TEKNISK DOKUMENTATION

Tim Fornell

Version 1.2

Status

Granskad	JH, TF	2016-05-22
Granskad	MH	2016-06-02
Granskad	TF	2016-06-13

PROJEKTIDENTITET

Grupp 8, VT16
Linköpings tekniska högskola, ISY

Namn	Ansvar	Telefon	E-post
Tim Fornell	Projektledare (PL)	072-394 90 05	timfo734@student.liu.se
Oscar Frylemo	Dokumentansvarig (DOK)	076-209 70 77	oscfr634@student.liu.se
Martin Hinnerson	Designansvarig hårdvara (DHÅ)	072-210 43 72	marhi386@student.liu.se
Jacob Holmberg	Testansvarig (TA)	070-391 50 33	jacho926@student.liu.se
Marcus Homelius	Designansvarig sensorenhet (DS)	070-245 90 96	marho949@student.liu.se
John Stynsberg	Datorenhetsansvarig (DE)	072-057 21 54	johst529student.liu.se

Kund: Tomas Svensson, 581 83 LINKÖPING,
kundtelefon: 013-28 13 68, tomass@isy.liu.se

Kursansvarig: Tomas Svensson, 581 83 LINKÖPING,
telefon: 013-28 13 68, tomass@isy.liu.se

Handledare: Peter Johansson, 581 83 LINKÖPING
telefon: 013-28 13 45, Peter.A.Johansson@liu.se

Handledare: Anders Nilsson, 581 83 LINKÖPING
telefon: 013-28 26 35, Anders.P.Nilsson@liu.se

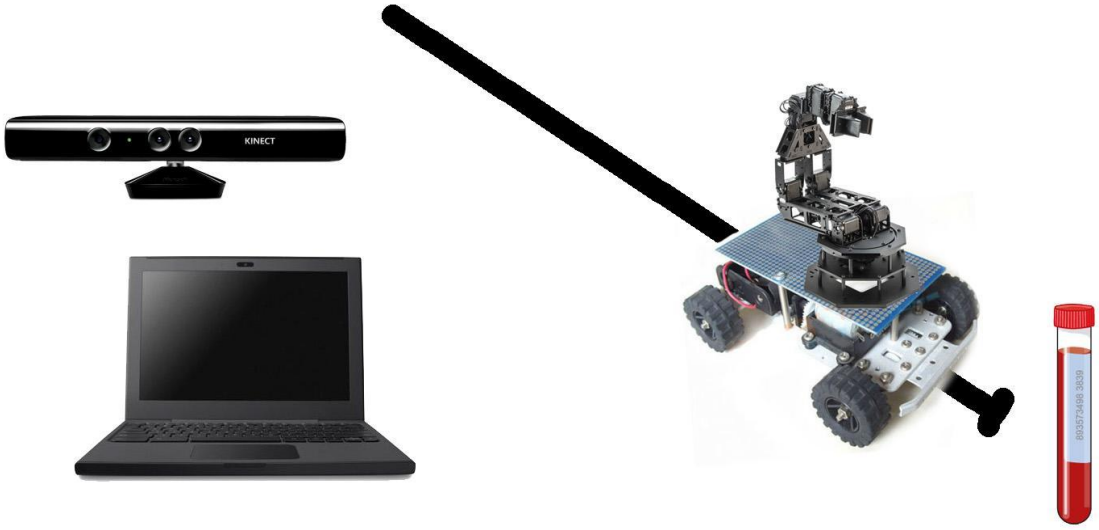
Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2016-05-24	Första versionen	Alla	JH, TF
1.1	2016-06-02	Appendix B, ändrade figurer	JH	MH
1.2	2016-06-13	Ändrade figurer	TF	MH

Innehållsförteckning

1 Inledning

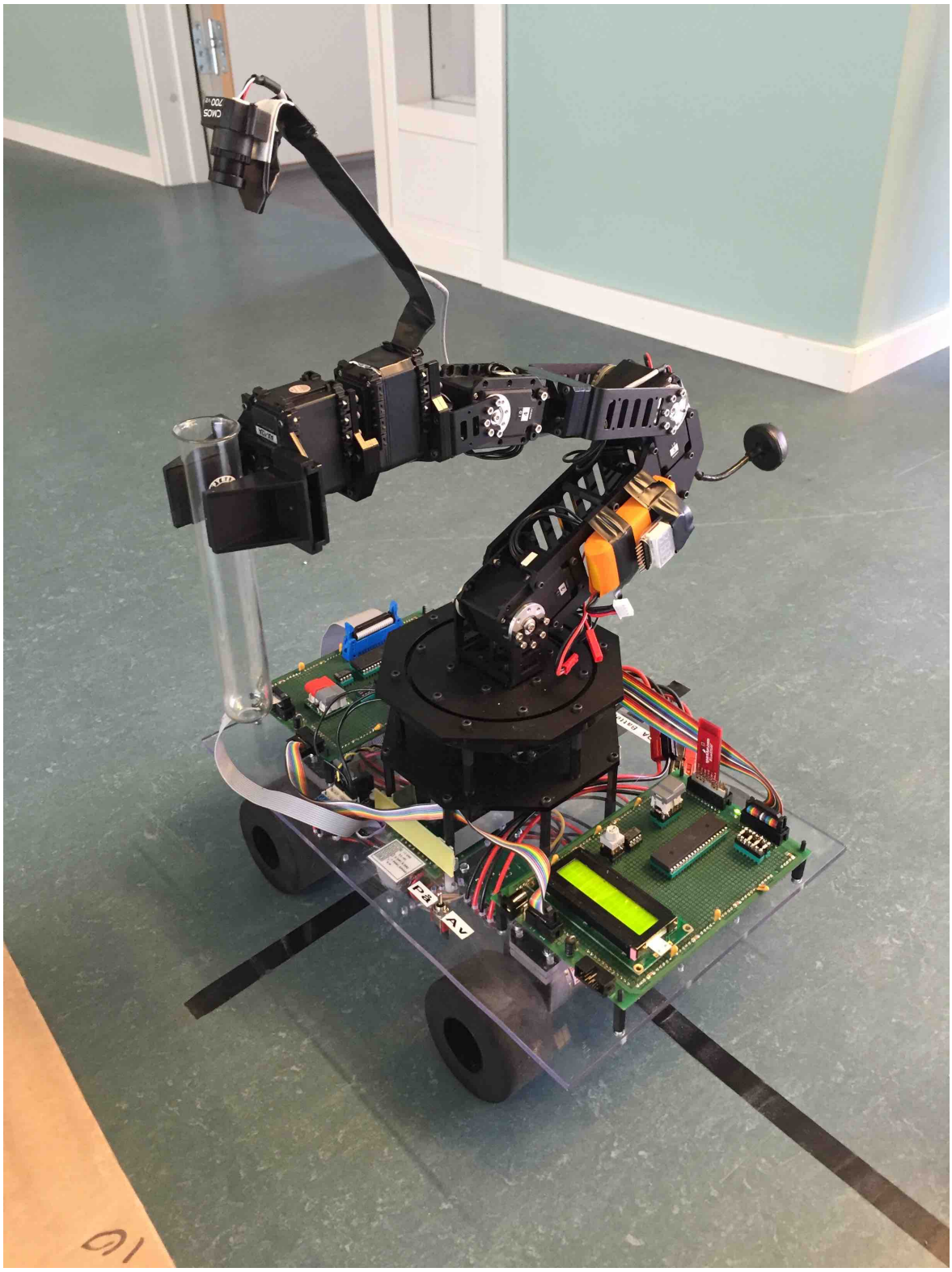
Projektet hade som syfte att utveckla en robot med Kinect-styrd arm. Systemet består av en dator med Kinect-enhet samt en robot, *se figur 1*. Roboten består av en plattform med fyra motordrivna hjul, tre moduler och en servostyrd robotarm. Tänkbara uppdrag för roboten är farliga miljöer där människan inte bör vistas, exempelvis uppdrag med hantering av farliga kemikalier i en laboratoriemiljö. Det konkreta resultatet av projektet är en robot med en arm som klarar av ett uppdrag enligt banspecifikationen.



Figur 1. Systemet i dess omgivning.

2 Produkten

Roboten kan arbeta både manuellt och autonomt i en miljö som är skadlig för människor att vistas i. Autonom manövrering sker genom att roboten med hjälp av en linjesensor följer en på förhand markerad väg. När målet, exempelvis ett laboratorieprov, är nått ska robotarmen som styrs manuellt via en Kinect-sensor användas för att utföra önskad uppgift. Se *figur 2* för den färdiga produkten.



Figur 2. Den slutliga produkten.

3 Teori

I projektet har följande teori använts.

3.1 Reglerteori

För att roboten ska kunna följa en linje används PID-reglering. Sensormodulen hämtar sensordata från reflexsensormodulen och räknar ut ett reglerfel som sedan skickas till styrmodulen. För att beräkna reglerfelet görs först en tyngdpunktsberäkning

$$\boldsymbol{k_T = \sum_k m_k k / \sum_k m_k} \quad \textbf{(1)}$$

där k är ett heltal som går från 1 till 7 och representerar varje reflexsensor och m_k är resultatet från varje sensor. Reglerfelet beräknas sedan enligt:

$$\boldsymbol{e[n] = 4 - k_T} \quad \textbf{(2)}$$

Felet blir alltså 0 då tyngdpunkten är 4 som motsvarar att reflexsensorn i mitten är över tejpen. Styrmodulen får reglerfelet från sensormodulen och beräknar en styrsignal

$$\boldsymbol{u[n] = K_P * e[n] + K_I * (e[n] + e[n-1] + e[n-2] + ..) + K_D * (e[n] - e[n-1])} \quad \textbf{(3)}$$

där K_P, K_I och K_D är konstanter, $e[n]$ är det aktuella reglerfelet och $e[n-1]$ är det förra reglerfelet. Om $u[n] = 0$ ska roboten köra rakt fram, om $u[n] < 0$ ska den svänga åt vänster och om $u[n] > 0$ ska den svänga åt höger. Ju större $u[n]$ är desto snabbare kommer roboten svänga.

3.2 Invers kinematik

Invers kinematik avser användandet av kinematikekvationer för att bestämma hur mycket respektive led ska böjas för att en robots ”manipulator” ska uppnå en önskad position. Roboten kan t.ex. vara en robotarm med en gripklo som ”manipulator”. Invers kinematik för robotarmen kan dessutom förenklas från ett 3-dimensionellt till ett 2-dimensionellt problem då den endast har en led som möjliggör 3-dimensionella rörelser.

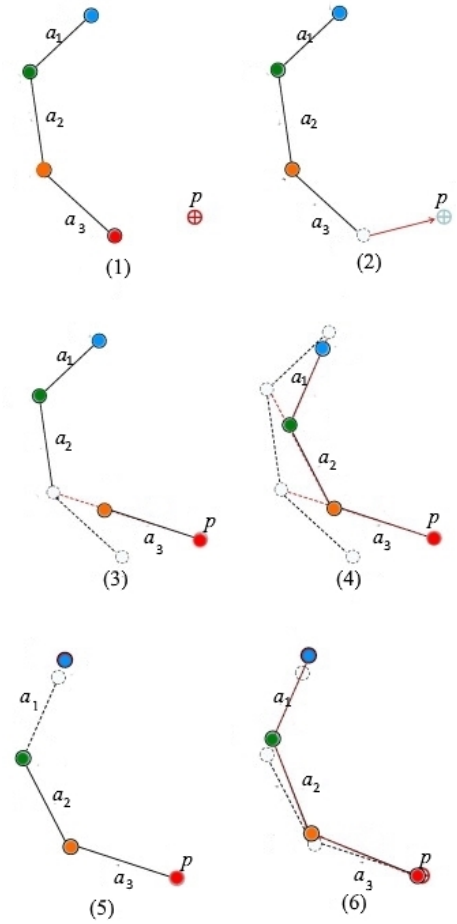
Fördelarna som invers kinematik presenterar är att endast den önskade slutpositionen för armen behöver kännas till. Vid fjärrstyrning med en Kinect-enhet räcker det med en slutposition för robotens manipulator detekteras. Datorenheten använder sedan sin programmerade algoritm för invers kinematik för att beräkna ledernas vinklar för att nå denna position. Den utför nya beräkningar för varje slutposition utifrån sitt nuvarande läge på servona, istället för att använda förbestämda slutvinklar.

Algoritmen som valdes för att styra armen heter FABRIK [1]. Denna algoritm tar in en önskad slutposition för armen för att sedan genom iteration flytta robotarmen dit. Iterationen för att förflytta manipulatorn hos en arm med tre ($n = 3$) leder till en punkt *P* sker enligt följande steg:

1. Kontrollera om den önskade punkten $P = (x, y)$ ligger innanför räckvidden hos robotarmen. Robotarmens räckvidd räknas ut enligt följande ekvation:

$$\|P\| \leq \sum_{i=1}^{m} a_i$$

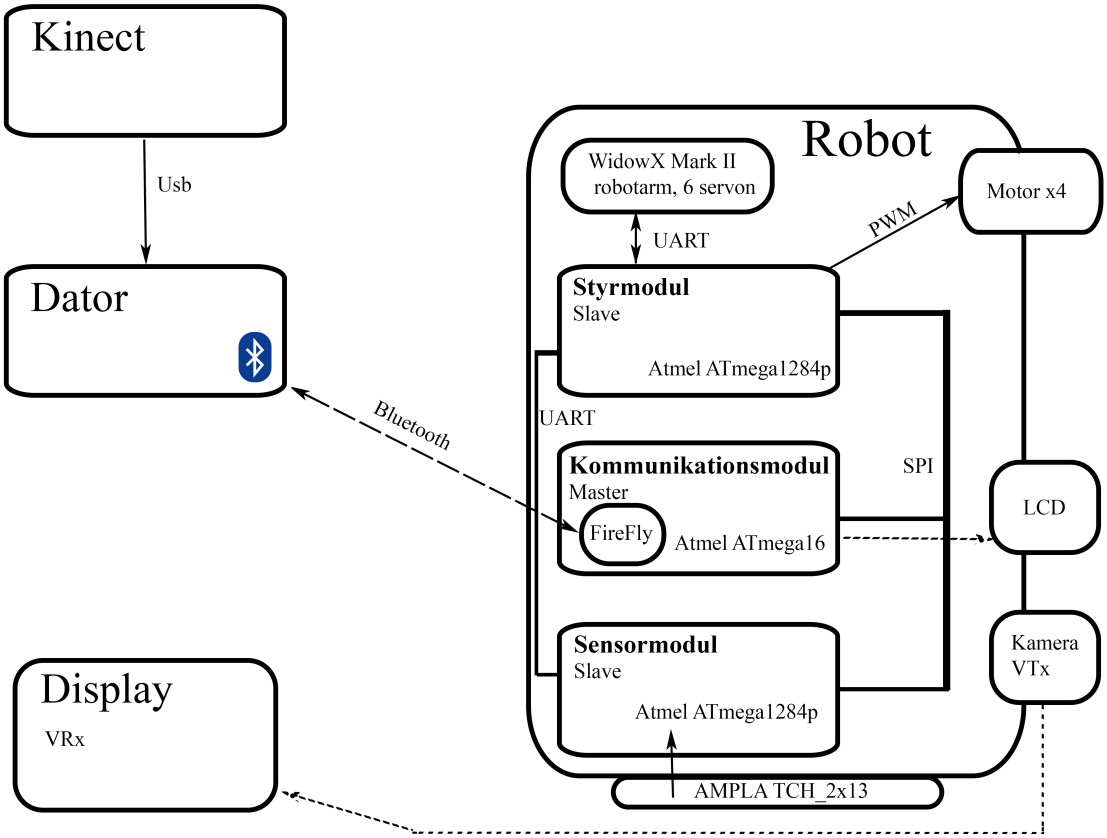
   där $m = n - 1$ och a_i är avståndet mellan respektive led.
2. Om den ligger utanför är det omöjligt att nå punkten. (Beroende på tillämpning hanteras detta fall olika)
3. Flytta ändpunkten på armen till den önskade slutpunkten (Steg 2 i *figur 3*).
4. Flytta sedan nästa led så att den ligger på linjen mellan sin gamla punkt och den föregående ledens nya punkt (Steg 3 i *figur 3*).
5. Upprepa flyttandet tills roten flyttats.
6. Nu görs samma sak fast den första flytten är att roten flyttas tillbaka till sin ursprungsposition, och låter sedan resten av lederna följa på samma sätt som i steg 4 (Steg 5 i *figur 3)*.
7. Upprepa processen framåt och bakåt tills slutleden är tillräckligt nära den önskade punkten eller, i fallet där punkten ligger utanför armens räckvidd, jämför förra iterationens kloposition och den nuvarande. Om den nya är sämre än den förra, avbryt algoritmen.



Figur 3. Iterationer av FABRIK.

4 Systemöversikt

Systemet är uppdelat i tre delsystem; datorenheten, Kinect-enheten och robotenheten. Kinect-enheten och datorenheten är ihopkopplade via USB. Kommunikation mellan datorenheten och robotenheten sker trådlöst via Bluetooth. Systemet illustreras i *figur 4,* där svart indikerar att det är ett krav **1** och rött krav **2** (görs i mån av tid).



Figur 4. En överblick av systemet.

Kinect-enheten har som uppgift att läsa av koordinater för användarens arm och sedan skicka dessa till datorenheten.

Datorenheten används för att skicka kommandon och parametrar, samt att ta emot data (t.ex. styrbeslut och sensorvärden) från robotenheten. Den har ett användargränssnitt där dessa värden illustreras. Datorenheten behandlar även data från Kinect-enheten för att kunna vidarebefordra detta till styrmodulen vid styrning av robotarmen.

Kommunikationsmodulen har i uppgift att distribuera information till datorenheten, styrmodulen och sensormodulen.

Sensormodulen ansvarar för insamling av data från omgivningen, detta görs med hjälp av en reflexsensormodul. Reflexsensormodulen är placerad längst fram på undersidan av robotplattformen och består utav 11 reflexsensorer varav 7 används.

Styrmodulen har som uppgift att kontrollera robotenhetens servon, som är lokaliserade i robotarmen och hjulen. Beroende på om det är armen eller hjulen som styrs kommer styrmodulen att förses kontinuerligt med data antingen från sensormodulen eller datorenheten via kommunikationsmodulen. Data som tas emot tolkas och används för styrbeslut.

5 Kinect-enhet

Kinect-enheten, är en avancerad sensor innehållande kameror, mikrofoner och en accelerometer. Den innehåller också mjukvara som kan bearbeta skelett-data, färg och djup. Kinect innehåller följande hårdvarukomponenter:

• En RGB-kamera med möjlighet att ta bilder i en upplösning på högst 1280x960.

• En IR sändare samt IR djupsensor för att kunna få ett djup i bilderna av rummet.

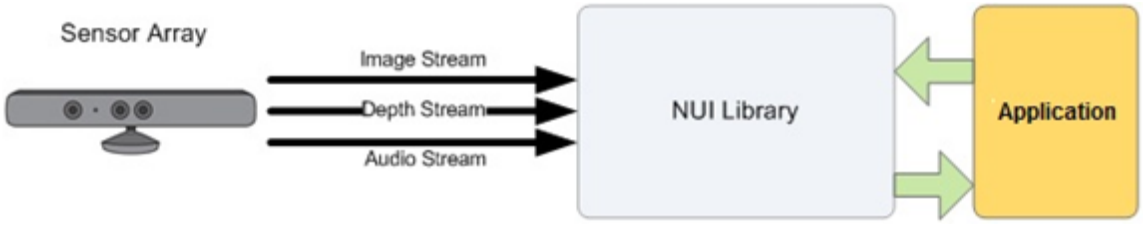
• Fyra stycken mikrofoner. För att bland annat kunna avgöra var ljudkällan i rummet ligger.

• En accelerometer för att kunna avgöra Kinect-enhetens läge.

I projektet används Kinect-enheten för att få fram koordinater i ett rum för armens leder (axelleden, armbågsleden, handleden och handen), därför behöver endast Kinect-enhetens skelett-data användas. Koordinaterna anges i ett kartesiskt koordinatsystem där Kinect-enhetens position utgör origo. Z-axeln anger avståndet bort från enheten till användaren.

5.1 Kommunikation med andra system

Utvecklings miljön (The SDK, Software Development Kit) för Kinect är uppbyggt enligt *figur 5*. Där sensorn skickar information om bild, djup och ljud till NUI:n (Natural User Interface) som sedan kan användas av applikationer. Datorn måste använda operativsystemet Windows.



Figur 5. Översiktlig bild av Kinectsensorn.

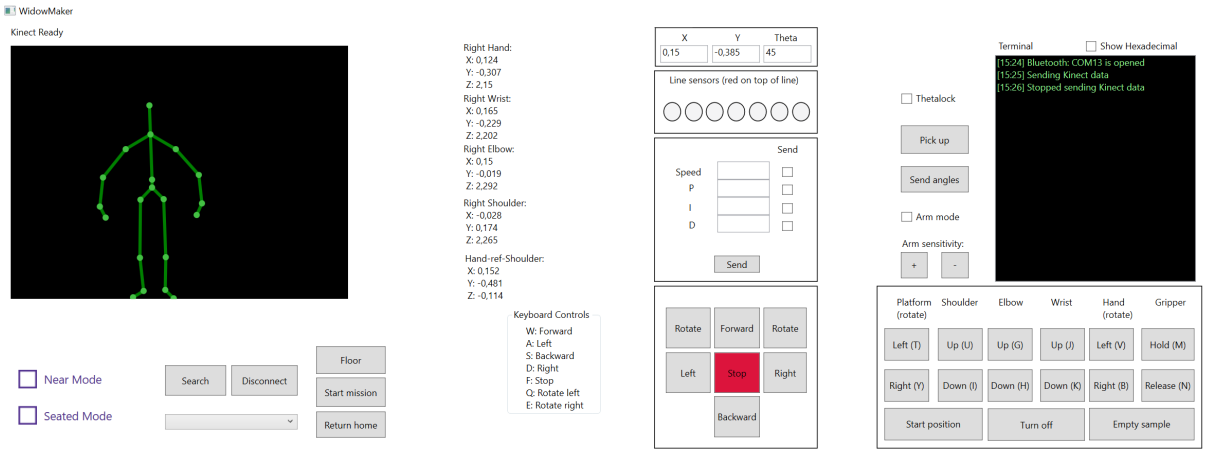
Koordinaterna skickas från Kinect-enheten till användargränssnittet i datorn som skickar det vidare till kommunikationsmodulen som skickar det vidare till styrmodulen.

6 Datorenhet

Datorenheten har ett användargränssnitt som kopplar samman Kinect-enheten (via USB) och robotenheten. Datorn använder operativsystemet Windows. Via användargränssnittet kan användaren skicka kommandon till roboten, styrningen kan ske autonomt eller manuellt. Vid autonom styrning skickas ett kommando från datorn som säger till roboten att följa en markerad bana (enligt banspecifikation) fram till ett markerat slut.

6.1 Användargränssnitt

Via det användargränssnitt som finns kan användaren koppla upp sig mot robotplattformen för att skicka och ta emot data, se sensorvärden och kunna kalibrera linjesensorn. Det är även detta program som har i uppgift att tolka data och sensorvärden ifrån Kinect-enheten. Det har utvecklats med programmeringsspråket C#. Användargränssnittet ser ut enligt följande, *se figur 6.*



Figur 6. En översikt av det grafiska gränssnittet.

Roboten kan manövreras i två lägen, autonomt och manuellt. Vid manuell styrning används ett tangentbord. Vid autonom styrning följer roboten en markerad bana *(se banspecifikation)* och manövrerar så att den inte lämnar linjen. Autonom styrning initieras via ett kommando i användargränssnittet. I både manuell och autonom styrning kontrolleras robotarmen med Kinect-enheten samt ett tangentbord.

Vid autonom och manuell styrning av roboten visar användargränssnittet sensordata och styrbeslut från roboten. Sensordata representeras i form av lampor där tänd lampa motsvarar att en sensor detekterat tejpen och styrbesluten visas i terminalen. Det är möjligt att kalibrera linjesensorerna samt ändra parametrar för reglering via användargränssnittet.

6.2 Kommunikation

Datorenheten kommunicerar med robotplattformen via Bluetooth och med Kinect-enheten via USB. Vid kommunikation med roboten skickas data i båda riktningarna. Datorenheten skickar kommandon när användaren avser göra detta, roboten skickar kontinuerligt data i form av sensorvärden och styrbeslut till datorn. Mellan datorenheten och Kinect-enheten skickas data endast i en riktning; från Kinect-enheten. Data skickas då Kinect-enheten har identifierat ett objekt som den kan följa med hjälp av sina sensorer. Eftersom FireFly följer standard protokollet 115200 bps, 8-N-1 begränsas denna kommunikation inte eftersom datastorleken kommer vara väldigt liten i jämförelse.

6.3 Implementation

Gränssnittet som utvecklats för detta projekt bygger vidare på Johan Hydéns [1] program. Funktionerna som denna programvara använder faller under ett par olika kategorier då många är snarlika i sina uppgifter. Observera att bara funktioner som blivit modifierade eller tillagda sedan Hydéns program nämns nedanför.

6.3.1 Knappfunktioner

Alla knappar som styr roboten har en funktion som skickar en karaktär via COM-porten till roboten. Denna karaktär känns igen av roboten som sedan utför motsvarande kommando.
Ex: Left_Button_Click skickar karaktären som motsvarar ’sväng vänster’ till roboten.

Funktionerna som har denna form är:

Forward_button_Click	Backward_ button_Click	Right_button_Click	Left_button_Click
Stop_button_Click	Rotate_Left_ button_Click	Rotate_Right_ button_Click	Platform_Left_ button_Click
Platform_Right_ button_Click	Shoulder_Up_ button_Click	Shoulder_Down_ button_Click	Elbow_Up_ button_Click
Elbow_Down_ button_Click	Wrist_Up_ button_Click	Wrist_Down_ button_Click	Hand_Left_ button_Click
Hand_Right_ button_Click	Gripper_Hold_ Button_Click	Gripper_Release_ Button_Click	Turn_off_button_Click
Start_position_ button_Click	Empty_Sample_ Button_Click	Arm_speed_increase_ button_Click	Arm_speed_decrease button_Click
Send_angles_Click	Pick_up_Click	Start_mission_Click	Return_home_Click
Calibrate_Click			

Knappfunktionerna som gör mer än dessa är följande:

- SearchButton_Click – Skriver ut tillgängliga COM-portar för användaren
- ConnectionButton_Click – Försöker ansluta till den valda COM-porten.
- Send_Click – Hämtar vinklar/regulatorparametrar och använder kommunikationsfunktionerna för att skicka dessa.

6.3.2 Beräkningsfunktioner

Dessa funktioner använder Kinect-datan och behandlar denna på olika sätt.

- PrintCoordinates – Skriver ut koordinaterna för skelettets högra arm. Den tar även genomsnittet av de senaste 10 mätvärdena och använder denna för att beräkna koordinaterna och rotationen som ska skickas till CalculateAngles.
- CalculateAngles – Tar in koordinaterna och rotationen beräknade av PrintCoordinates och beräknar genom invers kinematik vilka vinklar som robotarmen ska ha. Denna beräkning sker genom att kalla på ett MatLab-skript. Den mappar även om vinklarna till formatet som matas till servona.

6.3.3 Kommunikationsfunktioner

Dessa funktioner används i kommunikationen till roboten.

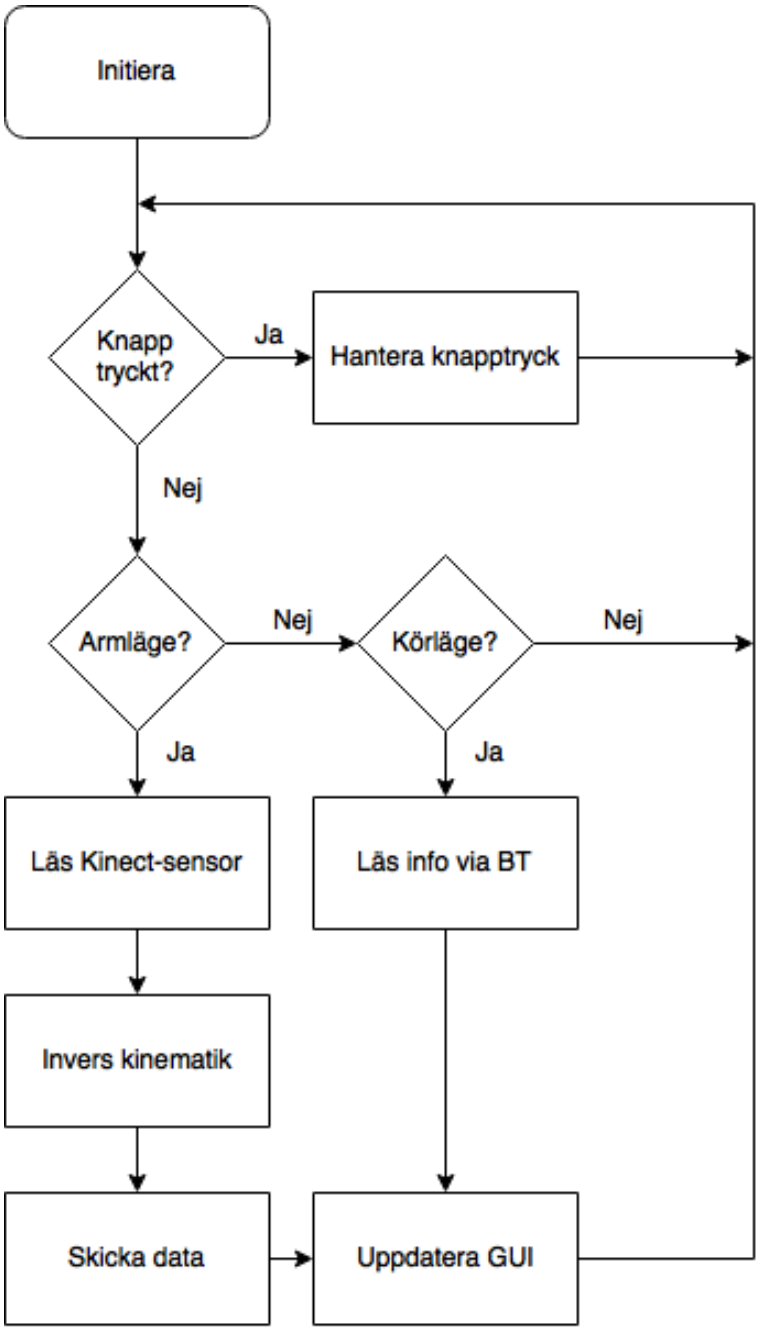
- SendAngles – Gör om vinklarna till bytes och skickar.
- SendMove – Skickar ett kommando.
- SendStartbit – Skickar en igenkänningsbyte.
- SendSpeed – Skickar hastigheten inmatat i fönstret.
- SendParameter – Skickar regulatorparameter.
- SendValue – Används för övriga godtyckliga värden.
- inPort_DataRecieved – Funktion som tar emot data från roboten och kallar på processMessage för att hantera informationen.
- processMessage – Tar in datan och lägger informationen på en buffer. Beroende på datans karaktär, kallar på lämplig funktion för att förvalta den.
- DisplayInfo – Skriver ut informationen i terminalen.
- DisplayAngles – Tolkar vinklarna och skriver ut dem i terminalen.
- ChangeLamps – Om information mottagits från reflexsensorn, används denna funktion för att tända och släcka lamporna i fönstret.

6.3.4 Övriga

Övriga intressanta funktioner

- Map_Angles – Mappar om ett värde ifrån ett intervall till ett annat.
- Out_timer_tick – En funktion som kallas 30 gånger per sekund och kallar på CalculateAngles och SendAngles om roboten är i arm-läge.

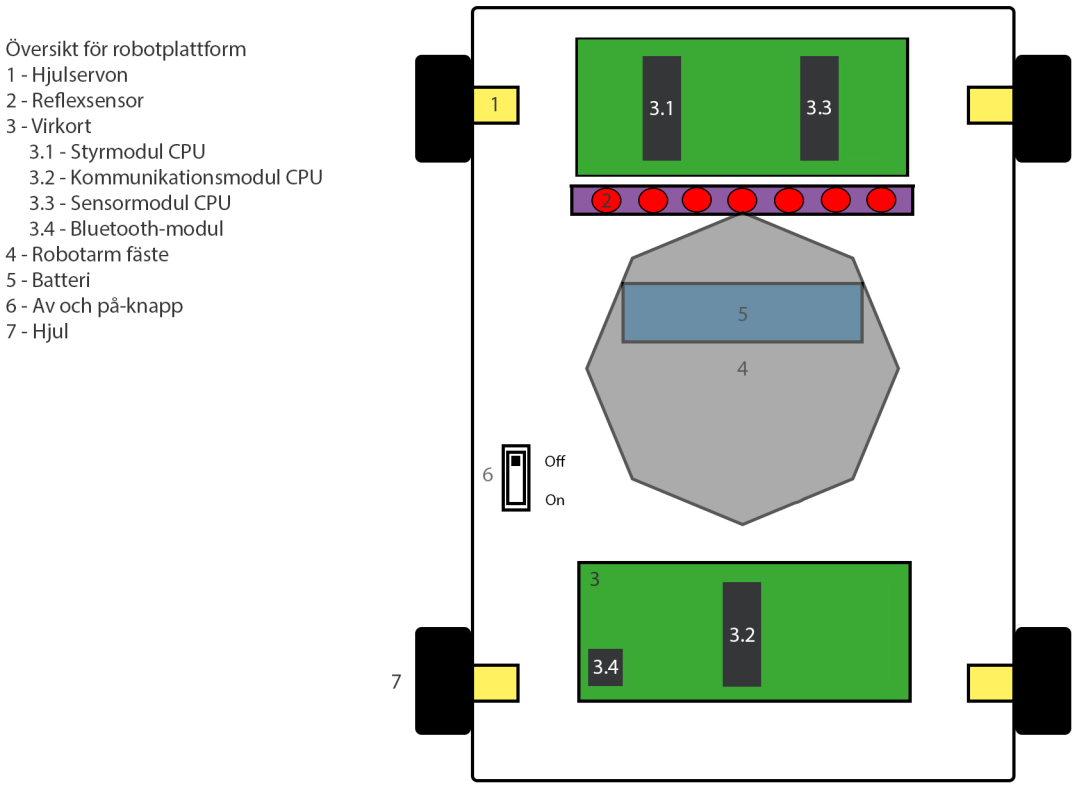
Flödesschema för datorenheten är enligt följande, *se figur 7*, på nästa sida.



Figur 7. Flödesschema för datorenheten.

7 Robotenhet

Robotenheten består av 3 moduler; styrmodul, kommunikationsmodul och sensormodul. På roboten sitter en robotarm av typen WidowX Mark II, 4 stycken motorer samt en reflexsensormodul. Roboten har även en LCD skärm som visar sensordata. Kommunikation mellan robotenheten och datorenheten sker via Bluetooth. Robotplattformen ser ut enligt *figur 8*.

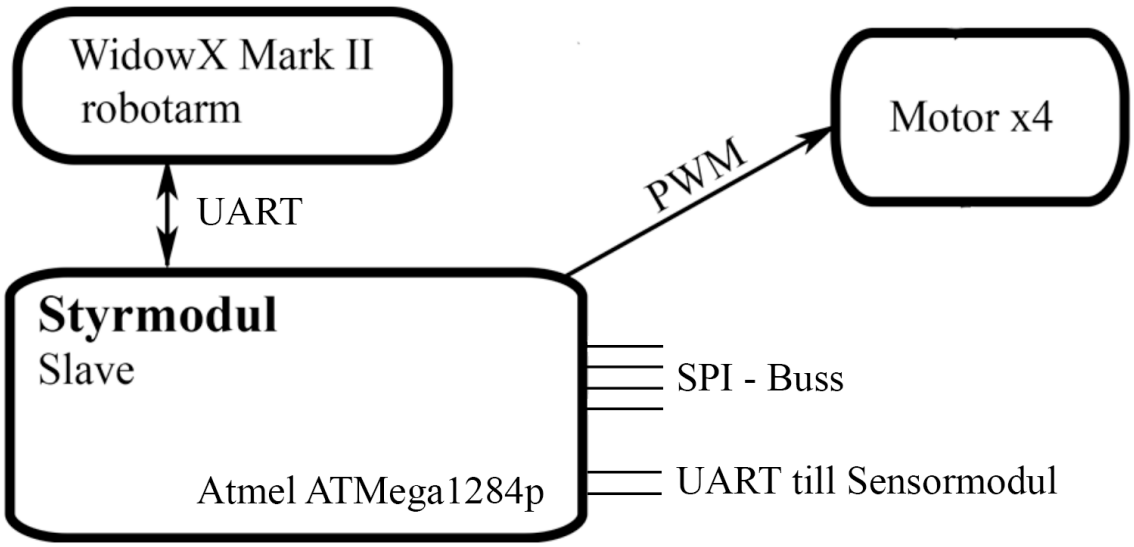


Figur 8. Översikt av robotenheten.

7.1 Styrmodul

Styrmodulen har i uppgift att kontrollera alla robotens rörelser, vilket innebär att den kontrollerar all rörelse av robotarmen samt all rörelse av hjulen. För att styra robotarmen tar styrmodulen emot data från datorenheten i form av positioner dit armen önskas flyttas. Efter detta räknar styrmodulen ut acceleration för respektive servo så att alla servon når den önskade positionen samtidigt.

För att styra robotplattformens hjul tar styrmodulen emot data från sensormodulen i form av värden från linjesensorn. Utifrån dessa värden ska styrmodulen fatta beslut om vart robotplattformen befinner sig i förhållande till linjen. För att sedan utifrån detta beräkna hur hjulen ska regleras för att plattformen ska fortsätta följa linjen på önskat vis. Styrmodulen är uppbyggd enligt *figur 9*.



Figur 9. Översikt av styrmodulen.

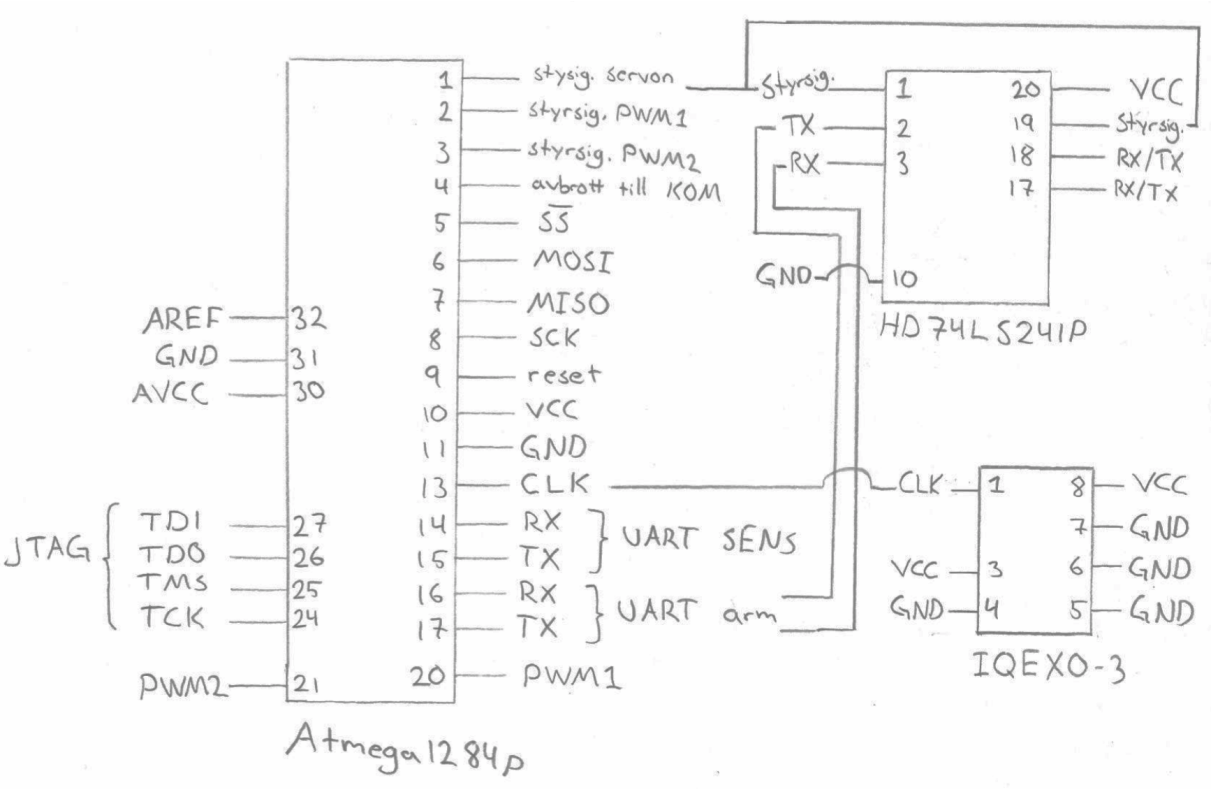
7.1.1 Komponentbudget

Styrmodulen består av följande komponenter:

- Atmel ATmega1284p mikrokontroller
- 4 st. PWM-styrda elmotorer
- WidowX Mark II robotarm av Trossen Robotics bestående av 2 st. MX-64, 2 st. MX-28 och 2 st. AX-12A Dynamixel Servon
- IQEXO-3 klockpulsgenerator

7.1.2 Kopplingsschema

I *figur 10* finns kopplingsschema för styrmodulen.



Figur 10. Kopplingsschema för styrmodulen.

- PD0-1 (RXD0, TXD0), dvs *UART* kopplas till sensormodulen
- PD2-3 (RXD1, TXD1), dvs *UART* kopplas till robotarmen
- PD4-5 (OC1B, OC1A) kopplas till motorernas PWM
- PD6-7 är motorernas riktningssignaler
- PB4-7 (SS, MOSI, MISO, SCK) kopplas till bussen.
- PC2-5 (TDI, TDO, TMS, TCK) kopplas till JTAG-kedjan.
- XTAL1 kopplas till en klockpulsgenerator.
- RESET ska anslutas för att underlätta felsökning.

7.1.3 Mjukvara

Mjukvaran kan delas upp efter styrmodulens två uppgifter, att styra hjulen så att linjen följs och att styra robotarmen. Alla styrbeslut ska också skickas via bussen till kommunikationsmodulen för att kunna vidarebefordras till datorenheten.

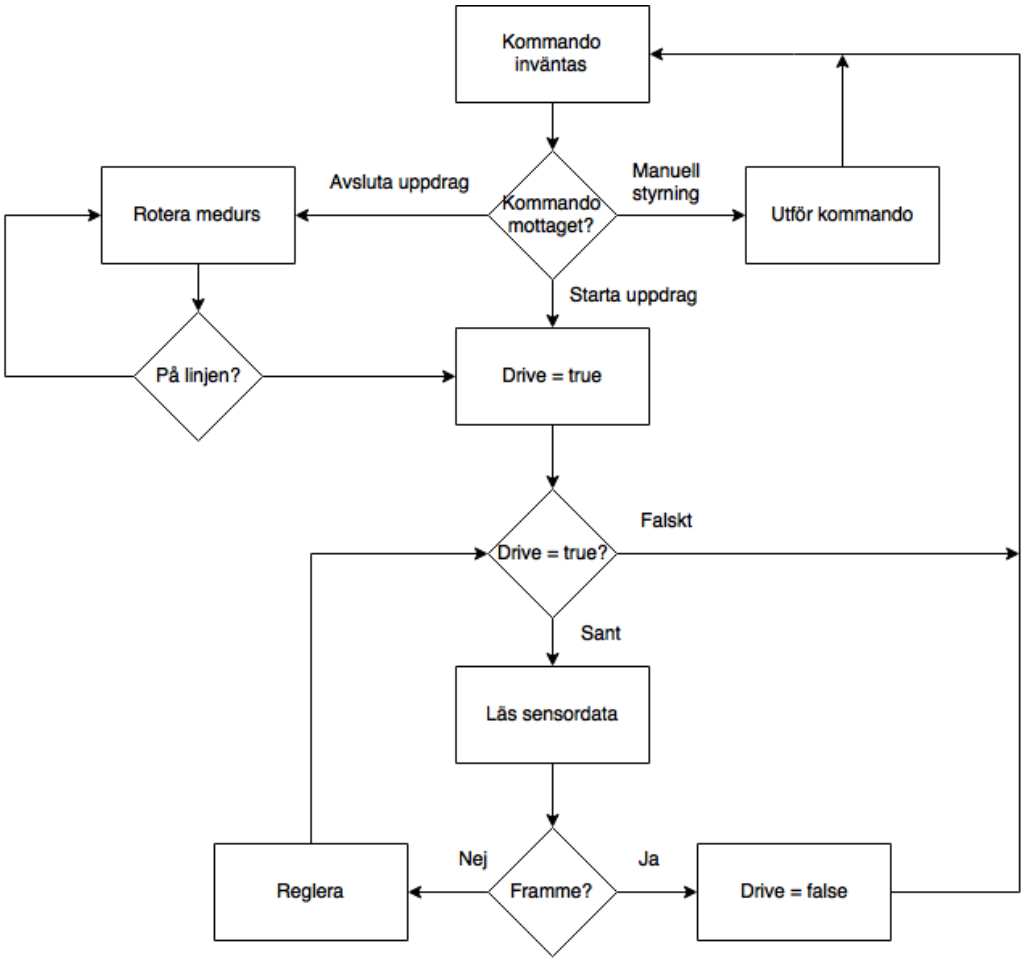
7.1.3.1 Styrning av hjul

Styrning av hjulen kan ske på två sätt; autonomt och manuellt. Autonom styrning initieras genom ett kommando i form av ”starta uppdrag” alt. ”avsluta uppdrag” och manuell styrning initieras genom enstaka kommandon som t.ex. ”kör framåt” alt. ”rotera vänster”.

Vid styrning av hjulen arbetar styrmodulen enligt flödesschemat i *figur 11*. Styrmodulen väntar på att ett kommando skickas. Om kommandot som fås innebär manuell styrning ska kommandot tolkas och sedan utföras enligt, på förhand, definierade algoritmer.

Om däremot kommandot som fås är ”starta uppdrag” eller ”avsluta uppdrag” ska den autonoma styrningen startas. Beroende på vilket av dessa kommandon som fås ska robotplattformen agera olika. Vid ”avsluta uppdrag” ska robotplattformen rotera medurs tills den detekterar tejpen. Efter detta kommer kommandona ”avsluta uppdrag” och ”starta uppdrag” ha samma flödesschema.

Denna del av flödesschemat ska se till att robotplattformen följer linjen på marken. Detta görs med hjälp av PD-reglering även om möjligheten att implementera en I-del från datorenheten finns. Styrmodulen kommer först kolla om den har ”klartecken” att köra genom att kontrollera variabeln *drive.* Om den har klartecken att köra hämtar den data från sensormodulen i form av sensorvärden, sensorvärdena kommer tala om för styrmodulen vart robotplattformen är i förhållande till linjen. Utifrån dessa sensorvärden ska styrmodulen avgöra om robotplattformen är framme vid en start/slut-markering eller om hjulen ska regleras och i sådant fall hur de ska regleras. Om kommandot ”Stopp” fås under körning ska *drive* sättas till ”false” och körningen kommer då att avbrytas.



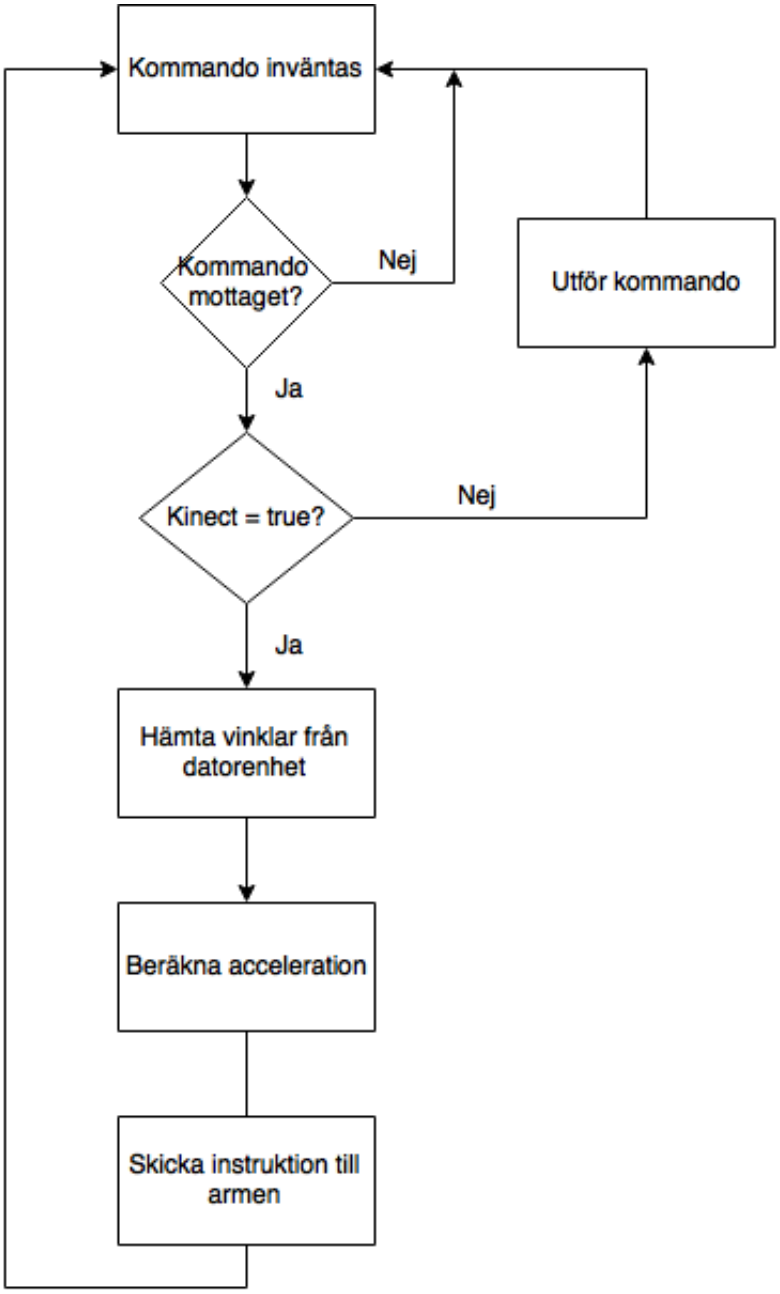
Figur 11. Flödesschema för styrning av hjul.

7.1.3.2 Styrning av arm

Styrningen av armen kan ske på två sätt; manuellt via kommandon eller via Kinect. I användargränssnittet är det möjligt att välja i vilket läge robotarmen ska styras.

Om armen önskas styras manuellt via kommandon kommer styrmodulen att vänta på att ett kommando skickas. Detta kommando kan vara t.ex. ”rotera klo” eller ”vrid arm moturs”, styrmodulen kommer då utföra detta kommando enligt, på förhand, bestämda algoritmer.

Om armen däremot önskas styras via Kinect-enheten kommer styrmodulen arbeta enligt flödesschemat i *figur 12*. Datorenheten hämtar sensordata i form av koordinater för handen från Kinect-enheten för att sedan via ett MATLAB skript beräkna vinklarna för servona med hjälp av invers kinematik. Dessa vinklar skickas sedan från datorenheten via kommunikationsmodulen till styrmodulen som utifrån dessa vinklar beräknar en acceleration för att uppnå en mjuk förflyttning.



Figur 12. Flödesschema för styrning av robotarm.

7.2 Kommunikationsmodulen

Kommunikationsmodulens huvudsakliga uppgift är att behandla all kommunikation till och från roboten, den är uppbyggd enligt *figur 13*. Data skickas till och från kommunikationsmodulen mellan de andra modulerna med ett SPI-protokoll, medan kontakt mellan kommunikationsmodul och dator sker via Bluetooth.

7.2.1 SPI

SPI är ett Master-Slave protokoll, kommunikationsmodulen är masterenhet medan sensor- och styrmodul är slav-enheter. Masterenheten bestämmer vilken modul den vill kommunicera med, detta resulterar i att kommunikationsmodulen blir central i systemet.

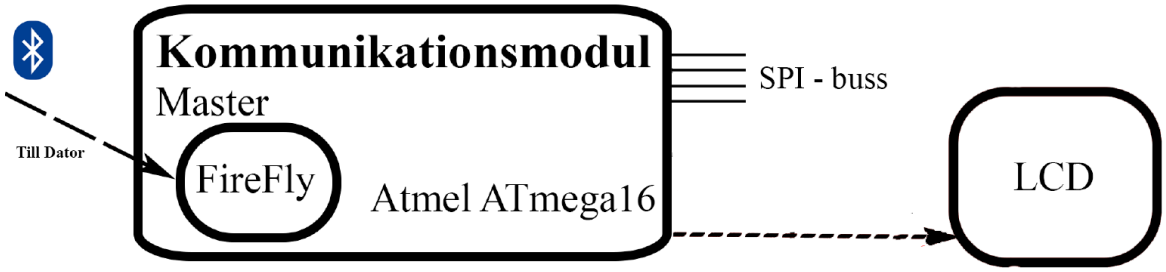
När masterenheten vill sända eller ta emot data från en slavenhet sätts tillhörande slave-select till logiskt 0. Sedan placerar masterenheten de bitar som ska skickas i sitt skiftregister, samma procedur sker för slavenheten. En klockpuls genereras från masterenheten vilket resulterar i att bitarna i masterenhetens skiftregister skickas via *Master Output Slave Input* (MOSI) till slavenhetens skiftregister. Bitarna i slavenhetens skiftregister skickas på motsvarande sätt via *Master Input Slave Output* (MISO) till masterenhetens skiftregister. När en överföring är färdig kommer innehållet i masterenhet och slavenhet ha bytt plats med varandra.

I vissa fall behöver även slavarna kunna initiera en överföring vilket är en begränsning i SPI då endast en master kan göra detta. Detta löstes med ett externt avbrott från varje slav till kommunikationsmodulen som säger att data finns att hämta.

Data kan skickas via bussen byte för byte eller som hela ord (flera sammanhängande bytes i följd). Om ord ska skickas börjas överföringen med en startbyte vilket följs av ett fördefinierat antal bytes av data.

7.2.2 Bluetooth

Kommunikationsmodulen kommunicerar med datorn via Bluetooth. Detta möjliggörs med en FireFly-modul, som parkopplas med datorn och är kopplad till kommunikationsmodulens UART, *se figur 13*.



Figur 13. Översikt av kommunikationsmodulen.

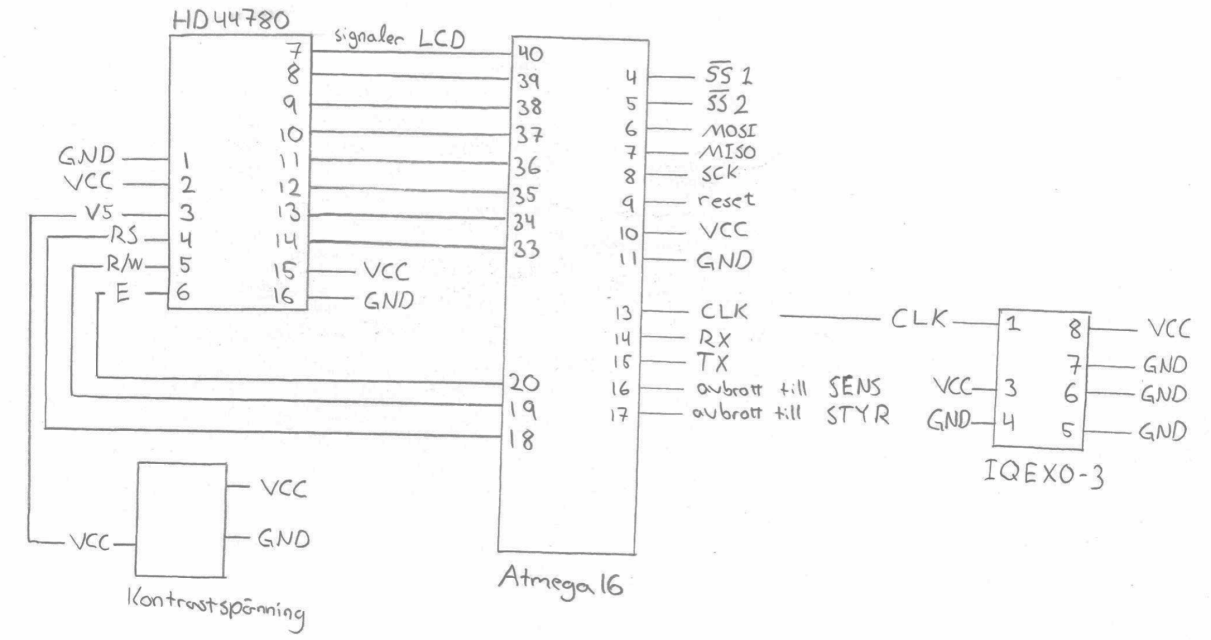
7.2.3 Komponentbudget

Kommunikationsmodulen består av följande komponenter:

- Atmel ATmega16 mikrokontroller
- FireFly Bluetoothmodul
- IQEXO-3 Kristalloscillator
- G121C LCD-skärm

7.2.4 Kopplingsschema

Nedan finns ett kopplingsschema för kommunikationsmodulen, *se figur 14*.

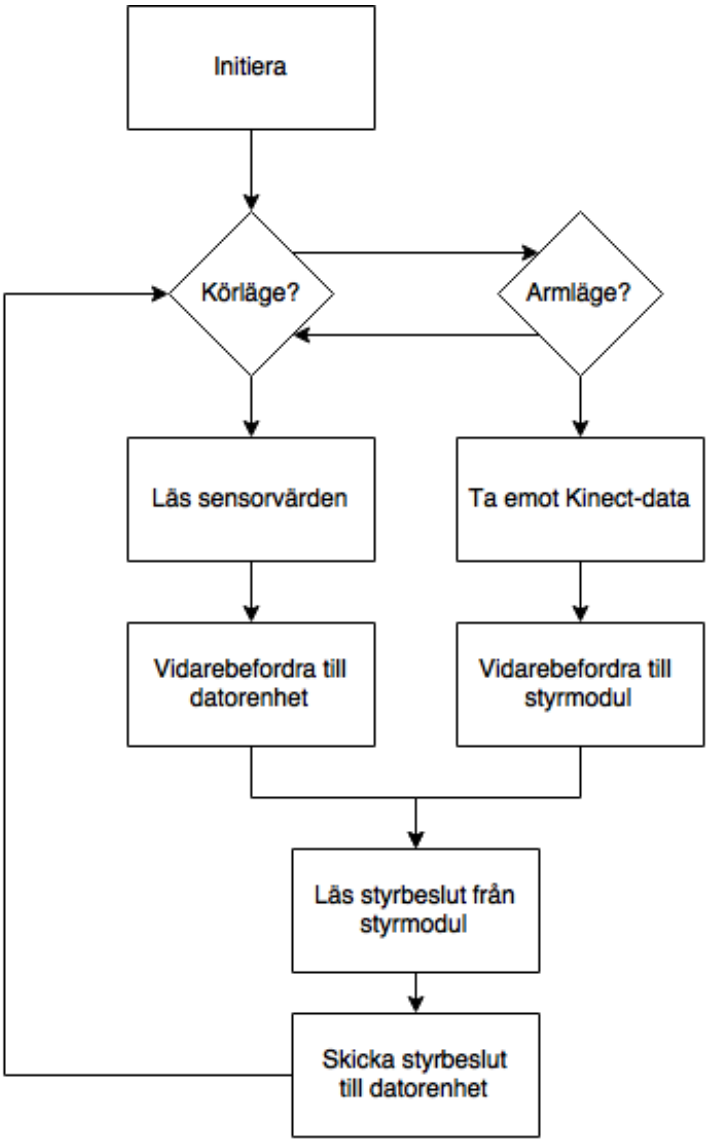


Figur 14. Kopplingsschema för kommunikationsmodulen.

- PB4-7 (SS, MOSI, MISO, SCK) kopplas till bussen.
- RXD och TXD kopplas via UART till FireFly-modulen.
- PC2-5 (TDI, TDO, TMS, TCK) kopplas till JTAG-kedjan.
- XTAL1 kopplas till en klockpulsgenerator.
- RESET ska anslutas för att underlätta felsökning.

7.2.5 Mjukvara

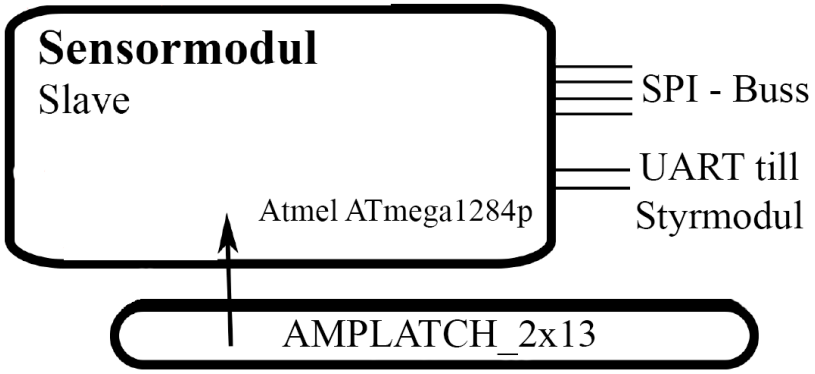
Denna modul använder sig kontinuerligt av huvudloopen och avbrott för att hantera kommunikation med datorn och övriga moduler. Kommandon från dator skickas vidare till respektive modul som t.ex. ”Kalibrera Sensor” eller ”Starta Uppdrag”. Sensorvärden från sensormodulen och styrbeslut från styrmodulen skickas vidare till datorenheten, för flödesschema se *figur 15*. Kommandon från datorenheten genererar avbrott och avbrottsrutinen kommer föra vidare kommandot till rätt enhet.



Figur 15. Flödesschema för huvudloopen i kommunikationsmodulen.

7.3 Sensormodulen

Sensormodulens uppgift är att ta emot och behandla sensordata från robotens sensorer. Att behandla sensordata innebär att översätta spänningar till mätdata som kan användas av andra moduler på roboten. Sensormodulen är uppbyggd enligt *figur 16.*



Figur 16. Översikt av sensormodulen.

7.3.1 Kommunikation

Sensormodulen är kopplad till kommunikationsmodulen via SPI-bussen. Där skickas sensordata till kommunikationsmodulen för att sedan skickas vidare till datorn. Sensormodulen kommer även att ta emot kommandon från datorn via denna buss som exempelvis kalibrering av sensorn. Utöver detta finns även en direktbuss till styrmodulen, detta för att skicka reglerfelet som regulatorn är beroende av. Förklaring av detta finns under punkt 7.2.

7.3.2 Komponentbudget

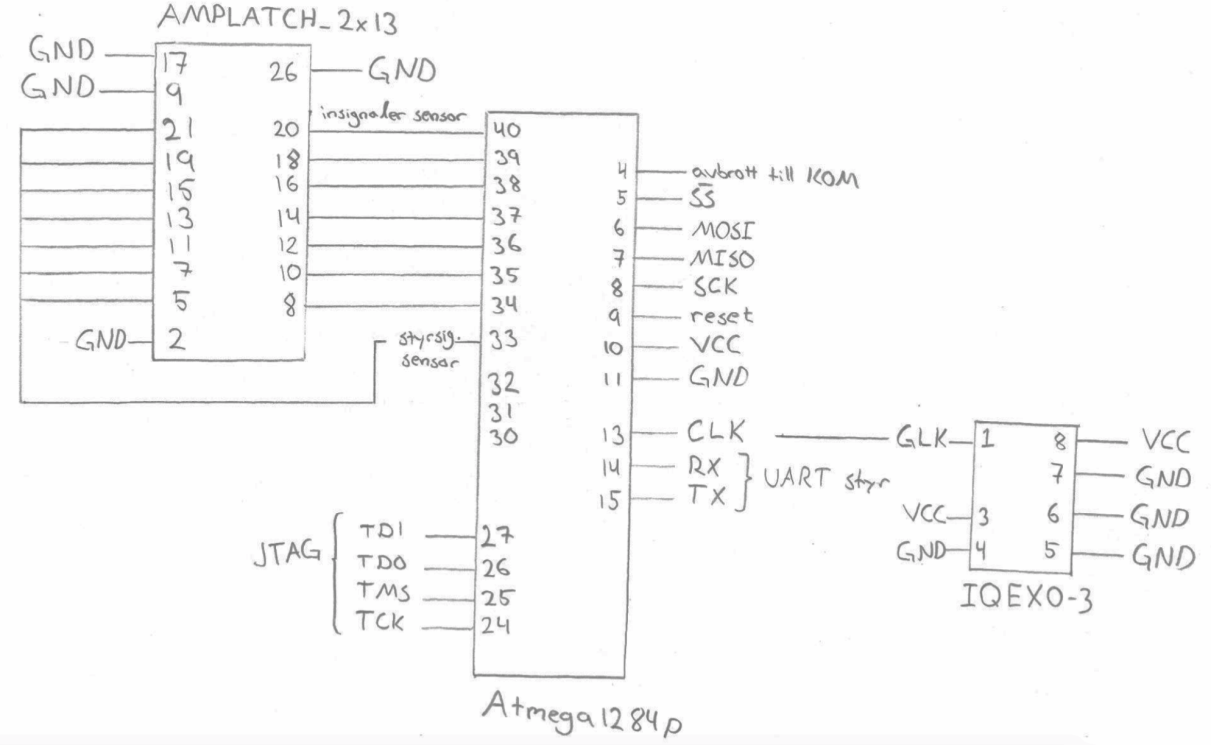
Sensormodulen består av följande komponenter:

- Atmel ATmega1284p mikrokontroller
- AMPLATCH_2x13
- IQEXO-3 klockpulsgenerator

AMPLATCH_2x13 är en reflexsensormodul bestående av 11 reflexsensorer varav 7 används. Reflexsensorerna skickar ut infrarött ljus och mäter hur mycket ljus som reflekteras från underlaget. Informationen skickas till processorn där resultatet omvandlas till digital data. Data avrundas till antingen 0 eller 1 beroende på kalibreringen av reflexsensorerna.

7.3.3 Kopplingsschema

Nedan finns ett kopplingsschema för sensormodulen, *se figur 17.*

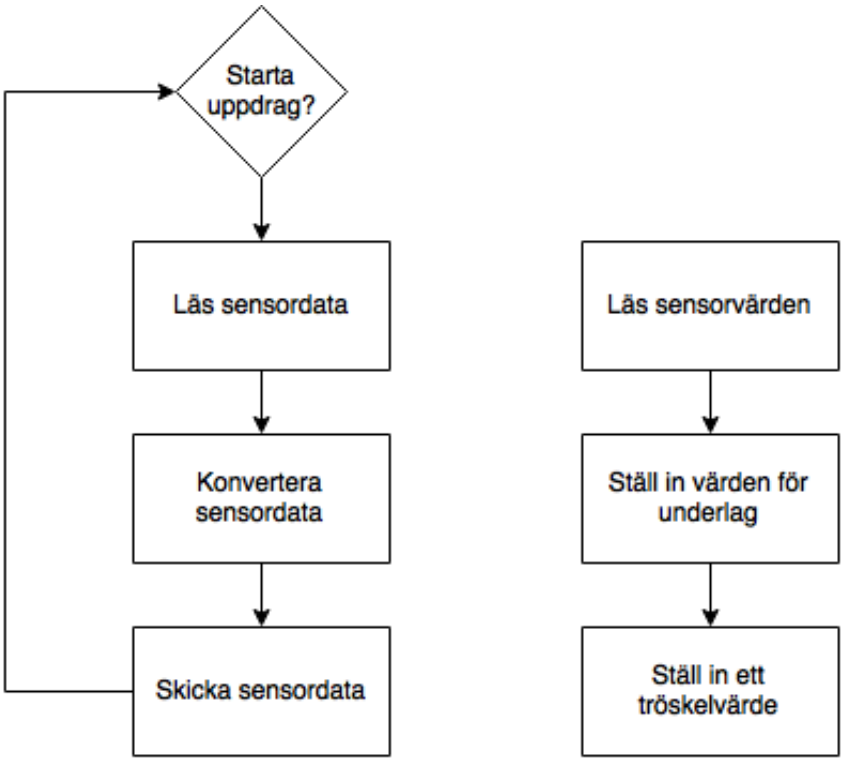


Figur 17. Kopplingsschema för sensormodulen.

- PB0-3 samt PD2-4 kopplas till reflexsensormodulens ingångar. Processorn skickar ut logiskt 1 för att aktivera en reflexsensor annars logiskt 0.
- PA0-6 kopplas till reflexsensormodulens utgångar. Processorn tar emot analoga värden från sensorerna. Dessa värden omvandlas till digitala värden inuti processorn.
- PB4-7 (SS, MOSI, MISO, SCK) kopplas till bussen.
- RXD och TXD kopplas via UART till styrmodulen.
- PC2-5 (TDI, TDO, TMS, TCK) kopplas till JTAG-kedjan.
- XTAL1 kopplas till en klockpulsgenerator.
- RESET ska anslutas för att underlätta felsökning.

7.3.4 Mjukvara

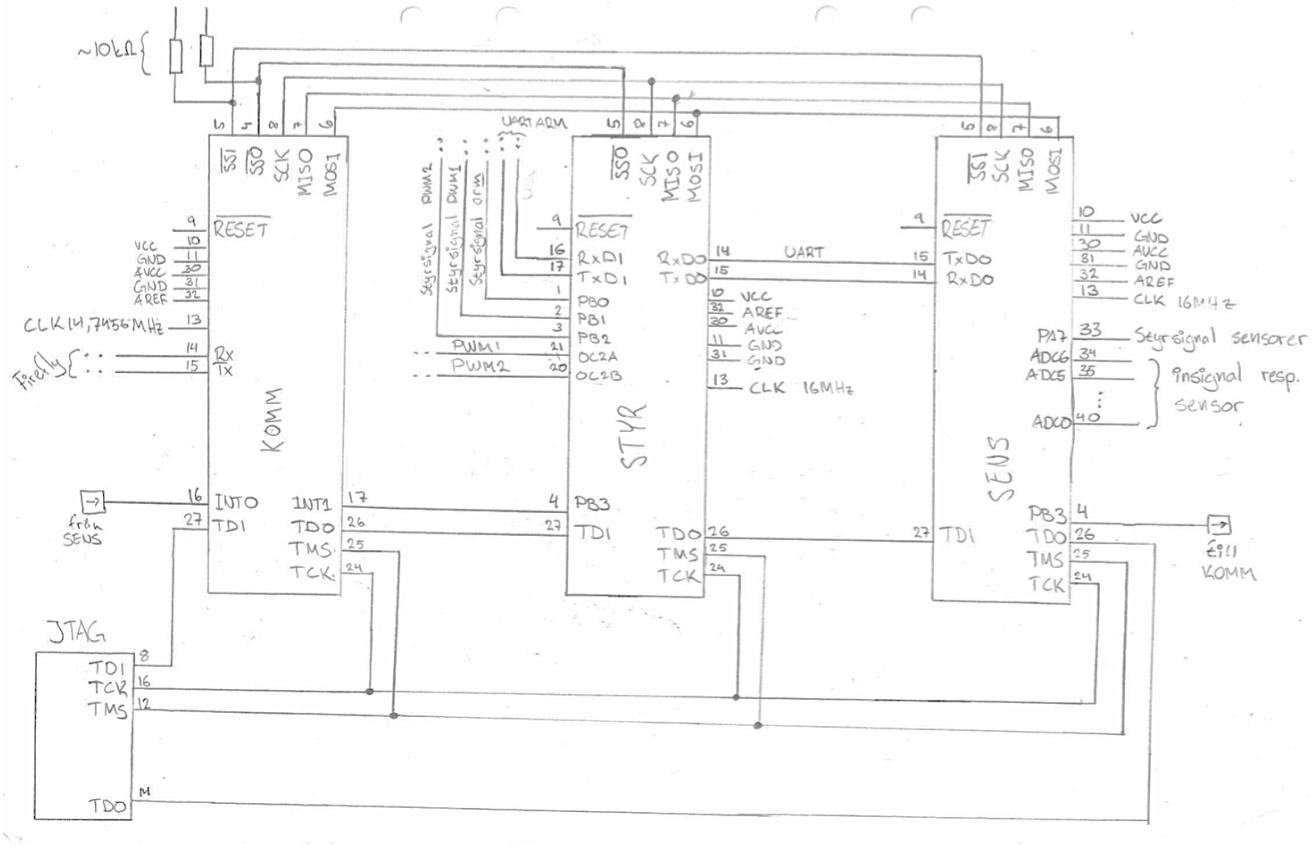
Avläsningsloopen för sensormodulen kommer under ett uppdrag läsa av sensordata, konvertera sensordata och skicka sensordata, *se figur 18.* Avbrott används vid A/D – omvandlingen i avläsningsloopen, men funktionen behöver inte utföra något under tiden det sker. Det finns en knapp i användargränssnittet för att kalibrera sensorerna, första gången knappen trycks bestäms vad som är underlag och nästa gång bestäms vad som är tejpmarkering. Ett tröskelvärde bestäms sedan för att kunna tolka sensordata och avgöra om signalen ska bli logiskt 1 eller logiskt 0 innan den skickas till kommunikationsmodulen. Beroende på styrkan av ljuset som reflekteras från varje reflexsensor räknas ett reglerfel ut som beskriver hur roboten befinner sig i förhållande till linjen. Detta reglerfel skickas sedan direkt till styrmodulen där själva regleringen sker.



Figur 18. Flödesschema för avläsning samt kalibrering.

8 Beskrivning av kommunikation mellan delsystem

Här beskrivs hur de olika modulerna i roboten kommunicerar med varandra och även hur robotenheten kommunicerar med datorenheten. En översiktlig beskrivning vad det är för information som går mellan de olika modulerna. Robotenheten har en intern SPI-buss där kommunikationsmodulen är master och en direktbuss mellan styrmodulen och sensormodulen. *Se figur 19* för översiktlig bild för systemet.



Figur 19. Översiktligt kopplingsschema för systemet.

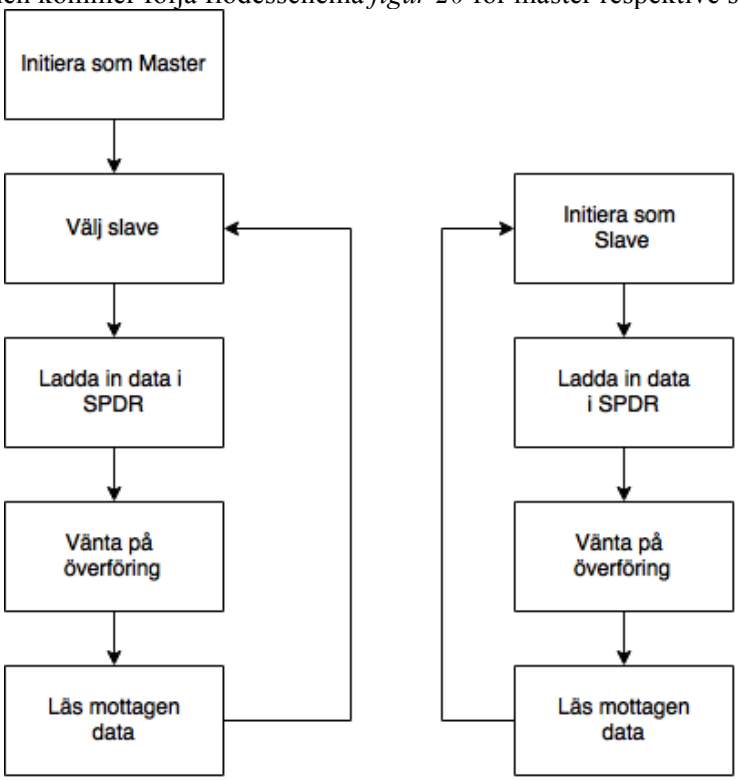
8.1 SPI-Bussen

SPI-bussen används framförallt för att distribuera kommandon som användaren skickar från datorn och för att skicka tillbaka styrbeslut och sensorvärden till datorn. Mellan datorenheten, styrmodulen, sensormodulen och kommunikationsmodulen kommer följande, *se tabell 1,* data skickas. För utförligare beskrivning av gränssnitten mellan de olika modulerna, *se appendix B*.

Tabell 1, översikt av data som överförs mellan modulerna samt datorenheten.

	Datorenhet	Styrmodul	Sensormodul
Kom.modul	**Data:** Styrbeslut, Sensorvärden, Regulatorvärden **Kommandon:** Start , Stop, Manuella styrkommandon arm/hjul, Kalibrera	**Data:** Styrbeslut, Regulatorvärden **Kommandon:** Start, Stop, Manuella styrkommandon arm/hjul	**Data:** Sensorvärden **Kommandon:** Start, Stop, Kalibrera

SPI-kommunikationen kommer följa flödesschema *figur 20* för master respektive slav.

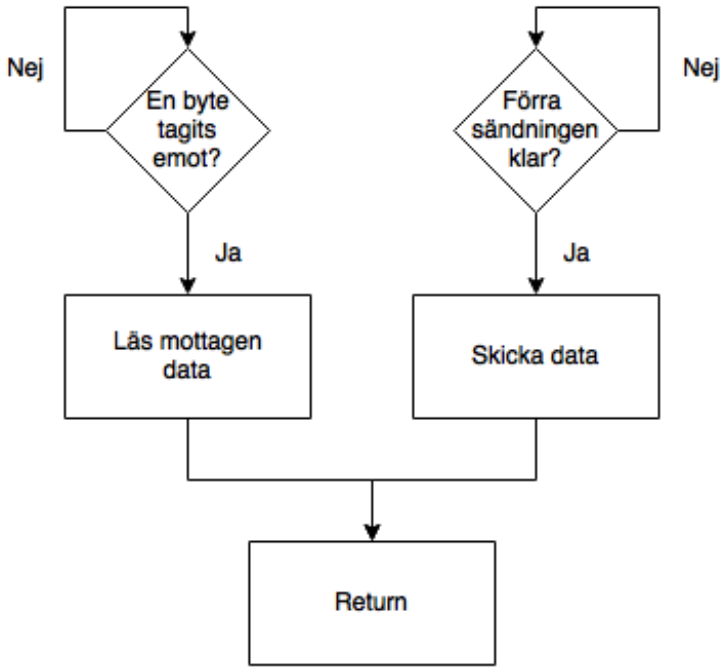


Figur 20. Flödesschema för mottagning samt sändning via SPI.

8.2 Direktbussen

Mellan sensormodulen och styrmodulen finns en direktbuss för att skicka reglerfelet som ska användas för att följa linjen, vilket kommer göras via UART. För protokoll se *appendix B*. Detta för att minska risken för eventuella fördröjningar av data från sensormodulen till styrmodulen, då styrmodulen behöver aktuell sensordata. Utan denna direktbuss hade informationen behövt gå via kommunikationsmodulen för att skickas vidare till styrmodulen.

UART-kommunikationen kommer följa flödesschema enligt *figur 20* för att skicka resp. ta emot data. För Pseudokod *se appendix A*.



Figur 21. Flödesschema för sändning och mottagning via UART.

9 Slutsatser

Som systemet är uppbyggt nu består det av en kommunikationsmodul, en sensormodul samt en styrmodul. Styrmodulen hanterar både armen och regulatorn, det gör att sensordata måste skickas från sensormodulen till styrmodulen. Detta har lösts med en extern UART mellan modulerna. En betydligt smidigare lösning hade varit att ha sensorenhet och regulatorn på samma modul, då hade den externa UART:n inte behövts.

För att kunna vidareutveckla robot till helt autonom bör inverskinematiken beräknas på styrmodulen. Detta skulle innebära att roboten med hjälp av sensorer själv kan ta beslut om vart robotarmen ska.

Det mest kritiska när armen ska röra sig mjukt är en bra algoritm för inverskinematik alternativt rörelseplanering. I nuläget bygger algoritmen på enkla iterationer av förflyttningar av respektive led tills slutpositions nås. Utvecklingsmöjligheterna för inverskinematiken är väldigt stora, då antalet tillvägagångssätt är många.

Avläsningar från Kinect-enheten är ibland icke pålitliga, då den ibland tappar positionen för en mätpunkt. Detta innebär att programmet tror att armen förflyttar sig väldigt snabbt och resultatet ger en robotarm som upplevs skakig och instabil. Flera alternativ för att lösa detta har diskuterats, var av ett är ett låg passfilter som filtrerar bort värden som avviker från föregående.

Referenser

[1] J. Hydén, ”Teknisk Dokumentation,” Linköpings Universitet, Linköping, 2015.

[2] T. Svensson och C. Kryssander, Projektmodellen LIPS, 1:1 red., Lund: Studentlitteratur AB, 2011.

Appendix

A. Programlistning

Kod för kommunikationsmodulens UART och SPI

```
//***************uart_master.cpp*****************************

// Funktion för att skicka data
void uart_sendChar(unsigned char data)
{
     // Vänta tills registret är ledigt
     while (!( UCSRA & (1<<UDRE)));

     // Ladda in data i registret
     UDR = data;
}

//Skicka helt commando
void uart_sendCommand(const char string[])
{
     if(strlen(string) == uart_commandLength)
     {
            uart_sendChar(startCommand);
            for(uint8_t i = 0; i <= strlen(string); i++)
            {
                        uart_sendChar(string[i]);
            }
     }
}

//***************SPI_master.cpp*****************************

//Funktion för att skicka och ta emot 8 bitar
unsigned char spi_WriteRead_char(unsigned char data, uint8_t slave_nr)
{
     //sens = 0, styr = 1, ingen = 2
     select_slave(slave_nr);
     //Ladda in data som ska skickas
     SPDR = data;

     //Vänta tills överföring är klar
     while(!(SPSR &(1<<SPIF)));

     select_slave(2); //koppla från
     //Returnera mottagen data
     return(SPDR);
}
```

```
//Lika dan funktion utan slave select (används i write_string)
unsigned char spi_WriteRead(unsigned char data)
{
     //Ladda in data som ska skickas
     SPDR = data;

     //Vänta tills överföring är klar
     while(!(SPSR &(1<<SPIF)));

     //Returnera mottagen data
     return(SPDR);
}

//skicka hel sträng med text (char-array)
void spi_Write_string(const char string[], const uint8_t slave_nr)
{
     uint8_t length = 5; //sizeof(string)/sizeof(string[0]);
     if ((length == sens_commandLength) || (length ==
control_commandLength))
     {
            cli();
            select_slave(slave_nr);
            spi_WriteRead(startCommand);
            for(uint8_t i = 0; i < length; i++)
            {
                        spi_WriteRead(string[i]);
            }
            select_slave(2); //koppla från
            sei();
     }
}
```

Utdrag av kod för styrmodulens kommunikation med arm

```
//****************servos.cpp*****************************

// Funktion för att beräkna sista biten som ska skickas till servona
// Inargument: id (till servot), instruction (vad som ska göras på serovt),
adress (adress i servots minne)
//                       par_length (längden på parameters), parameters
(vad som ska skickas till servot)
uint8_t checksum_send(uint8_t id, uint8_t instruction, uint8_t adress,
uint8_t par_length, uint8_t parameters[])
{
     //+ 1 för addres egentligen ska vara i parameters
     int length = par_length + 3;
     int temp = 0;
     for(int i = 0; i < par_length; i++)
     {
            temp += parameters[i];
     }
     return 255-((id+length+adress+instruction+temp)%256);
}

// Funktion för att beräkna check-summan vid mottagen data från servo
```

```
// Inargument: id (till servot), length (längden på parameters), parameters
(vad som ska skickas till servot)
uint8_t checksum_rec(uint8_t id, uint8_t length, uint8_t parameters[])
{
	int temp = 0;
	for(int i = 0; i<length-2;i++)
	{
		temp += parameters[i];
	}
	return 255-((id+length+temp)%256);
}

/*
	Skicka information till servot enl. standard
	Inargument: id, instruktion, adress (address i servots minne),
parameters (vad som ska skickas)
	par_length (längden av parameters)
*/
void send_info(uint8_t id, uint8_t instruction, uint8_t adress, uint8_t
parameters[], int par_length)
{
	// Aktivera Tx
	TXonRXoff();
	uint8_t length =  par_length + 3;

	uint8_t temp[length+4];

	//Startbitar
	temp[0] = 0xFF;
	temp[1] = 0xFF;

	// ID, längd, instruktion, adress, sen parametrar
	temp[2] = id;
	temp[3] = length;
	temp[4] = instruction;
	temp[5] = adress;

	// Lägg till alla parametrar + checksum i temp
	for(uint8_t i = 0; i < par_length + 1 ; i++)
	{
		if(i ==  par_length)
		{
			temp[i + 6] = checksum_send(id, instruction, adress,
par_length, parameters);
		}
		else
		{
			temp[i + 6] = parameters[i];
		}
	}

	uart_sendString_arm(temp, length + 4);
	TXcomplete();
}
```

```
/*
	Läser på en adress
	Inargument: id, adress och parameters (antal adresser som ska läsas)
	parameters kan max vara 2.
*/
unsigned int read_info(uint8_t id, uint8_t adress, uint8_t parameters[])
{
	clearbuffer();

	// Max 2 bytes åt gången
	uint8_t num_param = parameters[0];
	if(num_param > 2)
	{
		// Fel har inträffat
		return 0xFFFF;
	}

	// Läsning
	uint8_t instruction = 0x02;

	//Skicka till servot att vi vill läsa från adressen med num_param
antal parametrar
	send_info(id, instruction, adress, parameters, 1);

	// Aktivera Rx
	TXoffRXon();


	//Mottagen data
	uint8_t reccived_data[num_param+6];
	//Hämta in 6 bytes + antalet parametrar
	while (uart_ctr1 < num_param+6) {}

	for(int i = 0; i < num_param+6; i++)
	{
		reccived_data[i] = uart_buffer1[i];
	}

	//Kunna skicka till check_sum
	uint8_t received_param[num_param];
	//Det vi ska returnera
	unsigned int index = 0;

	if(num_param == 1)
	{
		received_param[0] = reccived_data[num_param+4];
		index = reccived_data[num_param+4];
	}
	else if(num_param == 2)
	{
		//Börja leta här, troligtvis denna som gör att checksum blir fel
		received_param[0] = reccived_data[num_param + 3];
		received_param[1] = reccived_data[num_param + 4];
```

```
            uint8_t low = reccived_data[num_param+3];
            uint8_t high = reccived_data[num_param+4];
            index = (high << 8) + low;
      }

      uint8_t rec_checksum = reccived_data[num_param+5];

      //checksum blir fel för get angle
      if(checksum_rec(id,reccived_data[3],received_param) != rec_checksum)
      {
            // Fel har intäffat
            return 0xFFFF;
      }

      // Aktivera Tx
      TXonRXoff();
      return index;
}
```

Utdrag av kod för sensormodulen

```
//****************sensor_functions.cpp*****************************


//Kalibrera sensorerna
void calibrate()
{
      start_signal();
      _delay_ms(10);
      for(int current_lamp = 1; current_lamp<8; current_lamp++)
      {
            index = current_lamp - 1;
            select_input(current_lamp);
            start_convert();
            while(!conversion_complete) { }
            conversion_complete = false;
      }
      calibrate_floor = false;
      calibrate_tape = false;
}

//Beräkna reglerfelet som ska skickas till styr
int calculate_e()
{
      int k = 0;
      int p = 0;
      for(int i=1; i<8; i++)
      {
            k += mass[i-1]*i;
      }
      for(int i=1; i<8; i++)
      {
            p += mass[i-1];
      }
      k = (k/p); //tyngdpunktsberäkning
```

```
    int e = (4-k); //reglerfelet

    e_buff[e_counter++] = e;
    if(e_counter == 3)
    {
          e_counter = 0;
    }
    int e_send = floor((e_buff[0] + e_buff[1] + e_buff[2])/3 + 0.5);

    return e_send;
}
```

Utdrag av kod från datorenheten

```
//****************MainWindow.xaml.cs*******************************

private void printCoordinates()
        {
            stopwatch.Start();
            string[] temp = new string[3] { " X: ", " Y: ", " Z: " };
            rightHandStatus.Text = "Right Hand: ";
            rightWristStatus.Text = "Right Wrist: ";
            rightElbowStatus.Text = "Right Elbow: ";
            rightShoulderStatus.Text = "Right Shoulder: ";
            rh_ref_rs_Status.Text = "Hand-ref-Shoulder: ";


            for (int i = 0; i <= 2; i++)
            {
                rightHandStatus.Text += Environment.NewLine + temp[i] +
Math.Round(rightHand[i], 3);
                rightWristStatus.Text += Environment.NewLine + temp[i] +
Math.Round(rightWrist[i], 3);
                rightElbowStatus.Text += Environment.NewLine + temp[i] +
Math.Round(rightElbow[i], 3);
                rightShoulderStatus.Text += Environment.NewLine + temp[i] +
Math.Round(rightShoulder[i], 3);
                rh_ref_rs_Status.Text += Environment.NewLine + temp[i] +
Math.Round(rightHand[i] - rightShoulder[i], 3);
            }

            /*
            if (!checkBox_Mode.IsChecked.Value) //För att mata in
koordinater in i textboxarna
            {*/

            double x_pos = rightHand[0] - rightShoulder[0];
            double y_pos = rightHand[1] - rightShoulder[1];
            double z_pos = rightHand[2] - rightShoulder[2];
            double theta;

            if (x_queue != null && x_queue.Count() == 10)
            {
                double[] x_array = x_queue.ToArray();
                double[] y_array = y_queue.ToArray();
```

```
                double[] theta_array = theta_queue.ToArray();
                av_x = x_array.AsEnumerable().Average();
                av_y = y_array.AsEnumerable().Average();
                if(!theta_lock_checkBox.IsChecked.Value)
                av_theta = theta_array.AsEnumerable().Average();

                X_BOX.Text = "" + Math.Round(av_x, 3);
                Y_BOX.Text = "" + Math.Round(av_y, 3);
                theta_BOX.Text = "" + Math.Round(av_theta, 3);

                x_queue.Dequeue();
                y_queue.Dequeue();
                theta_queue.Dequeue();

            }

            x_queue.Enqueue(Math.Round(Math.Sqrt(x_pos * x_pos + z_pos *
z_pos), 3) * 4 / 5);
            y_queue.Enqueue(Math.Round(y_pos, 3) * 4 / 5);
            if (z_pos <= 0)
            {
                theta = Math.Asin(x_pos / (Math.Sqrt(x_pos * x_pos + z_pos
* z_pos)));
            }
            else
            {
                theta = Math.PI - Math.Asin(x_pos / (Math.Sqrt(x_pos *
x_pos + z_pos * z_pos)));
            }
            theta = theta * (180.0 / Math.PI);
            if (Math.Abs(theta) > 45)
            {
                theta = Math.Sign(theta) * 45;
            }
            theta_queue.Enqueue(theta);
            stopwatch.Stop();
            // Angle4.Text = "" + System.Math.Round(rightHand[2] -
rightShoulder[2], 3);
            /*}
        */
        }
```

B Gränssnitt mellan moduler

All kommunikation med datorenheten går först via kommunikationsmodulen.

Kommando	Beskrivning	Läge	Sändare	Mottagare
W	Kör framåt	Hjul	Datorenhet	Styrmodul
A	Kör vänster	Hjul	Datorenhet	Styrmodul
S	Kör bakåt	Hjul	Datorenhet	Styrmodul
D	Kör höger	Hjul	Datorenhet	Styrmodul
Q	Rotera vänster	Hjul	Datorenhet	Styrmodul
E	Rotera höger	Hjul	Datorenhet	Styrmodul
F	Stanna	Hjul	Datorenhet	Styrmodul

T	Plattform vänster	Arm	Datorenhet	Styrmodul
Y	Plattform höger	Arm	Datorenhet	Styrmodul
U	Servo 1, upp	Arm	Datorenhet	Styrmodul
I	Servo 1, ner	Arm	Datorenhet	Styrmodul
G	Servo 2, upp	Arm	Datorenhet	Styrmodul
H	Servo 2, ner	Arm	Datorenhet	Styrmodul
J	Servo 3, upp	Arm	Datorenhet	Styrmodul
K	Servo 3, ner	Arm	Datorenhet	Styrmodul
V	Klo motsols	Arm	Datorenhet	Styrmodul
B	Klo medsols	Arm	Datorenhet	Styrmodul
N	Klo släppa	Arm	Datorenhet	Styrmodul
M	Klo greppa	Arm	Datorenhet	Styrmodul

u	Arm hastighet +	Arm	Datorenhet	Styrmodul
d	Arm hastighet -	Arm	Datorenhet	Styrmodul

5	Kalibrera golv	Hjul	Datorenhet	Sensormodul
6	Kalibrera tejp	Hjul	Datorenhet	Sensormodul
7	Starta uppdrag	Hjul	Datorenhet	Sensormodul
8	Återvända hem	Hjul	Datorenhet	Sensormodul

0	Hjul hastighet	Arm	Datorenhet	Styrmodul
1	P-värde	Arm	Datorenhet	Styrmodul
2	I-värde	Arm	Datorenhet	Styrmodul
3	D-värde	Arm	Datorenhet	Styrmodul

a	Välja armläge	Hjul	Datorenhet	Styrmodul
b	Välja hjulläge	Arm	Datorenhet	Styrmodul

P	Autonom avstängning	Hjul/Arm	Datorenhet	Styrmodul
O	Töm provrör	Arm	Datorenhet	Styrmodul
X	Start position för arm	Arm	Datorenhet	Styrmodul
p	Plocka upp funktion	Arm	Datorenhet	Styrmodul
v	Skicka armvinklar	Arm	Datorenhet	Styrmodul
t	Låsa armrotation	Arm	Datorenhet	Styrmodul

s	Mål hittat	Hjul	Sensormodul	Styrmodul
!	Startbit sträng	Arm/Hjul	Datorenhet/Sensormodul/Styrmodul	Datorenhet/Sensormodul/Styrmodul
e	Startbit sensordata	Hjul	Sensormodul	Datorenhet