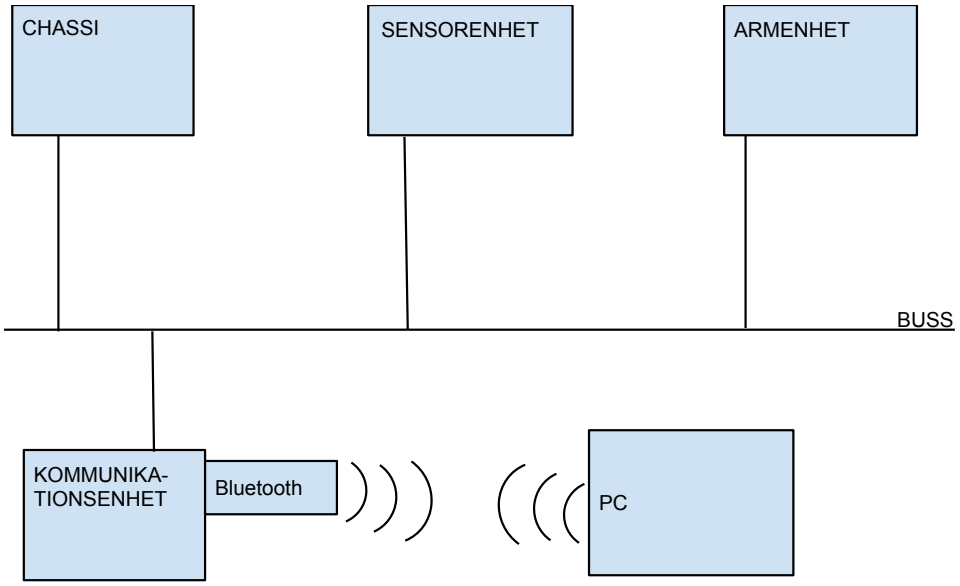
Sensorenheten övervakar kontinuerligt de för tillfället aktuella sensorerna och anpassar sensordata för tolkning av övriga delsystem.

Kommunikationsenheten fungerar som en förbindelselänk mellan roboten och omvärlden. Den hanterar kommunikationen mellan roboten och datorprogramvaran. Den förmedlar order, robotens status och sensordata. Vidare är kommunikationsenheten utrustad med en skärm som visar information från olika delsystem.

Chassienheten fungerar som det beslutande organet på roboten. Den vet vid autonomt läge hur roboten ska agera i olika situationer. Chassienheten samverkar med sensorenheten och ser till att roboten följer linjen, genom reglering av hjulens gaspådrag, baserat på den sensordata sensorenheten tillhandahåller.

Armenheten kan vid en upplockningsstation, med vetskapen om var ett föremål befinner sig, plocka upp föremålet med robotarmen. Den har kontroll över samtliga servon i armen och vet hur dessa ska röras vid kommando. Med hjälp av inverterad kinematik kan den, vid automatiskt läge, beräkna hur vardera led ska röra sig för att nå föremålet och sedan plocka upp det.

Den datorbaserade programvaran består av ett instrumentbräde där användaren kan se vilka beslut som tas av roboten samt vilka sensordata de olika sensorerna får in. Vidare kan roboten, inklusive armen, styras genom kommandon i det grafiska gränssnittet.



Figur 3: En översikt över hur de olika delsystemen är sammankopplade.

3.2 Sensorer och sensorplacering

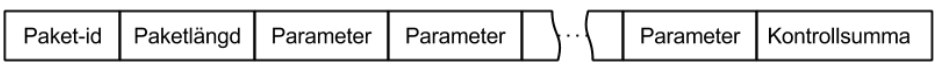
En linjesensor sitter längst fram på roboten, nära marken. En RFID-läsare sitter parallellt med och nära marken under roboten för att underlätta avläsning av RFID-taggar. På vardera sida av roboten är en avståndssensor monterad på ett servo, för att kunna svepa över ett område och skanna efter föremål att plocka upp.

4 Gemensamma funktioner

Detta avsnitt beskriver olika funktioner som inbegriper eller används av flera olika moduler.

4.1 Protokoll för seriell kommunikation mellan robot och PC

Protokollet som roboten och PC-gränssnittet använder är uppbyggt av olika typer av paket. Varje typ av data som skickas har ett fördefinierat unikt paket-ID, som indikerar för en mottagare hur data ska hanteras. Paket-ID:t är alltid den första byten av en överföring. Figur 4 visar hur ett paket är uppbyggt.



Figur 4: Ett paket i den seriella kommunikationen mellan robot och PC, varje ruta är en byte lång.

Paketlängden som skickas med är antalet parametrar + 1, eftersom även kontrollsumman ingår i paketet. Paketlängden indikerar alltså hur många byte som finns kvar i paketet efter paketlängden.

Kontrollsumman bildas genom att summera värdena av alla bytes i paketet, exklusive kontrollsumman själv, trunkera summan till 8 bitar, och invertera värdet bitvis.

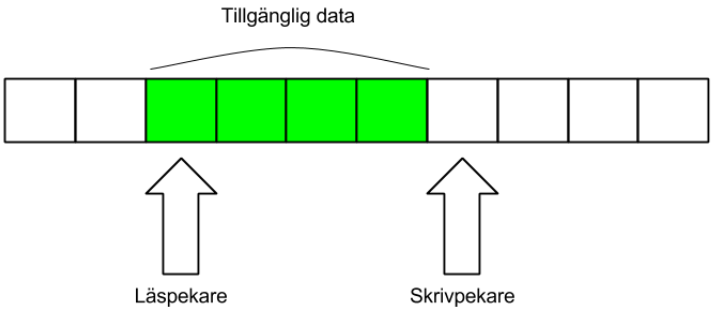
Tabeller över vilken information som skickas mellan roboten och persondatorgränssnittet är bifogade i appendix D.

4.2 USART

AVR-processorns inbyggda USART[1]-funktionalitet används av flera delsystem. Därför finns ett gemensamt bibliotek med funktionalitet för att skicka och ta emot data över en seriell lina.

Biblioteket använder en databuffer för att lagra mottagen data. Till buffern hör två pekare, en läspekare och en skrivpekare, som pekar till olika element i buffern och flyttas upp när man läser eller skriver i buffern. Genom att pekarna endast kan anta värden mellan 0 och 255, samma antal värden som det finns element i buffern, fås funktionalitet som i en ringbuffer eftersom pekarnas värden svämmar över till 0 när de ökas från 255.

När avbrottet, som indikerar att en byte har tagits emot, sker kommer den byte som tagits emot skrivas in i buffern på den position som skrivpekaren pekar på. Därefter ökas pekarens värde ett steg.



Figur 5: Byte-buffern som USART-biblioteket använder, och vilken data som räknas som tillgänglig och ej ännu läst. En ruta föreställer en byte.

För att läsa en byte från buffern anropas funktionen `usart_read_byte`. Den undersöker om det finns någon data tillgänglig som inte redan har lästs (med andra ord, om läs- och skrivpekaren skiljer sig från varandra, se figur 5). I annat fall väntar den tills dess att det finns data, dock som mest under en bestämd timeout-tid. När det finns data läser den in en byte från den position i buffern som läspekaren pekar på till en variabel, och stegar upp läspekaren med ett steg.

[1] Universal Synchronous/Asynchronous Receiver/Transmitter funktionalliet hos ATmega1284P som möjliggör överföring från paralell till seriell kommunikation. [1]

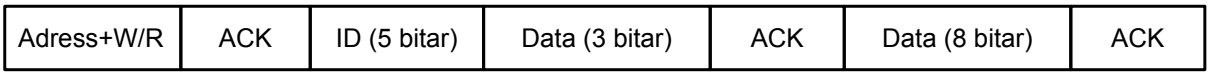
För att skicka data används funktionen `usart_write_byte` som undersöker om den inbyggda USART-modulens statusflaggor tillåter att man skickar data, och i sådana fall ger funktionen en byte till USART-modulen som direkt matar ut den på den seriella linan. Går det inte att skicka väntar funktionen till dess att det går innan den skriver data till USART-modulen.

4.3 Intern buss

För att delsystemen internt ska kunna kommunicera med varandra finns en intern buss av typen multimaster, se avsnitt 4.3.2, I^2C[2] implementerad.

4.3.1 Protokoll

Alla transaktioner består av en adress med en skriv- eller läsbit, följt av två byte med data. Då enheten som initierat kontakten (master) vill skriva ut på bussen består de första 5 bitarna av ett ID. Detta ID bestämmer vilken funktion hos mottagaren, som alltså agerar slav, som ska hantera den data som överförs i de resterande 11 bitarna.



Figur 6: En lyckad skrivning från master till slav. Om NACK (No Acknowledgement, innebär att mottagaren ej mottagit data) skulle ha returnerats istället för ACK (Acknowledgement innebär att mottagaren tog emot data) kommer överföringen att avbrytas.

I fallet att master vill göra en förfrågan på bussen gör den först en skrivning till slaven. Detta gör då att slaven kör funktionen på det ID den fick med de resterande 11 bitar som inargument. Denna funktion returnerar ett 16 bitars värde vilket kommer vara det värde som skickas ut då mastern sedan begär en läsning från slaven. Mellan dessa två transaktioner är det viktigt att mastern inte släpper kontroll över bussen, eftersom slaven då skulle kunna returnera fel värde om en annan enhet skrivit till den efter den första transaktionen.

4.3.2 Implementation

Bussen är implementerad så att alla enheter kan initiera en transaktion och därmed kan alla enheter vara master på bussen. Detta gör att två eller fler enheter kan börja kommunicera på bussen samtidigt. Då detta sker kommer den enhet som skickar en låg bit först att vinna bussen och fortsätta med sin transaktion. De andra enheterna kommer då upphöra med sina transaktioner och efter att ett stop skickats på bussen kommer de att försöka igen. Detta kommer enheterna att göra tills dessa att de lyckats med en transaktion.

[2] Inter-Integrated Circuit, en synkron seriell multimasterbuss.[2]

Enhet	**Adress**
Sensor	4
Kommunikation	5
Chassi	1
Arm	6

Tabell 1: Adresser till de olika delsystem på den interna bussen.

I tabell 1 finns adresserna till de olika delsystemen. Adresserna har valts så att en prioritering av transaktioner till de olika delsystemen finns då flera använder bussen samtidigt. Det betyder i detta fall att transaktioner till chassit alltid har högst prioritet.

När en enhet gör en transaktion på bussen och enheten som tar emot data returnerar NACK efter någon byte i överförningen kommer transaktionen att avbrytas. Detta innebär även att en ny transaktion inte kommer att startas per automatik.

För att bestämma vilken funktion som ska hantera data på vilket ID finns funktionerna `bus_register_receive`, som registrerar en funktion för mottagen data, samt `bus_register_response` som registrerar en funktion då data ska returneras över bussen.

Tabeller över vilka funktionsanrop som kan göras och vilka ID:n som finns implementerade på de olika delsystemen samt vad de gör är bifogade i appendix C.

Funktionen för att skicka data på bussen är `bus_transmit`, och för att göra en förfrågan finns `bus_request`. Båda dessa returnerar 0 då transaktionen lyckades. Detta innebär att om användaren vill vara säker på att en transaktion ska gå fram på bussen måste dessa funktioner köras tills dess att de returnerar 0.

Hastigheten på bussen har valts till så låg som möjligt, detta för att få maximal stabilitet. Eftersom hastigheten sätts genom en division av klockhastigheten till processorn kommer bussen att gå olika fort beroende av vilken enhet som är master. Detta innebär att hastigheten varierar mellan cirka 70 och 90 kHz.

4.4 Utmatning på LCD-skärm

Alla enheter kan mata ut information på kommunikationsenhetens LCD-skärm. Detta görs genom biblioteket `lcd_interface` och funktionen `display`. Funktionen kan ta godtyckligt antal parametrar, dock som minst två där den första anger vilken av LCD-skärmens två rader man vill använda och den andra är en sträng med texten som ska skrivas ut. Övriga parametrar är variabler som man vill mata ut, dessa refereras till i textsträngen med samma syntax som är standard i funktionen `printf` i C.

LCD-skärmen kommer att rotera mellan de fyra olika enheternas meddelanden med ca två se-

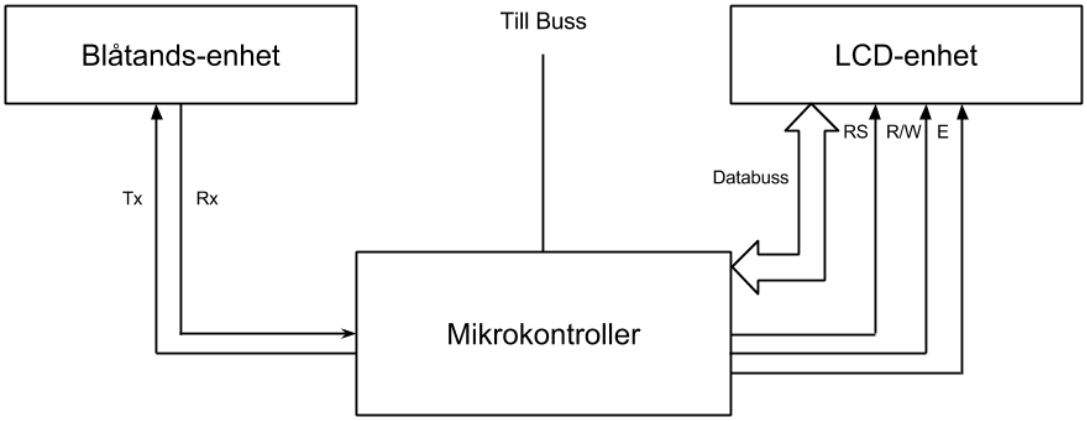
kunders intervall. Varje ”sida” som visas på skärmen identifieras med en bokstav för vilken enhet som visas för tillfället: ”C” för chassienheten, ”S” för sensorenheten, ”K” för kommunikationsenheten och ”A” för armenheten.

5 Kommunikationsenhet

Kommunikationsenheten är robotens gränssnitt för interaktion med omvärlden och har två uppgifter. Den ska

- agera som ett gränssnitt mot LCD-skärmen och möjliggöra att alla delsystem kan skriva ut information på den.
- hantera den seriella kommunikationen över blåtand och möjliggöra att roboten kan skicka och ta emot information till och från PC-gränssnittet.

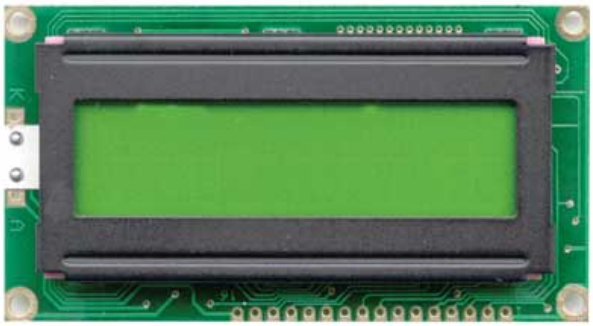
Figur 7 ger en övergripande bild över kommunikationsenhetens beståndsdelar.



Figur 7: Blockschema över kommunikationsenheten.

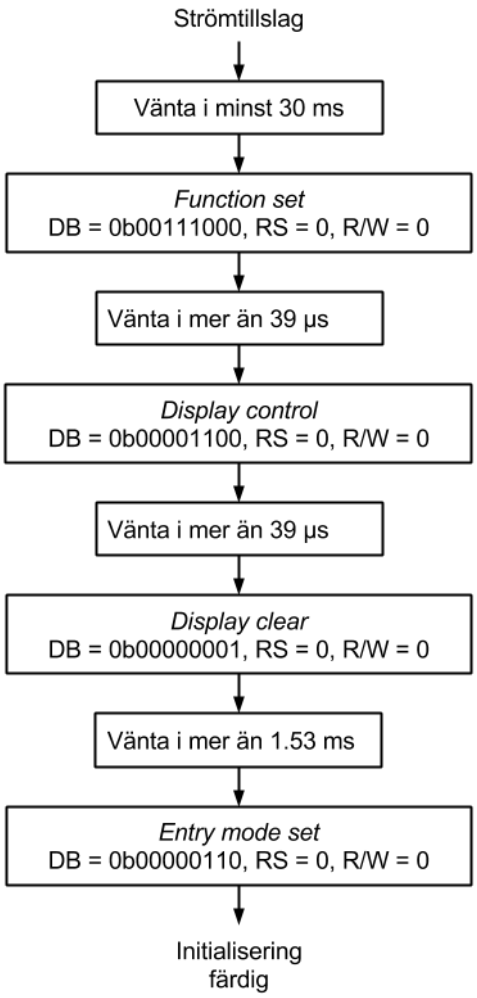
5.1 LCD-gränssnitt

LCD-skärmen som är monterad på kommunikationsenheten är av modellen JM162A. Den styrs genom en parallell databuss och ett antal styrsignaler från kommunikationsenhetens processor till en intern processor i LCD-enheten. Skärmen har två rader med 16 tecken vardera. [3]



Figur 8: Bild på LCD-skärmen.

Information skickas till LCD-enheten antingen som instruktioner eller data. Styrsignalernas koppling visas i figur 7. `RS` styr om databussens värde tolkas som instruktion eller data, `R/W` styr ifall information ska skrivas eller läsas, och styrsignalen `E` aktiverar överföring av information. Genom att låta `E` gå från hög till låg läser LCD-enheten in värdet som finns på databussen, antingen som en instruktion som ska utföras (om `RS` är låg) eller som data som ska lagras i LCD-enhetens minne (om `RS` är hög). Figur 9 visar vilka kommandon som skickas för att initiera LCD-enheten.



Figur 9: Flödesschema över initialisering av LCD-enhet. Kommandon som skickas är kursiverade och har värdena på DB (i binär notation), RS och R/W som parametrar.

Efter denna initiering kan data skrivas ut på skärmen genom att sätta `RS` till 1 och skriva ut ASCII-koden[3] för en symbol på databussen. Genom att först skicka en instruktion för att ställa in vilken adress i dataminnet som ska skrivas till härnäst kan man välja var en symbol ska skrivas ut, en viss adress i dataminnet svarar mot en position på skärmen.

Alla delsystem kan skriva ut information på skärmen genom ett gemensamt bibliotek, *lcd_interface*, som beskrivs närmare i avsnitt 4.4.

5.2 Seriell kommunikation över blåtand

Anslutningen till datorn sker med blåtandsmodemet *BlueSMiRF Gold* som är monterat på roboten [7]. På PC-sidan skapas vid parkoppling med modemet en virtuell serieport, som emulerar

[3] American Standard Code for Information Interchange, ett sätt att koda grundläggande alfanumeriska tecken och andra symboler.

en fysisk COM-port eller motsvarande. Kommunikationen sker sedan över denna port som om den vore en vanlig RS-232-port[4].

På robotsidan används AVR-processorns inbyggda modul för USART, och det gemensamma biblioteket för USART som används av flera delsystem – se avsnitt 4.2. Följande parametrar ställs in i det protokoll som roboten använder:

- Datahastigheten är 115 200 bps.
- Data skickas som 8-bitars värden utan någon paritetsbit.
- Ingen flödeshantering eller handskakning används.
- Varje värde om 8 bitar avslutas med en stoppbit.

Hur olika typer av information överförs mellan robot och PC beskrivs närmare i avsnitt 4.1.

6 Sensorenhet

Sensorenhetens uppgift är att samla in rådata från de olika sensorerna och formatera denna till information som är användbar för robotens övriga delsystem. På begäran av andra delsystem skickar sensorenheten ut data via bussen. Sensorenheten är utrustad med en linjesensor vars uppgift är att ge information om robotens position i förhållande till tejplinjen. Informationen används som styrdata och skickas över bussen i form av en tyngdpunkt till chassimodulen för styrreglering.

Vidare är roboten utrustad med två sidoskannrar, en på vardera sida. Dessa används för att lokalisera föremålet roboten ska plocka upp. Om ett föremål hittas räknar sensorenheten ut en koordinat för föremålet och skickar koordinaten över bussen till armenheten.

6.1 Sensorer

Sensorenheten använder tre olika sensortyper: avståndssensor, linjesensor och RFID-läsare. Dessa beskrivs kortfattat nedan. En djupare beskrivning av hur komponenterna fungerar finns i bilaga sensorfördjupning.

6.1.1 Avståndssensorer

Avståndssensorerna används till robotens sidoskannrar. De mäter avstånd genom att skicka ut infrarött ljus som sedan reflekteras tillbaka i en PSD, Position Sensitive Detector. Avståndssensorerna som används fungerar för avstånd mellan $4 - 30$ cm. Eftersom de först mäter ett digitalt

[4]RS-232 är en typ av seriell kommunikation vid överföring av data [8].

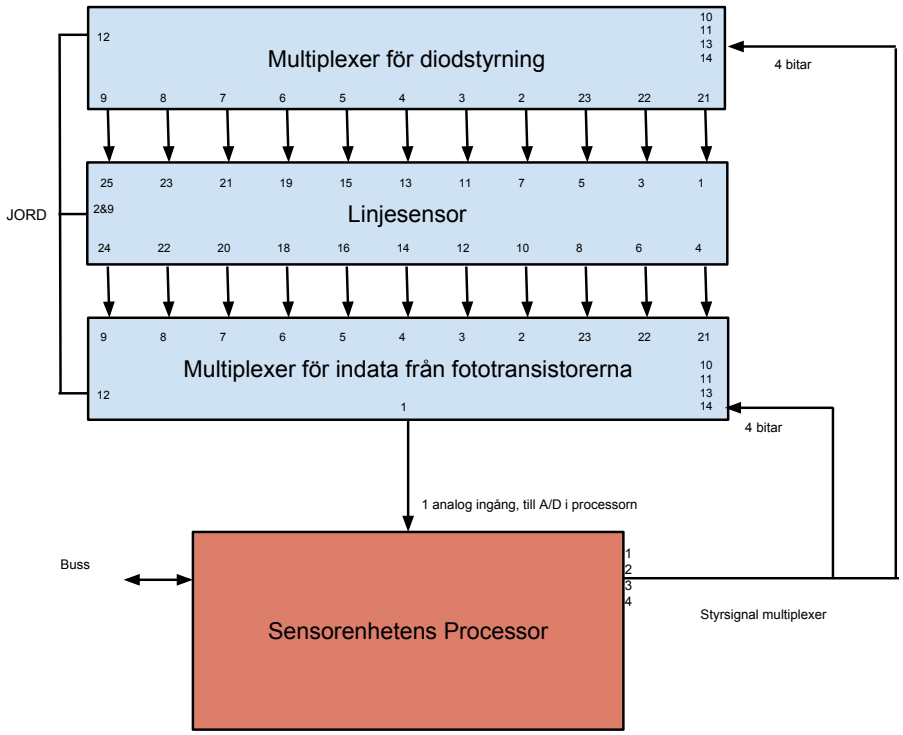
värde och sedan skickar ut en analog spänning uppstår brus på utsignalen. Detta brus är relativt högfrekvent varför det filtreras bort med hjälp av ett lågpassfilter innan signalen når processorn. Se kopplingsschema, figur 17. [4]

En avläsning från en avståndssensor görs genom att omvandla sensorns analoga utspänning till ett digitalt värde. När A/D-omvandlingen är klar jämförs den A/D-omvandlade spänningen med en tabell över kända avstånd och spänningar för att översätta denna till ett avstånd i millimeter. Värden som inte finns i tabellen beräknas med linjärinterpolering.

Avståndssensorerna är, inom det intervall de är designade att arbeta, kapabla till att uppnå hög precision men olika föremål kan få avståndssensorn att ge ut olika spänningar för samma avstånd. Detta leder till att det är av stor vikt att alla avståndssensorer kalibreras i den miljö de är tänkta att användas. Kalibreringen sker genom att identifiera vilka spänningar som svarar mot kända avstånd och föra in dessa som konstanter i sensorenhetens programkod. Avståndssensorerna har även ett område inom vilket de klarar av att känna av avstånd. Detta område breder huvudsakligen ut sig i horisontell led. Detta har på roboten tagits i beaktning genom att montera de avståndssensorer som sitter på sidoskannrarna vertikalt snarare än horisontellt.

6.1.2 Linjesensor

Reflexsensormodulen består av en uppsättning IR-dioder och fototransistorer som tillsammans används för att mäta reflektionsförmågan hos underlaget. Eftersom reflexsensormodulen består av 11 stycken separata IR-dioder och fototransistorer används en multiplexer och en demultiplexer för att styra sensorn. Demultiplexern driver IR-dioderna och multiplexern används för att ta in insignalerna från fototransistorerna. Linjesensorn är kopplad i enlighet med kopplingsschemat i figur 17. [9]



Figur 10: Ett blockschema över linjesensorn'.

Reflexsensorn ger ut en analog spänning som är omvänt proportionell mot underlagets reflekterande ljusstyrka. En inläsning från linjesensorn sker genom ett funktionsanrop till linjesensorinläsningsfunktionen som finns på sensorenheten. När denna funktion anropas uppdateras en uppsättning variabler som är lagrade i en vektor. Vektorn är 11 element lång, där varje element representerar en reflexsensor.

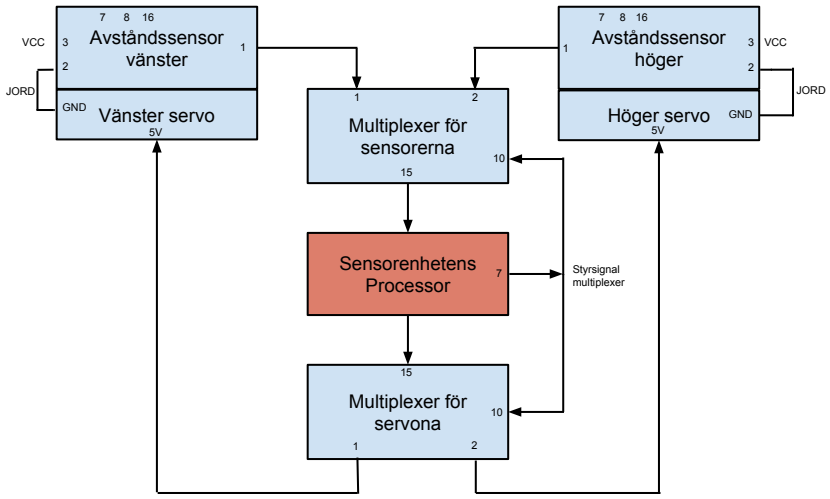
När uppdateringen av linjesensorn startas för första gången är multiplexrarna som styr linjesensorn inställda så att reflexsensorn längst till vänster väljs. Efter detta startas A/D-omvandling på kanal 0 som svarar mot den pinne på processorn som reflexsensormodulen är kopplad till. Under vidare körning så påbörjas varje uppdatering av reflexsensorerna med att en A/D-omvandling startas varefter programmet väntar på att omvandlingen ska slutföras. När A/D-omvandlingen är färdig läggs sedan det erhållna värdet in i linjesensorns datavektor.

6.1.3 RFID-läsare

Den RFID-läsare som används är en “Parallax Serial” [6]. Denna kräver två pinnar på processorn enligt kopplingsschema i figur 17. När chassit har stannat på en station beordras sensorenheten att göra en RFID-läsning. Då körs ett program på sensorenheten som aktiverar läsaren, rensar läsarbufferten och sedan väntar programmet på att en inläsning sker, vilket i regel tar cirka 150 - 300 ms. Läsaren kommunicerar med sensorenhetens processor via USART och så fort antennen låst sig på en tagg skickas värdet till processorn där det lagras i en buffert. Så fort det finns data i bufferten väntar programmet ytterligare 50 ms för att all data ska hinna läsas in till bufferten från RFID-taggen innan avläsning av bufferten sker. Sedan jämförs det inlästa värdet med RFID-taggar som redan finns lagrade i processorns minne. Om någon av de redan lagrade taggarna matchar det inlästa värdet skickar sensorenheten tillbaka den siffra som står på motsvarande RFID-tagg. Om ingen inläsning gjorts innan 400 ms ger programmet upp och skickar till chassienheten att inget hittades.

6.2 Sidoskanner

Sidoskannerns uppgift är att hitta det objekt som ska plockas upp när roboten har stannat vid en plockstation. Sidoskannern består av två avståndssensorer, på höger respektive vänster sida om roboten, monterade på varsitt servo. Servona styrs med hjälp av pulsbreddsmodulering. Servot får en puls var 20:e ms och pulsbredden avgör vilket läge servot antar. En pulsbredd på ungefär 0.5 ms motsvarar servots minsta utslag och en pulsbredd på ungefär 2.5 ms motsvarar max vinkelutslag. Värden på dessa utslag varierar dock något från servo till servo. [10]

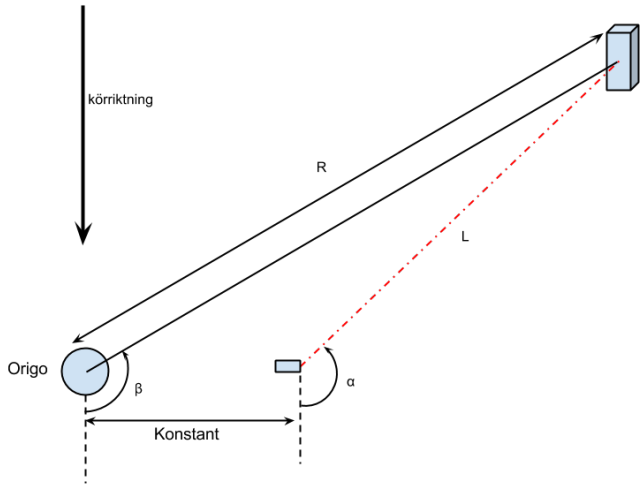


Figur 11: Ett blockschema över sidoskanrarna

För att få ett servot att svepa över robotens ena sida ökas pulsbredden inkrementellt med en stegkonstant motsvarande ett servoutslag på en grad. För varje iteration, dvs. för varje vinkel servot står i, kommer avståndssensorn lägga in 20 stycken A/D-omvandlade avståndsmätningar i en fältstruktur för att sedan ta medianvärdet av mätningarna. Detta görs för att bli av med eventuella avvikande värden som fås ur avståndssensorerna. Slutligen omvandlas värdet till ett avstånd i millimeter genom att interpolera mellan de olika referensvärden som finns lagrade i processorn. Om avståndet är större än vad armens räckvidd är innebär det att att inget föremål detekterats och servots vinkel stegas upp ytterligare en grad.

När avståndssensorn däremot påträffar ett föremål inom armens räckvidd sparas både vinkeln som servot står i och avståndet till föremålet undan innan sidoskannern stegar upp igen. För varje vinkel som avståndssensorerna fortfarande träffar föremålet sparas avståndet undan och slutligen även den sista vinkeln då den fortfarande träffade ett föremål.

Den vinkel som föremålet slutligen beräknas stå i ligger mitt emellan den första och sista vinkeln där föremålet detekterats. Vidare beräknas avståndet till föremålet som medianen av de mätningar som gjordes under tiden föremålet detekterats. Detta görs för att minimera den inverkan som kraftigt avvikande värden annars skulle kunna få. Medelvinkeln α och medianavståndet L används sedan för att beräkna en koordinat utifrån vilken armen kan plocka upp föremålet. Denna koordinat är planpolär och består således av en vinkel β och ett avstånd, R. Här är β den vinkel som basplattan ska stå i för att armen ska vara riktad mot föremålet och R är avståndet från robotens mittpunkt ut till föremålet. Se figur 12



Figur 12: Vänster sidoskanner.

6.2.1 Koordinatberäkning

När sidoskannern har hittat ett objekt och med flera mätningar noggrant identifierat avståndet L och vinkeln α används dessa för att räkna ut R och β genom att kalla på två funk-

tioner, `calculate_angle_coordinate` och `calculate_distance_coordinate`. Vid montering av sidoskannrarna mäts avståndet från robotens origo, dvs. armens mittpunkt, till servonas rotationsaxel, se figur 12. Avståndet definieras som konstanten Origo-to-scanner-distance och används i koordinatberäkningsfunktionerna.

En temporär koordinat för objektet i förhållande till origo bestäms:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} Konstant + L\sin(\alpha) \\ L\cos(\alpha) \end{pmatrix}$$

Sen fås R och β:

$$\begin{pmatrix} R \\ \beta \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2} \\ \arctan(\frac{x}{y}) \end{pmatrix}$$

6.3 Linjeföljning

Linjeföljning är en central del i robotens förmåga att utföra sitt uppdrag. Nedan beskrivs de väsentliga delar som ingår i denna process. Inläsning från linjesensorn beskrivs i 6.1.2.

6.3.1 Korsning, avbrott och plockstationsdetektering

Plockstationsdetektering sker genom att roboten först och främst kontrollerar huruvida linjesensorn registrerar tejp utöver den tejpade linjen. Ifall att de fyra sensorerna längst ut antingen till höger eller vänster på reflexsensormodulen indikerar tejp innebär detta att vi antingen är vid en plockstation eller vid en korsning. Först när det har skett 2000 A/D-omvandlingar kommer roboten att stanna på en plockstation. Under denna tid kommer roboten fortlöpande att kontrollera huruvida linjesensorn registrerar tejp på den andra sidan om roboten. På detta sätt säkerställs att roboten inte stannar vid korsningar. Om den å andra sidan registrerar tejp på andra sidan också innebär det korsning och den kan köra vidare.

För att hantera avbrott i tejplinjen kontrolleras fortlöpande huruvida roboten är över en linje. I det fall att roboten tappar linjen helt och hållet så kommer den utifrån sensorenheten skickade tyngdpunkten att vara 127, vilket motsvarar att linjen ligger på mitten. Eftersom att avbrott i tejpen i enlighet med banspecifikationen som längst får vara 10 centimeter långa är detta tillvägagångssätt fullt tillräckligt för att hantera avbrott i tejpen.

6.3.2 Tyngdpunktsberäkning

För att roboten ska känna till sin egen position i förhållande till linjen så ses de värden som registreras av de individuella sensorerna på reflexsensorn som tyngder. Ett högre värde motsvarar en större tyngd. De olika tyngder som finns på linjesensorn används sedan för att beräkna

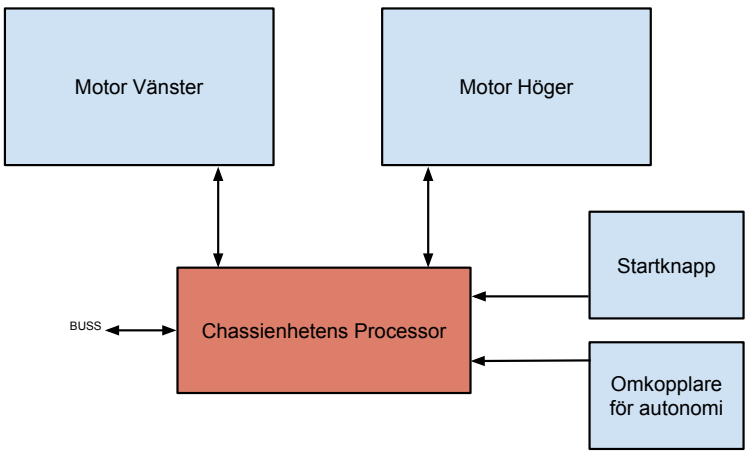
en tyngdpunkt hos linjesensorn. Eftersom att de reflexsensorer som läser av tejpen kommer att väga betydligt mer än de som läser av golvet kommer tejpen alltid att finnas där tyngdpunkten beräknas ligga hos linjesensorn.

Tyngdpunktsberäkningen använder sig av två stycken variabler. Den första variabeln innehåller linjesensorns totalvikt, alltså summan av de individuella reflexsensorernas värden. Den andra variabeln innehåller sensorns totalvikt, men där de olika reflexsensorerna även har blivit multiplicerade med en skalfaktor. Denna skalfaktor fyller två funktioner. Först och främst behövs den för att ge sensorerna på kanterna en större hävarm relativt linjesensorns mittpunkt. Vidare används den till att skala den slutgiltiga tyngdpunkten så att hela talområdet hos det åtta bitar stora returvärdet används för att representera linjen. Denna metod kommer att resultera i att tyngdpunkten representeras av ett åtta bitar stort heltalsvärde.

Då tyngdpunkten är 0 ligger tejpen alltså längst till vänster på reflexsensorn på samma sätt som en mekanisk tyngdpunkt hade legat längs till vänster om all massa hos en linje hade legat till vänster. Motsvarande innebär tyngdpunkt 255 att linjen ligger längs ut till höger på linjesensorn.

7 Chassienhet

Chassienheten tar alla övergripande beslut om vad som ska göras, t.ex. är det chassienheten som bestämmer när och på vilken sida armen ska plocka upp och lämna av objekt. Chassienheten sköter styrningen med PD-regleringen med hjälp av den tyngdpunkt som den får av sensorenheten via bussen. Detta för att kunna följa en svart tejpad linje på marken. På chassienheten finns en omkopplare för att kunna välja mellan autonomt och manuellt läge samt två tryckknappar, en tryckknapp för att starta linjeföljningsproceduren samt en för återställning.



Figur 13: Ett blockschema över chassienheten

7.1 Funktion

Chassit har flera funktioner som anges i följande punktlista:

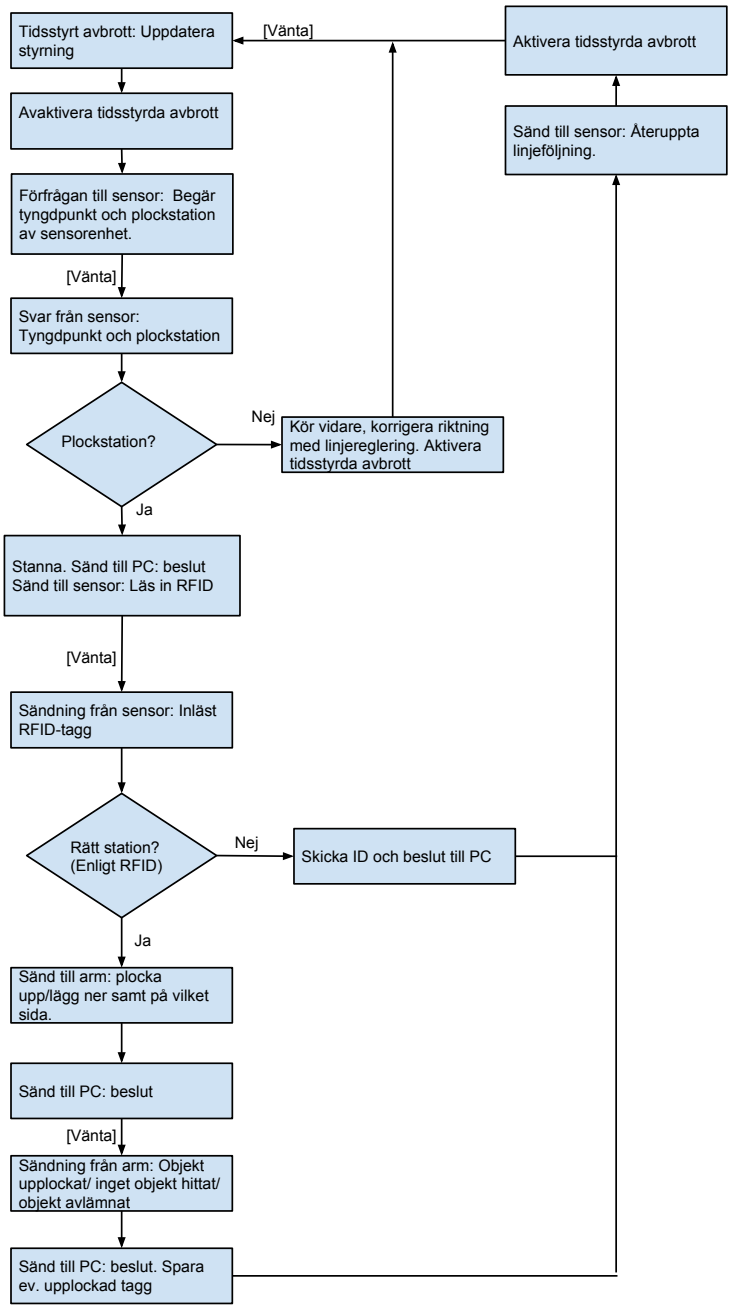
- Kan styras både autonomt och genom order från dator via kommunikationsenheten.
- Styr de fyra motorerna med PWM-styrning.
- Använder regleralgoritm för att kunna följa en svart tejpad linje utan att slingra sig fram.
- Bestämmer över övriga enheter genom att ta beslut och skicka kommandon till arm- och sensorenheten.
- Skickar styrbeslut till dator via kommunikationsenheten samt skriver ut dem på kommunikationsenhetens display.

7.2 Översiktlig beskrivning av programmet

Chassits huvudprogram startas genom ett knapptryck eller via ett kommando från persondatorn.

Programmet på chassit beräknar styrriktning med hjälp av en tyngdpunkt som den begär med jämna mellanrum från sensorenheten . Enheten räknar också hur många stationer den passerar innan den är tillbaka på stationen den började på. Detta för att roboten ska veta när samtliga stationer är hanterade. Enheten lyssnar även alltid på styr- och stoppkommandon från PC:n.

Chassit får information om att roboten är på plockstation av sensorenheten i samma paket som den får tyngdpunkten. Vid plockstation avgör roboten om den ska stanna eller inte. När den stannat på en station skickas ett kommando om att läsa RFID-tagg till sensorenheten. Efter detta avgör roboten om den ska skicka en instruktion till armenheten att plocka föremål, lämna av föremål eller om den bara ska köra vidare till nästa station. Alla beslut skickas via kommunikationsenheten till datorn.



Figur 14: Flödesdiagram för huvudprogram till chassienheten.

7.2.1 Följa linje samt motorkontroll

För att kontrollera hastighet till motorerna används PWM-styrning med en uppdateringshastighet på 1 kHz. Detta uppnås genom att klockdelaren (N) till timern sätts till 8 samt att timern räknar till 1999 innan en ny period ska starta.

För att kunna följa linjen utan att slingra sig fram implementeras en regleralgoritm. Den regle-

ring som görs är en PD-reglering, enligt formeln:

$$u[t] = K_p e[t] + \frac{K_d}{\delta} e[t]$$

Eftersom tiden mellan uppdateringarna är konstant kan δ sparas i processorns minne som ett konstant värde. Om $u[t]$ är negativ ska motor på höger sida bromsas medan den andra kör med full fart och vice versa. För att beräkna motorkontrollen till PWM används följande formel:

$$P[t] = 1999 - u[t]$$

Där $P[t]$ är värdet räknaren till vänster/höger motor räknar till. Om $P[t]$ är negativ innebär det att hjulen som bromsas istället börjar snurra åt motsatt håll med hastighet motsvarande $|P[t]|$. Den andra motorn går i full hastighet, alltså är $P[t] = 1999$ för den motorn. Av dessa värden kommer K_p och K_d att kunna ändras för att optimera regleringen.

7.2.2 Identifiering av plockstationer

Varje gång roboten stannar vid en plockstation skickar chassit en begäran till sensorenheten att läsa av plockstationens RFID-tagg. Taggens ID skickas tillbaka till chassit och sparas där i en lista. Detta gör att roboten kan räkna hur många stationer som finns på banan och således vet den när den är klar. Den kan även, efter att ha kört första varvet, förutse ifall roboten behöver stanna på nästa station eller inte.

7.2.3 Styrkommando

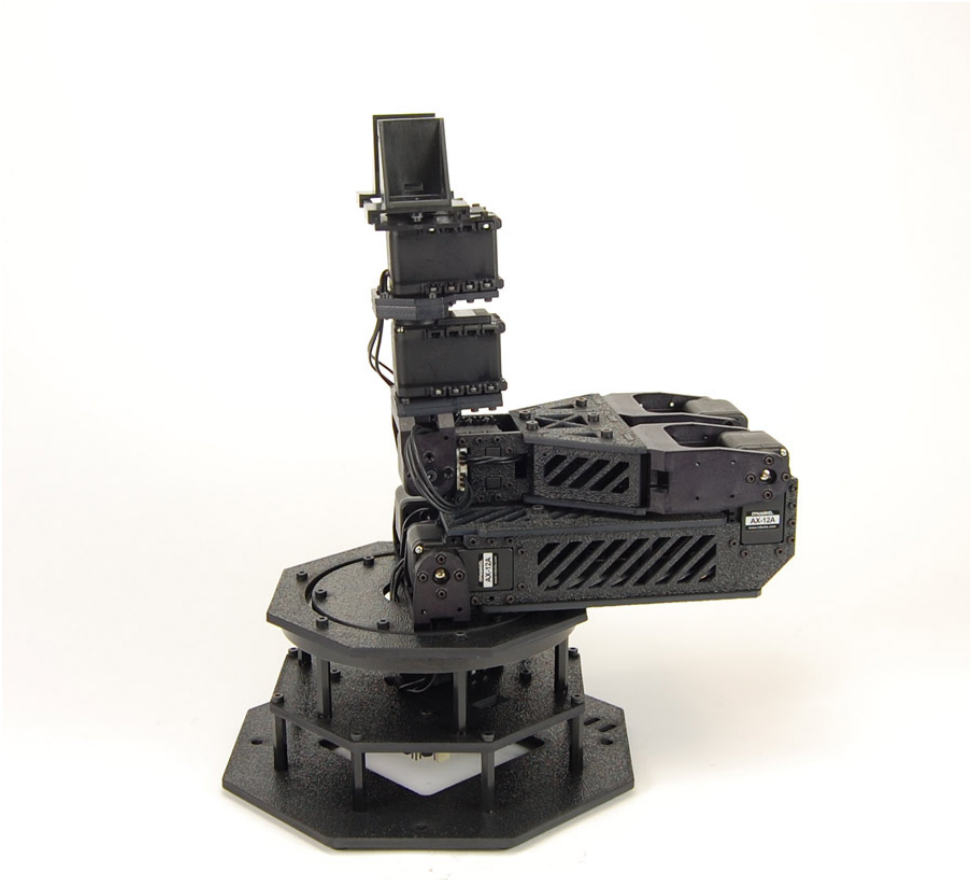
Chassit reagerar alltid på styrkommandon från bussen även om brytaren är satt i autonomt läge. Detta för att kunna korrigera roboten ifall den skulle få för sig att göra något oväntat. Ett styrkommando består av en ändring i riktning och en ändring i gaspådrag. Gaspådrag kan vara positivt och negativt, där positivt för roboten framåt och negativt för roboten bakåt.

7.2.4 Autonomt läge och start-/stoppkommando

När brytaren är i autonomt läge reagerar chassit på startknappen som sitter på chassit. Startknappen påbörjar linjeföljning med hantering av plockstationer beskrivet i 7.2.1.

8 Armenhet

Robotens arm är av modell PhantomX Reactor från Trossen Robotics, som är en servostyrd arm med fyra rotationsleder och en gripklo varpå det sitter totalt 8 st AX-12A-servon. Armen kontrolleras av en mikroprocessor, ATmega 1284P, genom att parallellkoppla servona till en seriell USART-port. Kommunikationen till servona sker via halv duplex via en tri-state-buffer. Armen har en maximal räckvidd på 43 cm och en bas med 300 grader rotationsfrihet.



Figur 15: En bild på robotarmen.

8.1 Funktion

Armen kan styras manuellt via PC-programvaran och kan köras i autonomt läge. Enhetens huvudprogram är avvaktande tills dess att kommando skickas till enheten vilket kan antingen vara ett manuellt styrningskommando eller en order från chassienheten att ta hand om upplockning av objekt.

8.1.1 Manuellt läge

Vid manuellt läge får armen ingen upplockningsorder från chassienheten och väntar därför endast efter ett styrkommando från PC. Från PC får den ett kommando att röra sig i en viss riktning i koordinatsystemet visat i figur 16.

Användaren kan välja att röra armen i djupled (X-axeln), i höjd (Y-axeln), rotera runt Y-axeln eller öppna och stänga klon. Vid ett styrkommando rör sig armen i den angivna riktningen tills dess att ett kommando om att stanna rörelse i den riktningen ges från PC eller om den nått maxläget. Rörelse görs genom att kontinuerligt beräkna hur vinkeln av vardera led på armen skall ändras för att röra sig ett litet steg i den beordrade riktningen. Detta görs genom beräkningar i inverterad kinematik (se avsnitt 8.1.3), sedan gör lederna en vinkelförändring för att sedan repetera enligt ovan.

Armen kan också via en persondator automatiskt röra sig till ett förutbestämt läge, ett startläge, där rörelsen även här beräknas med inverterad kinematik till den position som startläget har. Först rör sig armen rakt upp från det nuvarande läget för att sedan röra sig till den position som startläget har, detta för att eliminera risken att stöta i chassit och virkorten.

8.1.2 Autonomt läge

I autonomt läge avvaktar armenheten tills dess att chassit har identifierat att roboten befinner sig på en plockstation som är aktuell för upplockning eller avlämning. Vid station får armenheten en order, plocka upp eller lämna av föremål, till höger eller till vänster. När ordern inkommer och armen inte redan håller i ett föremål beordras sensorenheten att med hjälp av en sidoskanner söka igenom upplockningszonen. Håller armen redan i ett föremål rör den sig till samma koordinat som för upplockningen av samma föremål, släpper föremålet och rör sig sedan tillbaka till startläget för att slutligen skicka bekräftelse till chassit med status om avlämning lyckats.

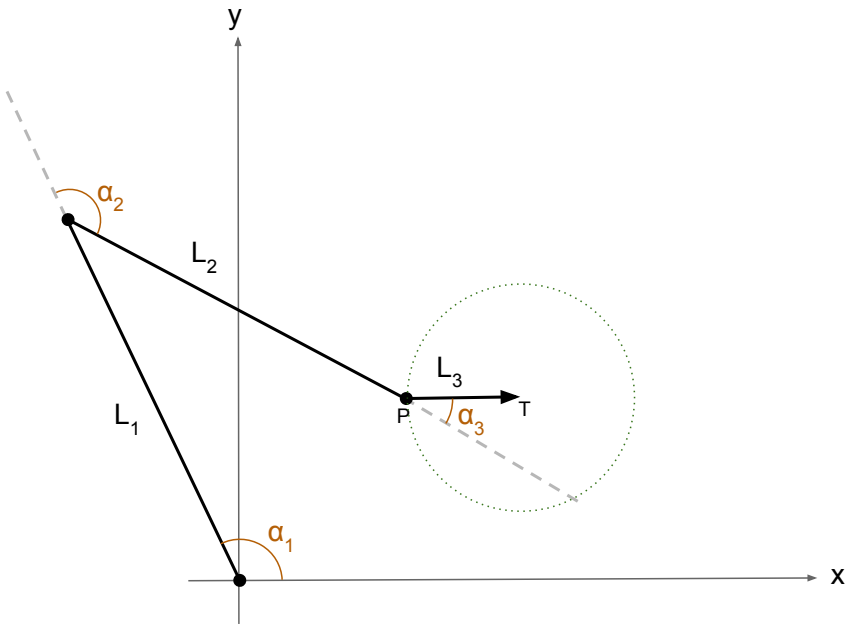
Sensorenheten kommer, när den är klar med sidoskannrarna, skicka tillbaka ett kommando till armenheten med en vinkel och ett avstånd till föremålet, om den hittat något, annars meddelar den att zonen är tom. Med den koordinat som fåtts från sensorenheten beräknas, med inverterad kinematik, hur armens olika leder behöver röra på sig för att nå koordinaten. Armen rör sig sedan mot objektet och väntar på att servona ska nå slutdestinationen, genom att fråga servona om de rör på sig, för att sedan gripa tag och röra sig tillbaka till startläget. Armenheten sparar sedan koordinaten där den plockat upp föremål, vilket används som avlämningsposition, och skickar bekräftelse till chassienheten om att upplockning av föremål lyckats.

8.1.3 Inverterad kinematik

Från sensorenheten ges armenheten en planpolär koordinat för objektet, dvs. ett avstånd till föremålet och en vinkel för bottenplattan. Inverterad kinematik innebär att utifrån givna koordinater bestämma vinklar på robotarmens leder för att få armens spets att nå den givna koordinaten. För att reducera problemets komplexitet kunde armen, utifrån den givna vinkeln, rotera basplattan

mot föremålet och således förenkla det inverterade kinematikproblemet till ett tvådimensionellt sådant.

Problemet som kvarstod blev således att utifrån tre ihopsatta leder finna en lösning för ledvinklarna så att spetsen av kedjan, gripklon, hamnar på den kända positionen för föremålet. Se figur 16



Figur 16: 2-dimensionell schematisk beskrivning över armen och dess leder.

Genom att betrakta figur 16 som ett koordinatsystem med en x- och en y-axel ges gripklons position enligt ekvation 1.

$$f(\boldsymbol{\alpha}) = \begin{pmatrix} L_1 cos(\alpha_1) + L_2 cos(\alpha_1 + \alpha_2) + L_3 cos(\alpha_1 + \alpha_2 + \alpha_3) \\ L_1 sin(\alpha_1) + L_2 sin(\alpha_1 + \alpha_2) + L_3 sin(\alpha_1 + \alpha_2 + \alpha_3) \end{pmatrix} = \begin{pmatrix} T_x \\ T_y \end{pmatrix}, \boldsymbol{\alpha} \in \mathbf{A} \quad (1)$$

$f(\boldsymbol{\alpha})$ har en definitionsmängd $\mathbf{A}$ enligt ekvation 2. Bivillkoren i mängden $\mathbf{A}$ kommer direkt ifrån robotarmens datablad.

$$\mathbf{A} = \left\{ \boldsymbol{\alpha} \in \mathbb{R}^3 : 0 \leq \alpha_1 \leq \pi, -\pi \leq \alpha_2 \leq 0, \frac{-\pi}{2} \leq \alpha_3 \leq \frac{\pi}{2} \right\} \quad (2)$$

Målet är att finna en funktion $f^{-1}(x, y) = \boldsymbol{\alpha}$.

Först väljs P någonstans längs cirkelperiferin runt T med L_3 som radie. Valet kommer med fördel alltid ligga på samma höjd som T, vilket resulterar i att L_3 hålls parallell underlaget,

såvida punkten P inte krockar med roboten. Med en känd koordinat P kan vinklarna α_1 och α_2 bestämmas genom att lösa ekvationssystemet, se ekvation 3.

$$\begin{pmatrix} cos(\alpha_1) \\ sin(\alpha_1) \end{pmatrix} = \frac{1}{L_1^2 + L_2^2 + 2L_2cos(\alpha_2)} \begin{pmatrix} L_1 + L_2cos(\alpha_2) & L_2sin(\alpha_2) \\ -L_2sin(\alpha_2) & L_1 + L_2cos(\alpha_2) \end{pmatrix} \begin{pmatrix} P_x \\ P_y \end{pmatrix} \quad (3)$$

När sinus- och cosinusvärdena är kända för både α_1 och α_2 kan den teckenkänsliga funktionen $atan2$ användas för att räkna ut vinklarna enligt ekvation 4.

$$\begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix} = \begin{pmatrix} atan2(sin(\alpha_1), cos(\alpha_1)) \\ atan2(sin(\alpha_2), cos(\alpha_2)) \end{pmatrix} \quad (4)$$

Eftersom $sin(\alpha_2)$ har två lösningar uppstår således dubbla lösningar till ekvationssystemet. Därför är det viktigt att stämma av resultatet mot definitionsmängden $\mathbf{A}$.

När väl punkten P och T samt vinklarna α_1 och α_2 är kända ges α_3 enligt ekvation 5. Därmed är beräkningarna klara.

$$\alpha_3 = atan2(sin(T_y - P_y), cos(T_x - P_x)) - \alpha_2 - \alpha_1 \quad (5)$$

8.2 Översiktlig beskrivning av programmet

Programmet kommunicerar med servona över USART via funktioner som finns i ett delat bibliotek för denna typ av kommunikation, se avsnitt 4.2. Servona styrs genom att skriva på deras minnen, där en skrivning på särskild plats i minnet leder till en särskild instruktion till servot. För att skriva eller läsa från servona skapas i programmet först ett paket, med utseende beroende på instruktion, med samtliga bitar som ska skickas för att sedan seriellt skicka dem via UART. Vid en överföring, bortsett från när instruktion skickas till alla servon, väntar programmet på att få svar från servot för att kunna diagnostisera om instruktionen kunnat genomföras eller för att tyda en avläsning av servominne. Svar omhändertas genom väntan på mottagningsavbrott på armenheten varpå inkommande data lagras i en mottagningsbuffer som läses av så fort den inte är tom.

Eftersom armenheten är avvaktande utför huvudprogrammet endast instruktioner när statusflaggor blivit satta genom funktionskallelser från andra enheter på bussen.

Med hjälp av inverterad kinematik kan programmet beräkna vilka vinklar armens servon ska anta för att armens gripklo ska anta en given position.

9 Programvara för persondator

Programvaran till persondatorer som finns till roboten används dels för att ge kommandon till roboten, och dels för att visa utmatning av sensordata och beslut som roboten har tagit. För detaljer om användning, se användarmanualen.

Programvaran är utvecklad i utvecklingsmiljön Qt Creator [11], och använder till stor del Qt:s egna standardbibliotek. Den är uppdelad i två klasser, en för att kommunicera med roboten via blåtand och en för det grafiska gränssnittet.

Klassen för blåtandskommunikation använder sig utav QSerialPort, vilket finns i Qt:s standardbibliotek. Ur detta hanterar klassen hur data skickas och tas emot från roboten, och vidarebefordrar denna data till klassen för det grafiska gränssnittet.

Klassen för det grafiska gränssnittet innehåller alla knappar, fält och liknande som finns i det grafiska gränssnittet. Denna använder också till stor del Qt:s egna standardbibliotek. Den hanterar allt som ska hända när en användare interagerar med gränssnittet.

I det grafiska gränssnittet finns också två grafer som ritar ut data från linjesensorn samt sidoskannrar. Dessa använder en klass som heter QCustomPlot, för ytterligare beskrivning se [12].

Referenser

[1] Datablad för mikroprocessorn ATmega1284.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/atmega1284p.pdf

[2] Specifikation för en I2C-buss.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/i2cspec2000.pdf

[3] Datablad för LCD-skärm av modell JM162A.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/jm162a.pdf

[4] Datablad för avståndssensor SHARP GP2D120.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/gp2d120.pdf

[5] Datablad för servo av modell AX-12.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/AX-12.pdf

[6] Datablad för RFID-läsare av modellen Parralax-Reader.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/rfid-reader-v21.pdf

[7] Datablad för Bluetooth modemet FireFly BlueSMiRF Gold.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/firefly.pdf

[8] Informationssida för den seriella kommunikationsformen RS-232
https://docs.isy.liu.se/twiki/bin/view/VanHeden/RS232

[9] Informationsblad om reflexsensoruppsättning.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/reflexsensormodul.pdf

[10] Informationsblad om servostyrning med vanliga hobbyservon.
https://docs.isy.liu.se/twiki/pub/VanHeden/DataSheets/servostyrning.pdf

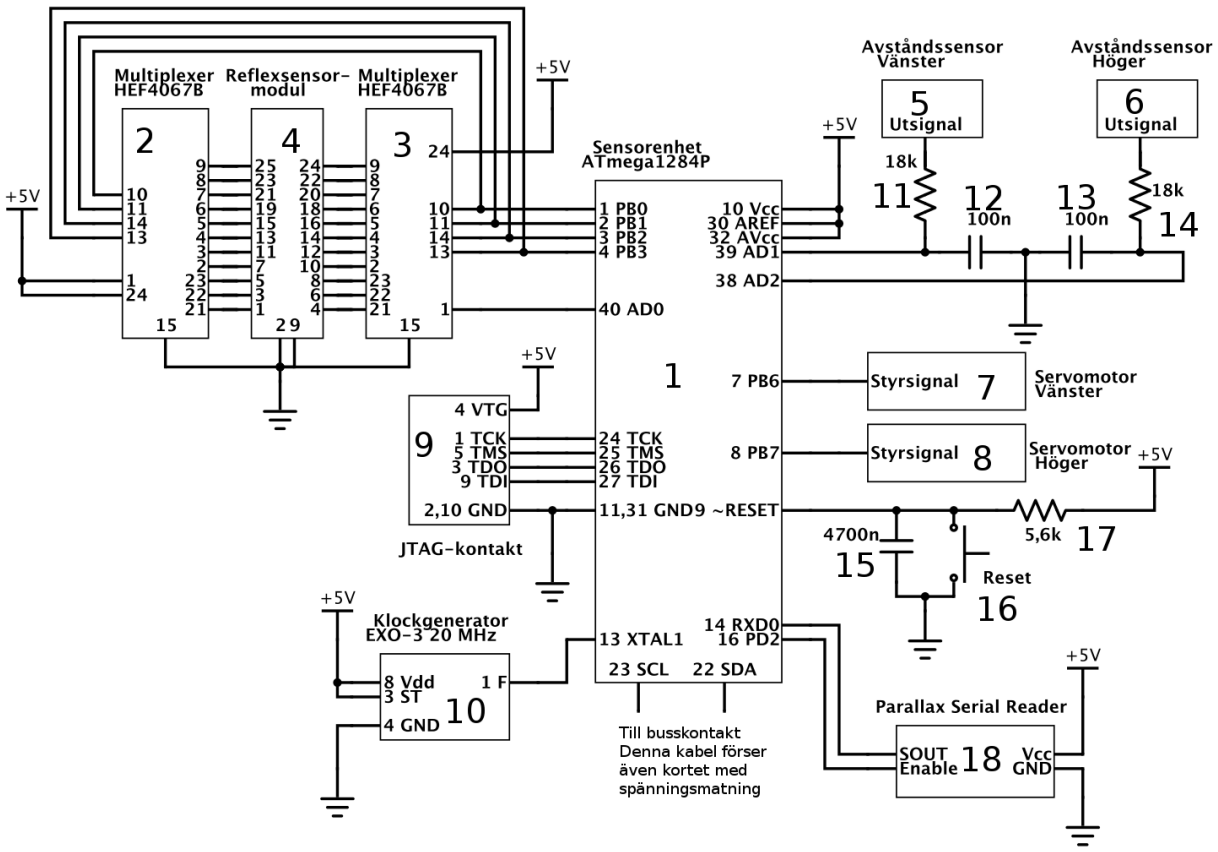
[11] Hemsida för utvecklingsverktyget QT.
http://qt-project.org/

[12] Hemsida för gränsnittskomponenten QCustomPlot.
http://www.qcustomplot.com/

A Kopplingsschema

Robotens elektronik är uppdelad på två virkort. Därför presenteras här ett kopplingsschema för varje virkort.

A.1 Sensorenhet

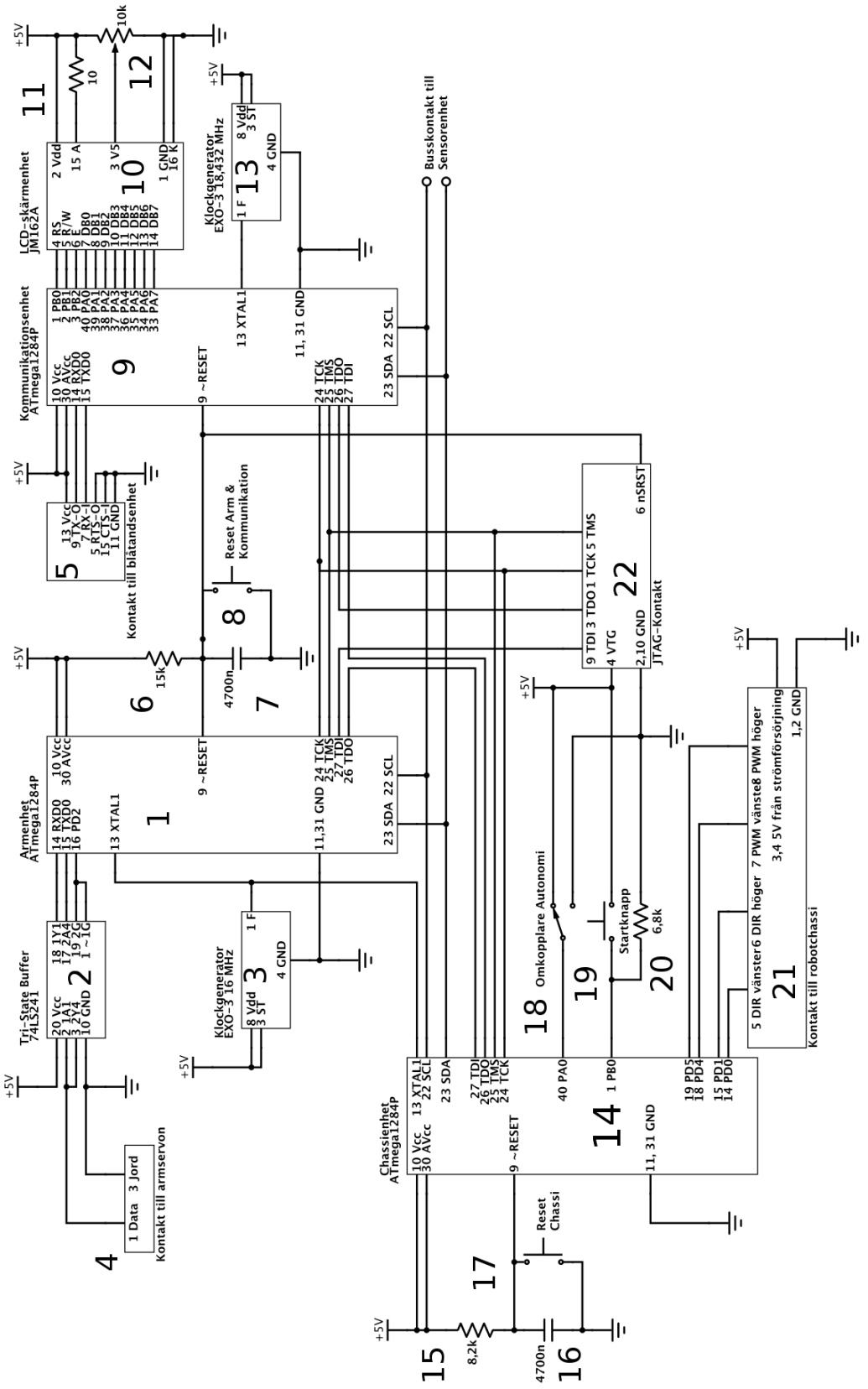


Figur 17: Kopplingsschema för virkortet som innehåller sensorenheten.

Komponentförteckning

1. Sensorenhetens processor, ATmega1284P
2 & 3. 16-kanals multiplexer för linjesensor, HEF4067B
4. Reflexsensormodul
5 & 6. Kontakt för kabel till avståndssensor för sidoskanner, GP2D120
7 & 8. Kontakt för kabel till servo för sidoskanner, HS-422
9. Kontakt för JTAG-kabel
10. Klockgenerator, EXO-3 20 MHz
11 & 14. Resistor för lågpassfilter, 18 kΩ
12 & 13. Kondensator för lågpassfilter, 100 nF
15. Kondensator för reset av Sensorenhet, 4700 nF
16. Resetknapp för Sensorenhet
17. Pullup-resistor för reset av Sensorenhet
18. Kontakt för kabel till RFID-läsare, Parallax Serial Reader

A.2 Kommunikationsenhet, Armenhet, Chassienhet



Figur 18: Kopplingsschema över virkortet som innehåller kommunikationsenheten, chassienheten och armenheten.

Komponentförteckning

1. Processor Armenhet, ATmega 1284P
2. Tri-state buffer, 74LS241
3. Klockgenerator för Armenhet och Chassienhet, EXO-3 16 MHz
4. Kontakt för kabel till armservon
5. Kontakt för kabel till blåtandsenhet
6. Pullup-resistor för reset av Kommunikationsenhet och Armenhet, 15 kΩ
7. Kondensator för reset av Kommunikationsenhet och Armenhet, 4700 nF
8. Resetknapp för Kommunikationsenhet och Armenhet
9. Processor Kommunikationsenhet, ATmega 1284P
10. Skärmenhet för LCD-skärm, JM162A
11. Resistor för att begränsa strömmen till LCD-skärmens bakgrundsbelysning, 10 Ω
12. Potentiometer för att justera LCD-skärmens kontrast, 0 - 10 kΩ
13. Klockgenerator för Kommunikationsenhet, EXO-3 18.432 MHz
14. Processor för Chassienhet, ATmega 1284P
15. Pullup-resistor för reset av Chassienhet, 8,2 kΩ
16. Kondensator för reset av Chassienhet, 4700 nF
17. Resetknapp för Chassienhet
18. Omkopplare för att växla mellan autonomt och fjärrstyrt läge
19. Startknapp för att påbörja autonom drift
20. Pulldown-resistor för startknappen
21. Kontakt för kabel till robotchassi, tillför spänning +5 V till virkortet
22. Kontakt för JTAG-kabel.

B Utdrag från programlistning

Utdrag ur programkod från tre filer, dessa är inte kompletta utan endast utdrag för att exemplifiera kodstruktur.

```
1  /**
2   *      @file bus.c
3   *      @author Andreas Runfalk & Patrik Nyberg
4   *
5   *      Functions for intra processor communication over the two wire
6   interface
7   */
8
9  #include <avr/io.h>
10 #include <avr/interrupt.h>
11
12 /**
13  *      Temporary storage for received data
```

```
14   */
15  uint16_t bus_data;
16
17  /**
18   *      Temporary storage for response data
19   */
20  uint16_t bus_response_data;
21
22  /**
23   *      Array of response callbacks used by bus_register_response()
24   and bus_call_response()
25   */
26  uint16_t (*response_callbacks[64])(uint8_t, uint16_t) = {0};
27
28  /**
29   *      Array of receive callbacks used by bus_register_receive() and
30   bus_call_receive()
31   */
32  void (*receive_callbacks[64])(uint8_t, uint16_t) = {0};
33
34  /**
35   *      Start listening on given address and enable interrupts
36   *
37   *      @param address 7-bit address to listen to
38   */
39  void bus_init(uint8_t address) {
40          // Bus speed, approximately 70 kHz
41          TWBR = 0x80;
42
43          // Enable internal pull-up for SDA and SCL
44          PORTC |= 0x03;
45
46          // Set address on bus
47          TWAR = (address << 1) | 1;
48
49          // Enable the bus
50          TWSR &= 0xfc;
51          TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWIE);
52
53          sei();
54  }
55
56  /**
57   *      Try to acquire control over the bus. Will continuously retry
58   until control is
59   *      aquired. Possible status codes are:
60   *
61   *      - 0 if bytes were successfully sent
```

```
62   *
63   *      @return Status code
64   */
65  uint8_t bus_start(void) {
66          uint8_t status_code;
67          cli();
68          TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN) | (1 << TWIE)
69                  | (1 << TWEA);
70
71          // Wait for TWINT to go high
72          while (!(TWCR & (1 << TWINT)));
73
74          status_code = TWSR & 0xf8;
75          sei();
76
77          // Check if failure to seize bus
78          switch (status_code) {
79                  case 0x08: // Start
80                  case 0x10: // Repeated start
81                          return 0;
82                  case 0x00:
83                          cli();
84                          TWCR &= 0xff ^ (1 << TWSTA);
85                          TWCR |= (1 << TWINT) | (1 << TWSTO);
86                          while (!(TWCR & (1 << TWINT)));
87                          sei();
88                  case 0xf8:
89                  default:
90                          // Catastrophic failure, retry
91                          return bus_start();
92          }
93  }
94
95  /**
96   *      Let go of the control over the buss
97   */
98  void bus_stop(void) {
99          TWCR |= (1 << TWINT) | (1 << TWSTO) | (1 << TWEA);
100 }
101
102 /**
103  *      Write a byte of data on the bus
104  *
105  *      @param data Data byte to write
106  *
107  *      @return Masked status code from TWSR
108  */
109 uint8_t bus_write(uint8_t data) {
```

```
110         uint8_t status_code;
111
112         cli();
113         TWDR = data;
114         TWCR &= 0xff ^ (1 << TWSTA) ^ (1 << TWSTO);
115         TWCR |= (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
116
117         // Wait for TWINT to go high (package sent)
118         while (!(TWCR & (1 << TWINT)));
119
120         status_code = TWSR & 0xf8;
121         sei();
122
123         return status_code;
124 }
```

```
1  /**
2   *      @file automatic_steering.c
3   *      @author Lucas Nilsson
4   *
5   *      Calculates steering direction based on current robot position
6   */
7
8  #include "automatic_steering.h"
9
10 /**
11  *      Store previous error as set by pd_update()
12  */
13 int8_t prev_error;
14
15 /**
16  *      Bus callback for setting proportional gain in PD controller
17  *
18  *      @param id Unused
19  *      @param kp_data Proportional gain to use
20  */
21 void engine_set_kp(uint8_t id, uint16_t kp_data)
22 {
23         proportional_gain = kp_data;
24 }
25
26 /**
27  *      Bus callback for setting proportional gain in PD controller
28  *
29  *      @param id Unused
30  *      @param kd_data Derivative gain to use
31  */
32 void engine_set_kd(uint8_t id, uint16_t kd_data)
```

```
33  {
34          derivative_gain = kd_data;
35  }
36
37  /**
38   *      Calculate new steering based on passed current error and
39   previous error
40   *
41   *      @param curr_error
42   *
43   *      @return Steering wheel position to use. Negative means turn
44   left and positive
45   *              turn right.
46   */
47  int16_t pd_update(int8_t curr_error)
48  {
49          double diff;
50          int16_t p_term;
51          double d_term;
52
53          // differentiation
54          diff = curr_error - prev_error;
55
56          // scaling
57          p_term = proportional_gain * curr_error;
58          d_term = derivative_gain  * diff;
59
60          // save current error as previous error for next iteration
61          prev_error = curr_error;
62
63          // Return regulated signal
64          return p_term + floor(d_term);
65  }
66
67  /**
68   *      Set regulator constants to default values. To be used when
69   initializing robot
70   */
71  void regulator_init(void)
72  {
73          prev_error = 0;
74          proportional_gain = 180;
75          derivative_gain = 5;
76  }
```

```
1   /**
2    *      @file servo.c
3    *      @author Andreas Runfalk
```

```
4    *
5    *      Handles tri-state buffer control and basic servo
6    communications. Many
7    *      functions in this file should only be used in conjunction with
8    'macros defined
9    *      in servo.h.
10   *
11   *      This library is not thread safe. No servo communication should
12   _ever_ be done in interrupts.
13   */
14
15   #include "servo.h"
16   #include "../shared/LCD_interface.h"
17   #include "../shared/bus.h"
18
19   /**
20    *      Set tri-state buffer to write mode
21    */
22   void servo_enable_write(void) {
23           // Set tri-state buffer to write
24           PORTD |= 1 << PORTD2;
25
26           // Wait for tri-state buffer to switch directions
27           _delay_us(20);
28   }
29
30   /**
31    *      Set tri-state buffer to read mode
32    */
33   void servo_enable_read(void) {
34           while (!usart_tx_complete());
35
36           // Set tri-state buffer to read
37           PORTD &= ~(1 << PORTD2);
38
39           // Wait for tri-state buffer to switch directions
40           _delay_us(20);
41   }
42
43   /**
44    *      Initialize USART communication and tri-state buffer control
45    */
46   void servo_init(void) {
47           usart_init(SERVO_DEFAULT_BAUD_RATE);
48
49           // Enable control of tri-state buffer and put in read mode
50           DDRD |= 1 << DDD2;
51           PORTD &= ~(1 << PORTD2);
```

```
52  }
53
54  /**
55   *      Calculate servo checksum for given array from _first_index_ to
56   _last_index_
57   *
58   *      @param first_index First index of array to include
59   *      @param last_index Last index of array to include
60   *      @param parameters[] Array to calculate checksum for
61   *
62   *      @return Checksum
63   */
64  uint8_t servo_calculate_checksum(uint8_t first_index, uint8_t last_inde
65          uint8_t i;
66          uint8_t checksum = 0;
67
68          for (i = first_index; i <= last_index; i++) {
69                  checksum += parameters[i];
70          }
71
72          return ~checksum;
73  }
74
75  /**
76   *      Read response from a servo. Possible status codes are:
77   *
78   *      - 0 if successful
79   *      - 1 if read timed out
80   *      - 2 if checksum is wrong
81   *      - 3 if incorrect non 0xff start bytes
82   *      - 4 if wrong servo return status
83   *      - 5 if servo didn't understand the instruction
84   *      - 6 if servo max torque can't handle applied load
85   *      - 7 if checksum is incorrect
86   *      - 8 if sent instruction is out of range
87   *      - 9 if servo is overheated
88   *      - 10 if goal position is out of limit range
89   *      - 11 if input voltage is too low or too high
90   *
91   *      @param[in] id Servo ID that response is expected from
92   *      @param[out] parameters Array that should hold response data.
93   *      This must be big enough to fit all returned parameters.
94   *
95   *      @return Status code
96   */
97  uint8_t servo_receive(uint8_t id, uint8_t *parameters) {
98          uint8_t i;
99          uint8_t data[256];
```

```
100         uint8_t data_length = 0;
101
102         // Fetch bytes into an array of length data_length
103         // 0:     Start (0xff)
104         // 1:     Start (0xff)
105         // 2:     ID of servo
106         // 3:     Number of data packets (error code + parameters +
107         checksum)
108         // 4:     Error code
109         // 5-...: Parameters
110         // Last:  Checksum
111         for (i = 0; data_length == 0 || i < data_length; i++) {
112                 if (usart_read_byte(&data[i]) != 0) {
113                         // Read timed out
114                         return 1;
115                 }
116
117                 // Mark how many bytes to expect and pray that this is
118                 wrong
119                 if (i == 3) {
120                         data_length = data[i] + 4;
121                 }
122         }
123
124         // Calculate checksum and compare against received one
125         if (data[data_length - 1] != servo_calculate_checksum(2,
126         data_length - 2, data)) {
127                 return 2;
128         }
129
130         // Verify that start bytes we're correct
131         if (data[0] != 0xff || data[1] != 0xff) {
132                 return 3;
133         }
134
135         // Verify that correct servo returned data
136         if (data[2] != id) {
137                 return 4;
138         }
139
140         // Check if there was an error code
141         if (data[4]) {
142                 if (data[4] & SERVO_ERROR_INSTRUCTION) {
143                         return 5;
144                 } else if (data[4] & SERVO_ERROR_OVERLOAD) {
145                         return 6;
146                 } else if (data[4] & SERVO_ERROR_CHECKSUM) {
147                         return 7;
```

```
148                 } else if (data[4] & SERVO_ERROR_RANGE) {
149                         return 8;
150                 } else if (data[4] & SERVO_ERROR_OVERHEAT) {
151                         return 9;
152                 } else if (data[4] & SERVO_ERROR_ANGLE_LIMIT) {
153                         return 10;
154                 } else if (data[4] & SERVO_ERROR_INPUT_VOLTAGE) {
155                         return 11;
156                 }
157         }
158
159         // Read parameters into given array if not null pointer
160         if (data_length > 5 && parameters != 0) {
161                 for (i = 5; i < data_length - 1; i++) {
162                         parameters[i - 5] = data[i];
163                 }
164         }
165
166         return 0;
167 }
168 /**
169  *      Transfer data to servo. To be used with variable length functio
170  *
171  *      @param id Servo ID
172  *      @param instruction Instruction type
173  *      @param data_length Number of parameters in data
174  *      @param data Variable argument list of parameters to send as
175  obtained by `va_start()`
176  */
177 void vservo_send(
178         uint8_t id, uint8_t instruction, uint8_t data_length, va_list
179         data)
180 {
181         uint8_t i;
182
183         uint8_t packet_length = data_length + 6;
184         uint8_t packet[packet_length];
185
186         packet[0] = 0xff;            // Start byte
187         packet[1] = 0xff;            // Start byte
188         packet[2] = id;              // Servo ID
189         packet[3] = data_length + 2; // Length of data + instruction +
190         checksum
191         packet[4] = instruction;     // Instruction type (read, write,
192         ping)
193
194         for (i = 0; i < data_length; i++) {
195                 packet[i + 5] = (uint8_t)va_arg(data, int);
```

```
196             }
197
198             // Add checksum to last byte
199             packet[packet_length - 1] = servo_calculate_checksum(
200                     2, packet_length - 2, packet);
201
202             // Disable interrupts while writing data
203             cli();
204
205             // Clear buffer before sending to be sure there is not junk
206             usart_clear_buffer();
207
208             servo_enable_write();
209             for (i = 0; i < packet_length; i++) {
210                     usart_write_byte(packet[i]);
211             }
212
213             // Indicate that all bytes are sent so servo_enable_read can
214             wait for all
215             // bytes to be transmitted before changing tri-state buffer
216             usart_tx_frame();
217
218             // Enable receiving so interrupts can be handled properly
219             servo_enable_read();
220
221             // Re-enable interrupts once write is complete
222             sei();
223     }
```

C ID:n till buss

C.1 Delsystem sensor

ID	Sändning/förfrågan	Funktion [data]
1	Oanvänd	
2	Sändning	Kalibrering av linjesensor
3	Förfrågan	Linjesensordata
4	Förfrågan	Tyngdpunkt
5	Sändning	Sätt tejpreferens [ny tejpreferens]
6	Oanvänd	
7	Oanvänd	
8	Oanvänd	
9	Sändning	Sätt aktivitet [0 = linjeföljning, 1 = skanna vänster, 2 = skanna höger]
10	Sändning	Läs RFID-tag

Tabell 2: De funktioner för att reagera på bussöverföringar som sensorenheten har implementerat.

C.2 Delsystem chassi

ID	Sändning/förfrågan	Funktion [data]
0	Sändning	Nödstopp
1	Oanvänd	
2	Sändning	Arm klar [0 = plockat up, 1 = lagt ner, 2 = hittade inget]
3	Sändning	Starta linjeföljning
4	Sändning	RFID inläst
5	Oanvänd	
6	Oanvänd	
7	Oanvänd	
8	Sändning	Uppdatera manuell styrning, via CMD_CHASSI_MOVEMENT i tabell 7
9	Oanvänd	
10	Oanvänd	
11	Sändning	Sätt Kp [nytt värde på Kp]
12	Sändning	Sätt Kd [nytt värde på Kd]

Tabell 3: De funktioner för att reagera på bussöverföringar som chassienheten har implementerat.

C.3 Delsystem kommunikationsenhet

ID	Sändning/förfrågan	Funktion [data]
1	Oanvänd	
2	Sändning	Rad 1 på skärm för sensorenheten
3	Sändning	Rad 2 på skärm för sensorenheten
4	Sändning	Rad 1 på skärm för armenheten
5	Sändning	Rad 2 på skärm för armenheten
6	Sändning	Rad 1 på skärm för chassienheten
7	Sändning	Rad 2 på skärm för chassienheten
8	Sändning	Vidarebefordra beslut från chassi till PC
9	Sändning	Vidarebefordra RFID-tag till PC
10	Sändning	Vidarebefordra data från sidoskanner till PC
11	Sändning	Sätt Kp [nytt värde på Kp]
12	Sändning	Sätt Kd [nytt värde på Kd]
13	Sändning	Vidarebefordra kalibreringsvärde till PC

Tabell 4: De funktioner för att reagera på bussöverföringar som kommunikationsenheten har implementerat.

C.4 Delsystem arm

ID	Sändning/förfrågan	Funktion [data]
0	Sändning	Nödstopp
1	Sändning	Kommando till klo [0 = stäng, 1 = öppna]
2	Sändning	Signal att roboten står på station [0 = vänster, 1 = höger]
3	Sändning	Vinkel till upplockning [vinkel]
4	Sändning	x koordinat till upplockning [x]
5	Sändning	Plocka upp objekt [0 = inget hittades, 1 = föremål hittat]
6	Sändning	x position för manuell styrning [x]
7	Sändning	y position för manuell styrning [y]
8	Sändning	Vinkel för manuell styrning om positiv [vinkel]
9	Sändning	Vinkel för manuell styrning om negativ [vinkel]
10	Sändning	Gå till position satt i manuell styrning
11	Sändning	Uppdatera manuell styrning, via CMD_ARM_MOVE i tabell 7
12	Sändning	Lämna av objekt [0 = vänster, 1 = höger]

Tabell 5: De funktioner för att reagera på bussöverföringar som armenheten har implementerat.

D Protokoll för blåtand

ID-beteckning	Beskrivning	Parametrar
PKT_STOP	Indikerar att roboten ska stoppa all rörelse.	Inga
PKT_ARM_COMMAND	Kommando till arm.	Kommando (1), argument (2-10)
PKT_CHASSIS_COMMAND	Kommando till chassi.	Kommando (1), argument (2-3)
PKT_CALIBRATION_COMMAND	Kommando för kalibrering av sensor.	Sensor id (1)
PKT_LINE_DATA	Data från linjesensorn.	Individuella sensorer (1-11), flaggor (12), tyngdpunkt (13), styrutslag (14)
PKT_RANGE_DATA	Avstånd från sidoskanner.	sida (1), avstånd (2-3)
PKT_RFID_DATA	Värde på RFID-tagg	värde (1)
PKT_CHASSIS_DECISION	Beslut från chassi.	beslut (1)
PKT_PACKET_REQUEST	Förfrågan om paket från dator.	ID på packet (1)
PKT_SPOOFED_REQUEST	Simulerad förfrågan över bussen	Adress för enheten som ska ta emot förfrågan (1), ID på funktionen som ska anropas(2), metadata som ska föras med förfrågan (3-4)
PKT_SPOOFED_RESPONSE	Svar på simulerad förfrågan.	Data som returneras av förfrågan (1-2)
PKT_SPOOFED_TRANSMIT	Simulerad sändning på bussen.	Data som skickas ut på bussen. (1-2)
PKT_CALIBRATION_DATA	Värde efter kalibrering.	Värde (1)
PKT_LINE_WEIGHT	Tyngdpunkt från linjesensor.	Tyngdpunkt (1)

Tabell 6: De olika pakettyper som kan skickas mellan roboten och PC och vad de innehåller. Parametrarnas ordningsnummer i parameterföljden anges inom parentes i den sista kolumnen.

ID-beteckning	**Beskrivning**	**Parametrar**
CMD_ARM_MOVE	Arm ska röra sig.	Koordinat (1), riktning (2)
CMD_ARM_GRIP	Stäng gripklo.	Inga
CMD_ARM_RELEASE	Öppna gripklo.	Inga
CMD_ARM_PREDEFINED_POS	Rör arm till fördefinierad position.	ID på position
CMD_ARM_STOP	Stanna alla rörelser.	Inga
CMD_ARM_MOVE_POS	Rör arm till position.	x (1), y (2), vinkel (3)
CMD_CHASSIS_START	Starta linjeföljning.	Inga
CMD_CHASSIS_STOP	Stoppa linjeföljning	Inga
CMD_CHASSIS_PARAMETERS	Sätt nya värde på K_p och K_d.	K_p(1), K_d(2)
CMD_CHASSIS_MOVEMENT	Kommando till chassi att ändra sin hastighet eller riktning	Kommando (1)

Tabell 7: De olika undertyper av paket som skickas som kommandon från persondatorgränssnittet till roboten, genom PKT_ARM_COMMAND eller PKT_CHASSIS_COMMAND. Parametrarnas ordningsnummer i parameterföljden anges inom parentes i den sista kolumnen.

ID-beteckning	**Beskrivning**
DEC_PICKUP_RIGHT	Plockar upp föremål höger.
DEC_PICKUP_LEFT	Plockar upp föremål vänster.
DEC_PUT_DOWN_RIGHT	Lägger ner föremål höger.
DEC_PUT_DOWN_LEFT	Lägger ner föremål vänster.
DEC_NO_MATCH	RFID var inte korrekt.
DEC_STATION_HANDLED	Station redan behandlad.
DEC_NO_ID_FOUND	Ingen RFID-tagg hittade på stationen.
DEC_ARM_PICKED_UP	Arm har plockat upp ett föremål.
DEC_ARM_PUT_DOWN	Arm har lagt ned ett föremål.
DEC_OBJECT_NOT_FOUND	Inget föremål hittades.
DEC_UNKNOWN_ERROR	Okänt fel inträffade.
DEC_ARM_FAILED	Arm misslyckades med upplockning.
DEC_START_LINE	Startade linjeföljning.

Tabell 8: De olika besluten som roboten kan ta och vidarebefordra till persondatorgränssnittet. Dessa skickas som en parameter i PKT_CHASSIS_DECISION.