

# Teknisk dokumentation

Grupp 6

21 maj 2025

Version 1.0



Status

Granskad	Ebba Lundberg	2025-05-21
Godkänd	Namn	2025-xx-xx

**Beställare:**

Mattias Krysander, Linköpings universitet  
Telefon: +46 13282198  
E-post: mattias.krysander@liu.se

**Handledare:**

Theodor Lindberg, Linköpings universitet  
E-post: theodor.lindberg@liu.se

**Projektdeltagare**

Namn	Ansvar	E-post
Linus Funquist		linfu930@student.liu.se
Ebba Lundberg	Dokumentansvarig	ebblu474@student.liu.se
Andreas Nordström	Projektledare	andno773@student.liu.se
Sigge Rystedt		sigry751@student.liu.se
Ida Sonesson	Dokumentansvarig	idaso956@student.liu.se
Lisa Ståhl	Designansvarig	lisst342@student.liu.se

**INNEHÅLL**

1	Inledning	1
2	Produkten	1
3	Bakgrund	2
3.1	Sensorenheten . . . . .	2
3.2	Styrenheten . . . . .	2
3.3	Kommunikationssystemen . . . . .	2
3.4	PC . . . . .	2
4	Systemet	3
5	Modulerna	3
5.1	Sensorenheten . . . . .	4
5.2	Styrenheten . . . . .	7
5.3	Kommunikationssystemen . . . . .	9
5.4	PC . . . . .	14
6	Slutsatser	15
7	Appendix	17
7.1	Grafer . . . . .	17
7.2	Kopplingsscheman . . . . .	18
7.3	Exempelkod . . . . .	19
7.4	Tabeller . . . . .	31
7.5	Övrigt . . . . .	33

**DOKUMENTHISTORIK**

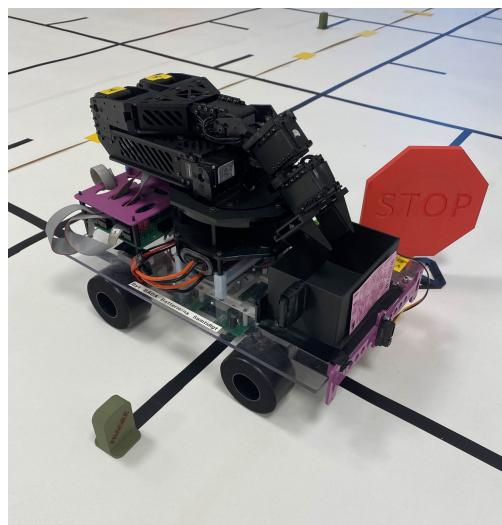
<b>Version</b>	<b>Datum</b>	<b>Utförda ändringar</b>	<b>Utförda av</b>	<b>Granskad</b>
1.0	2025-05-21	Första version	LF, EL, AN, SR, IS, LS	LF, EL, AN, SR, IS, LS

## 1 INLEDNING

Detta dokument är skrivet för att kunna användas som konstruktionsunderlag vid replikering av lagerroboten samt för att underlätta underhåll och felsökning av både hård- och mjukvara. Allt material som projektet lånat kommer att vara listat tillsammans med beskrivningar av både det översiktliga systemet och alla delsystem i detalj.

## 2 PRODUKTEN

Produkten som har tagits fram är en lagerrobot, se Figur 1, bestående av en robotplattform och en robotarm med en gripklo. Robotten kan navigera i en känd lagermiljö med hinder och specifika plockstationer där varor är placerade i förväg. Vid plockstationerna ska varorna plockas upp med hjälp av robotarmen för att sedan lämnas vid en utlämningsstation. Lagerroboten använder ett flertal sensorer för att utföra linjeföljning, korrekta svängar och kunna detektera hinder. Daten skickas via en SPI-buss till kommunikationsenheten som också skickar kommandon till styrenheten. På en separat PC kan användaren kontrollera robotten genom ett gränssnitt.



**Figur 1:** Lagerrobot med robotarm och robotplattform.

### 3 BAKGRUND

I följande kapitel ges en kort beskrivning av de komponenter som har använts i respektive delsystem.

#### 3.1 Sensorenheten

Sensorenheten är försedd med tre olika typer av sensorer, två specifika algoritmer som används i mjukvaran för särskilda beräkningar och en ATmega1284P mikrokontroller. En IR-sensor används för att detektera hinder, två reflexsensormoduler används för linjeföljning och ett gyroskop är installerat så att roboten ska kunna utföra mer precisa svängningar. Eftersom flyttalsberäkning är långsamma på en ATmega1284P används istället linjärinterpolation för att snabbare beräkna utdata från avståndssensorn. För att kunna avgöra om roboten är till höger eller vänster om linjen den följer, så genomförs en enkel tyngdpunktsberäkning för att ta fram data på avvikelsen mitten och mellan reflexsensormodulerna.

#### 3.2 Styrenheten

Styrenheten ansvarar för alla förflyttningar och rörelser som roboten kan utföra, och består av en ATmega1284P mikrodator. Det styrenheten styr är fyra hjul konfigurerade som två hjulpar och en robotarm med totalt 8 servon. Styrenheten gör inte så många beräkningar själv, utan nästan alla beräkningar görs på andra enheter. Styrenheten tar bara emot kommandon och utför dem. De kommandona kan antingen vara manuell styrning som kommer direkt från användaren via användargränssnittet, eller styrbeslut tagna under autonom körning baserat på sensorvärden och snabbaste-vägen-algoritmen.

#### 3.3 Kommunikationsenheten

Kommunikationsenheten sköter all intern och extern kommunikation. Den består till största del av en enkortsdator av typen Raspberry Pi. Den ska ta emot data från styr- och sensorenheten via virkort och överför sedan information till PC:n via bluetooth-anslutning för att kartlägga var roboten befinner sig. Kommunikationsenheten ska även kunna ta emot data via bluetooth från PC:n avsedd för styrenheten för manuell styrning. Vid autonom körning ska kommunikationsenheten, med hjälp av data från resterande enheter, kunna ta beslut om kortaste väg. Kommunikationen mellan Raspberry Pi:n och sensor- samt styrenheten kommer genomföras via en SPI-buss.

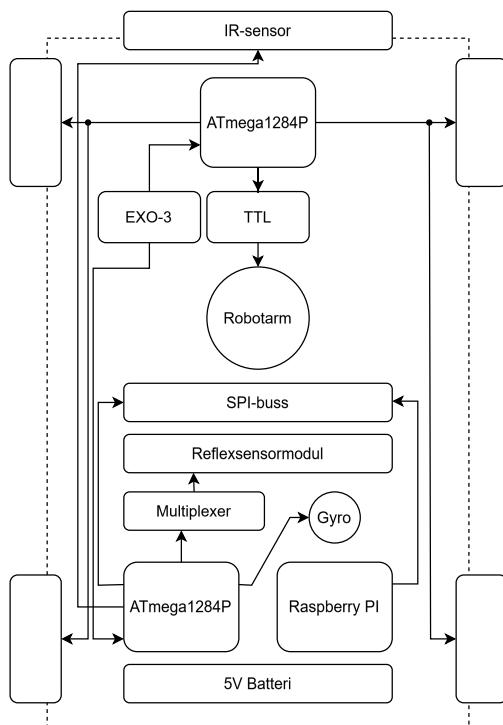
#### 3.4 PC

Denna enhet kommer endast vara en av gruppens bärbara persondator och har som huvuduppgift att kunna föra kommunikation med Raspberry Pi:n. På PC:n används ett användargränssnitt, GUI, för att föra all kommunikation. Via

denna kontrolleras manuell styrning och eventuell data från sensor- och styrenheten kommer visas. Det är även här som användaren kan välja banans storlek och placering av varor som i sin tur skickas över Bluetooth-anslutning till kommunikationsenheten för beräkning av kortaste väg.

## 4 SYSTEMET

Systemet består av en sensor-, kommunikations- och styrenhet. Dessa kommer att vara installerade på roboten som illustreras i Figur 2. Följande kapitel beskriver respektive delsystem i detalj.



**Figur 2:** Blockschema över systemet.

## 5 MODULERNA

Följande kapitel beskriver detaljerat hur alla delsystem är designade och implementerade i det större systemet.

## 5.1 Sensorenheten

Sensorenhetens uppgift är att samla in data som sedan skickas vidare till kommunikationsenheten. Sensordatana kommer att skickas kontinuerligt från samtliga sensorer vilket utgör grunden till de styrbeslut som tas samt kartläggningen av lagermiljön.

### 5.1.1 Komponenter och programpaket

I Tabell 1 listas de komponenter som ingår i sensorenheten.

**Tabell 1:** Lista över komponenter för sensorenheten.

Komponent	Egenskap	Antal
Mikrodator	ATmega1284P	1
Emulator	JTAG ICE 3	1
Oscillator	EXO-3	1
Avståndssensor	GP2Y0A21, 10-80cm	1
Reflexsensormodul		2
Multiplexer	För reflexsensormodulen	2
Gyroskop	MLX90609	1
Resistor	18 kΩ, lågpassfilter	1
Kondensator	100 nF, lågpassfilter	1

I Tabell 2 listas de programpaket som användes i sensorenheten.

**Tabell 2:** Lista över programpaket för sensorenheten.

Programpaket	Egenskap
avr/io.h	Konfigurerar I/O-portarna
avr/interrupt.h	Avbrott
math.h	Matematiska operationer
util/delay.h	Funktioner för tidsfördröjningar i koden

### 5.1.2 ATmega1284P

Funktionaliteten hos sensorenheten förutsätter att ATmega1284P mikrodatorn är korrekt ansluten till sensorerna och att dessa initieras korrekt i mjukvaran.

Figur 6 (se Appendix A.2) visar hur alla sensorer är uppkopplade till ATmega1284P. Mikrodatorn har två multiplexar inkopplade som styr vardera reflexsensormodul. Genom dessa kan ATmega1284P kontrollera vilken reflexsensor som är aktiv i modulen och läsa av dess analoga utsignal. Ytterligare en EXO-3 är inkopplad för att styra mikrodatorns klocka, då dess egna interna klocka inte är tillräckligt stabil [1]. IR-sensorn GP2Y0A21:s analoga utsignal lågpass-

filtreras innan den kopplas till ATmega1284P:s A/D-omvandlare. Sist är mikrodatorn kopplad till SPI-bussen genom dess SS, MISO och SCK-portar.

Sensorerna och delsystem inom modulen initieras som i Exempelkod 6 (se Appendix A.3). Alla sensorerna använder extern referensspänning och är inställda att vänsterjustera utdata vilket möjliggör att enklare spara endast de 8 mest signifika bitarna. Den enda skillnaden i deras ADMUX-inställningar är vilken port A/D-omvandlaren ska läsa av. ADCSRA är också identiska då förstoraren (*prescaler* på engelska) sätts till 128 och möjliggör aktivering av A/D-omvandlaren. Prescalern valdes till 128 för att mikrodatorns frekvens är 16 MHz, medan A/D-omvandlaren får bäst upplösning mellan 50-200 kHz. Med en prescaler på 128 fås frekvensen på A/D-omvandlaren till 125 kHz. Alla sensorer initieras separat trots att de har många gemensamma inställningar för att underlätta framtida implementeringar av andra sensorer som kan kräva andra inställningar. Andra viktiga delsystem att initiera är SPI-bussen och avbrott som båda måste aktiveras innan de kan användas.

### 5.1.3 *IR-sensor*

IR-sensorn GP2Y0A21 används för att detektera hinder i lagermiljön. Efter att ADMUX är inställt till rätt port görs en A/D-omvandling. För att undvika tunga beräkningar med flyttal så används linjär interpolation istället. I sensorns datablad finns en graf med vilka analoga spänningsvärdet som motsvarar vilket avstånd, se Figur 5 (Appendix A.1). Det går alltså att uppskatta ett värde  $y$  för ett givet  $x$  mellan två kända punkter  $(x_0, y_0)$  och  $(x_1, y_1)$ , genom formeln för linjär interpolation:

$$y = y_0 + (x - x_0) \cdot \frac{y_1 - y_0}{x_1 - x_0}. \quad (1)$$

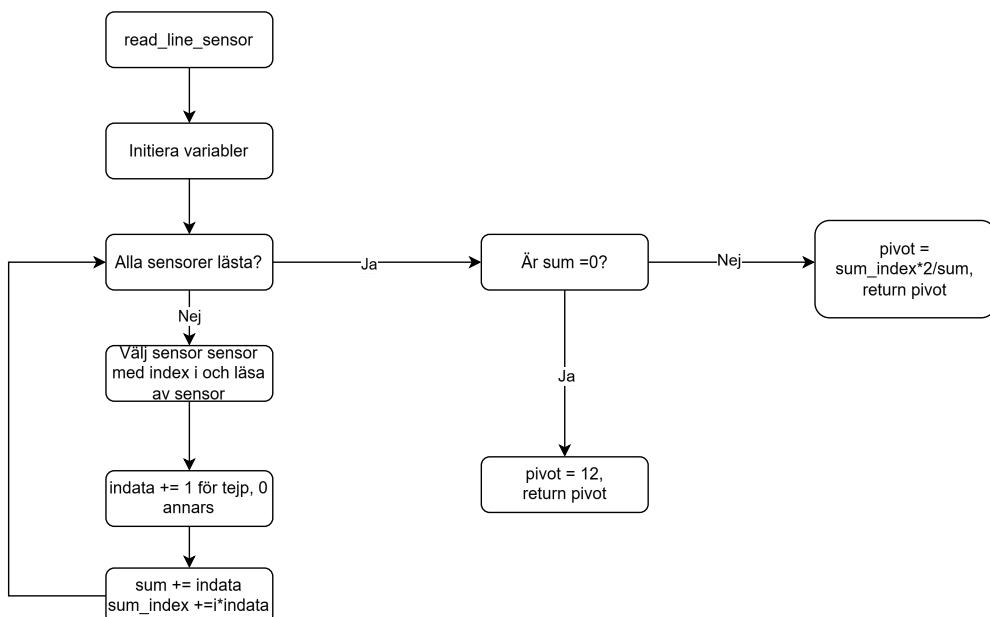
Där  $y$  är det avståndet från avståndssensorn och  $x$  är det uppmätta spänningsvärdet.  $(x_0, y_0)$  och  $(x_1, y_1)$  är kända värden från databladet. Avståndet  $y$  blir en approximation som har ett värde mellan  $y_0$  och  $y_1$ . Detta värde sparas sedan i en global variabel som kan skickas till Raspberry Pi:n genom SPI-bussen.

### 5.1.4 *Reflexsensormoduler*

Reflexsensormoduler behövs för att möjliggöra linjeföljning och detektera korsningar samt plockstationer. En enkel tyngdpunktsberäkning används för att beräkna hur långt ifrån sensorns mittpunkt som linjen är. Tyngdpunkten räknas ut genom:

$$k_T = \frac{\sum_k m_k k}{\sum_k m_k} \quad (2)$$

där  $k_T$  och  $m$  är tyngdpunkten respektive magnituden som en reflektor detekterar. I Figur 3 illustreras ett flödesschema över tyngdpunktsberäkningen för reflexsensorn.



**Figur 3:** Flöde för reflexsensor.

Flödesschemat implementerades enligt Exempelkod 1 (se Appendix A.3) där funktionen `read_line_sensor` returnerar ett pivotelement. Om summan av alla reflexsensorerna är noll, alltså att ingen sensor har sett tejp, så blir pivot 12, så att roboten ska åka rakt framåt. För att möjliggöra beräkningar med decimalprecision utan att pivot sparas som ett flyttal i mikrodatorn multipliceras den med 2. Då kan kommunikationsenheten dividera värdet den får med 2 för att få tillbaka det egentliga värdet. Två stycken reflexsensorer används för att underlätta regleringen av roboten. Mer information om regleringen finns i avsnitt 5.3.7.

### 5.1.5 Gyroskop

Gyrot har som uppgift att mäta 90 eller 180 graders svängar och kan aktiveras av kommunikationsenheten vid två tillfällen. Antingen när den främre reflexsensorn detekterar en korsning eller plockstation och när IR-sensorn detekterar ett hinder. I båda situationerna startar kommunikationenhetens timer via SPI-bussen som kommer att göra ett avbrott var 10 ms. Då utför gyrot en avläsning och uppdaterar datan som kommer att skickas till bussen. Kommunikationenhetens stänger av gyrot när gyrodatan har kommit upp i ett visst hårdkodat värde, i detta fall 60 och 120, som motsvarar 90 eller 180 grader.

## 5.2 Styrenheten

Styrenhetens uppgift är att på mikrodatorn ta emot instruktioner och data från kommunikationsenheten via SPI-bussen, och utifrån det kontrollera robotens olika motorer och servon. Det som styrs av styrenheten är två hjulpar och en robotarm bestående av 8 servon.

### 5.2.1 Komponenter och programpaket

I Tabell 3 listas de komponenter som ingår i styrenheten.

**Tabell 3:** Lista över komponenter för styrenheten.

Komponent	Egenskap	Antal
Mikrodator	ATmega1284P	1
Robotarm	PhantomX Reactor	1
Hjul		4
Oscillator	EXO3	1
Resistor	10 kΩ	1

I Tabell 4 listas de programpaket som användes i styrenheten.

**Tabell 4:** Lista över programpaket för sensorenheten.

Programpaket	Egenskap
avr/io.h	Konfiguerar I/O-portarna
avr/interrupt.h	Avbrott
math.h	Matematiska operationer
util/delay.h	Funktioner för tidsfördröjningar i koden

### 5.2.2 Styrning av robotplattformen

I styrenhetens mikrodator finns kod implementerad för styrning av robotplattformen, både manuellt och autonomt. Hur de olika styrmoderna fungerar på just styrenhetens mikrodator är enligt samma princip, skillnaden är till stor del bara vilka instruktioner som skickas från kommunikationenheten, och om det är en människa eller en regleralgoritm som tar beslutet.

På robotplattformen styrs dess respektive hjulpar separat. För att styra hjulparen används en kombination av PWM och DIR-signaler, där PWM-signalerna styr hjulparets fart och DIR styr hjulens riktning. PWM-signaler är periodiska signaler som varierar mellan högt och lågt, och ju större andel av tiden som signalen är hög desto snabbare rullar hjulen. DIR-signalen är binär, så 0 respektive 1 ger olika riktningar.

All kommunikation till styrenheten sker via SPI. När data kommer på bussen triggas ett avbrott, och där behandlas den data som skickats. Ibland skickas bara ett kommando som representerar en handling, exempel på det är fram, bak,

rotera höger och liknande. För mer avancerade kommandon, som det som används vid autonom linjeföljning, kan även extra data skickas med som ska användas som argument till funktionen. För detta används en state machine. Genom att byta till ett specifikt state när ett bestämt kommando skickas kan mikrodatorn sen vänta in så mycket data den behöver, till exempel tar mikrodatorn emot tre bytes som tillsammans representerar utsignalen från regleralgoritmen när linjeföljning är aktivt.

I styrenhetens mikrodators mainloop finns det en lång rad med if-satser, och beroende på vad senaste instruktionen från kommunikationsenheten var körs olika funktioner, som styr roboten på olika sätt. Både autonom styrning och manuell styrning fungerar på detta sätt, och skillnaderna beskrivs nedan.

#### **MANUELL STYRNING AV ROBOTPLATTFORMEN**

Under manuell styrning kommer alla instruktioner via kommunikationsenheten från användargränssnittet. Beroende på användarens knapptryck skickas olika kommandon från PC till kommunikationsenheten och sen vidare till styrenheten. Mer om vilket kommando som gör vad beskrivs i kapitlet om kommunikationsenheten.

#### **AUTONOM STYRNING AV ROBOTPLATTFORMEN**

Under autonom styrning kommer alla instruktioner från kommunikationsenheten att bero på sensorvärdet och styrbeslutet från kortaste-vägen-algoritmen. På robotplattformen går, som tidigare nämnt, att styra hjulpare separat. Genom att köra de två hjulparena olika snabbt går det då att få roboten att svänga, detta kallas differentialstyrning. För linjeföljningen sätts först en hastighet, och sedan ökas och minskas höger respektive vänster hjulparens hastighet med regleralgoritmens utsignal, se Exempelkod 4. Detta ger att plattformen svänger olika mycket beroende på hur stort reglerfelet är, vilket gör att roboten kan följa linjer. Utöver detta kan robotplattformen även rotera, 90 grader i korsningar och 180 grader om hinder stöts på. Dessa rotationer använder samma funktioner som rotationer i manuellt läge, men hur länge de körs styrs med hjälp av gyrodata från sensorenheten.

#### **5.2.3 Styrning av robotarmen**

Robotarmen består av åtta stycken servon, sex dynamixel AX-12A och två dynamixel AX-18A, men båda servotyperna styrs på samma sätt. Kommunikationen sker seriellt via UART, och följer dynamixels kommunikationsprotokoll [2]. Utifrån det protokollet har funktioner skrivits, bland annat `move_servo` och `get_angle`, för exempel på hur styrenheten styr servon se Exempelkod 3. Likt styrning av robotplattformen används här samma principer i manuell och autonom, men skillnader redogörs för nedan. När mikrodatorn startar sätts hastigheten på alla servon till ett värde som är längsammare än servonas originalvärde för att rörelserna inte ska bli lika ryckiga.

#### **MANUELL STYRNING AV ROBOTARMEN**

Vid manuell styrning av robotarmen finns det fyra kommandon som används: byt till högre led, byt till lägre led, rotera led motsols och rotera led med sols. Båda byt-led funktionerna byter aktiv led. Det finns sex numrerade ledar, där den första leden roterar basplattan, sjätte leden öppnar och stänger gripklon, och resterande styr armens fysiska ledar. Armens servon har begränsningar i vilka vinklar de kan nå så att de inte jobbar mot armens fysiska konstruktion, och i ledar med två servon används servonas inbyggda loadangle och action kommandon så att servona rörs exakt samtidigt. Det gör att servona laddas med önskade målvinklar, men väntar på ett till kommando, action, innan de faktiskt börjar

rotera. Det finns däremot inga begränsningar satta så att armen inte slår emot robotplattformen eller andra objekt under manuell styrning, det får användaren vara försiktig med själv.

Funktionen som får ett servo att rotera ändrar vinkeln kontinuerligt medan det kommandot skickas, som då skickas medan en knapp i användargränssnittet är nedtryckt. Till att få detta att ske kontinuerligt används en simpel räknare som räknar upp medan den aktiva instruktionen är rotera motsols eller medsols. När räknaren nått upp till ett specifikt värde nollställs den, och servot flyttas en kortare sträcka. Till detta används `get_angle` för att ta reda på den nuvarande positionen, sedan adderas eller subtraheras en konstant och den nya vinkeln ställs in på servot.

### **AUTONOM STYRNING AV ROBOTARMEN**

Alla instruktioner som skickas till robotarmen under autonom styrning är förutbestämda i kommunikationsenheten. Kommandon som kan skickas till robotarmen under autonom styrning är enbart `moveservo`. Styrenheten får via SPI vilken led som ska styras och vilken vinkel den ska ställas in i. Hur dessa vinklar bestämdes beskrivs i [5.3.5](#).

## **5.3 Kommunikationsenheten**

Kommunikationsenheten består av en Raspberry Pi. Dess uppgift är att sköta all kommunikation och de flesta beräkningarna. Kommunikationsenheten tar emot data från sensorenheten, bearbetar den och skickar informationen vidare både till PC:n för visning i användargränssnittet och till styrenheten för vidare styrning. Kommunikationsenheten tar även emot data från styrenheten som sedan skickas till PC:n för visning i användargränssnittet. På Raspberry Pi:n finns kod för SPI- och Bluetooth-kommunikation, reglering, kortaste vägen algoritm samt kod för körning i både manuellt och autonomt läge. På Raspberry Pi:n finns en separat fil med olika funktioner för manuell och autonom styrning samt en funktion som hanterar data.

### **5.3.1 Komponenter och programpaket**

I Tabell 5 listas de komponenter som ingår i kommunikationsenheten.

**Tabell 5:** Lista över komponenter för kommunikationsenheten.

<b>Komponent</b>	<b>Egenskap</b>	<b>Antal</b>
Mikrodator	Raspberry Pi	1
Nivåskiftare		1

I Tabell 6 listas de programpaket som ingår i kommunikationsenheten.

**Tabell 6:** Lista över programpaket för kommunikationsenheten.

<b>Programpaket</b>	<b>Egenskap</b>
spidev	SPI-kommunikation
time	Funktioner för tidsfördröjningar och tidsmätningar i koden
socket	Bluetoothkommunikation

### 5.3.2 SPI-kommunikation

Via SPI-bussen sker det kommunikation mellan kommunikationsenheten och sensorenheten samt mellan kommunikationsenheten och styrenheten. Mellan de olika enheterna skickas data i form av bytes som representerar olika information eller kommandon. Se kopplingsschema i Figur 7 (Appendix A.2).

#### KOMMUNIKATION MELLAN RASPBERRY PI OCH SENSORENHETEN

Se Tabell 8 (Appendix A.4) för beskrivning av den data som skickas mellan Raspberry Pi:n och sensorenheten.

I en SPI-transaktion skickar Raspberry Pi:n vilken sensor som ska skicka data och sensorenheten skickar datan från den valda sensorn. Undantaget är gyrot som först kan skicka data när Raspberry Pi:n skickar att sensorenheten ska starta gyro-timern. Det finns även ett värde som anger att gyro-timern ska stoppas. För att få reda på roadmark (korsning) status skickas ett specifikt värde och sedan får antingen 0, 1, 2 eller 3 tillbaks beroende på om roadmark är ”en rak väg”, en plockstation åt höger, en plockstation åt vänster eller en korsning.

#### KOMMUNIKATION MELLAN RASPBERRY PI OCH STYRENHETEN

Se Tabell 9 (Appendix A.4) för beskrivning av den data som skickas mellan Raspberry Pi:n och styrenheten.

Från Raspberry Pi:n skickas data till styrenheten för att styra robotplattformen samt robotarmen. Värdena 0-6 samt 31 och 32 skickas från PC:n till Raspberry Pi:n och sedan direkt vidare till styrenheten för att där leda till olika styrkommandon, detta går att se i Tabell 10 och Tabell 9. Värdet 30 indikerar att kommande tre data beskriver reglerinfo; först en ”status” som berättar om reglerfelet är positivt eller negativt, sedan ”high” och ”low” som är reglerfelet uppdelat i två delar. Värdena 40 och 41 skickar gaspådrag från vänster respektive höger sida tillbaks till Raspberry Pi:n. Värdet 50 berättar att de följande tre värdena beskriver armens aktuella läge för en viss led; först vilken led, sedan vinkeln även här uppdelad i en ”high” och en ”low”. Värdet 60 fungerar på samma sätt, där de tre följande värdena ser ut på samma sätt, men nu istället används för att ta reda på vilket läge den specifika leden ska flyttas till.

### 5.3.3 Bluetooth-kommunikation

Bluetooth-kommunikationen gäller mellan PC:n och Raspberry Pi:n. Den främsta anledningen till denna anslutning är samtliga dataöverföringar mellan dessa enheter samt för att genomföra manuell styrning av robotplattformen och robotarmen med gripklo.

#### KOMMUNIKATION MELLAN RASPBERRY PI OCH PC

Bluetooth-anslutningen görs via en socket, se Exempelkod 7 och 8 (Appendix A.3). All dataöverföring börjar med en begäran från PC:n i form av ett tal som skickas i bytes. Vidare svarar Raspberry Pi:n med korrekt data beroende på vilken data den tog emot från PC:n. Data som PC:n vill få tillbaka begärs i en loop som genomförs varje tiohundrasedssekund, se Exempelkod 9 och 10 (Appendix A.3). Data som ska skickas vidare till styrenheten för manuell styrning skickas vid specifika knapptryck. Oavsett om det är data från sensorenheten, styrenheten eller annan data som PC:n begär kommer koden i grund vara enligt exemplen.

Se Tabell 10 (Appendix A.4) för att se vad värdena 1-6 och 31-67 innebär. Värdet 20 betyder att nästa data är aktuell led för armen. Värdet 70 innebär att de tre kommande värdena som skickas kommer att vara lagerbredd, lagerhöjd sedan antal noder det finns varor mellan. Värdena 80, 81 och 82 innebär alla att antingen 0 eller 1 skickas tillbaka till PC:n beroende på status på om det finns nytt styrbeslut, om en vara är upplockad samt om autonom körning är klar. Alla dessa ”statusar” representeras av flaggor i Raspberry Pi-koden. Värdet 99 betyder att körläge ska växlas mellan autonomt och manuellt.

#### 5.3.4 *Main-loop*

I Raspberry Pi:n finns en main-funktion som först initierar bluetooth- och spi-kommunikation, se Exempelkod 14. Här skapas även tråden som Bluetooth körs på. Loopen som hela tiden körs kontrollerar om det är autonomt eller manuellt körläge.

#### 5.3.5 *Autonom körning*

Då autonom körning är aktivt startar en loop i Raspberry Pi-koden som heter autonomous-loop där styrbeslut hanteras samt eventuella hinder. Styrbesluten ligger i en lista som skapas med hjälp av data från PC:n och kortaste väg-algoritmen. Denna lista avgör i vilken ordning beslut ska göras och med hjälp av linjesensorerna sker dessa olika beslut. Hinder avgör med hjälp av avståndssensorn och då den ger ett specifikt värde kommer loopen hamna i en if-sats för nya beslut om kortaste väg och styrbeslut, se Exempelkod 15. I exempelkoden visas beslutet ”Plocka”, men det finns även beslut enligt följande: ”Höger”, ”Vänster”, ”Vänd” och ”Lämna”.

Under autonom körning så ska robotarmens dels plocka upp vara vid styrbeslutet ”Plocka” och även lämna av korgen vid styrbeslutet ”Lämna”, dessa rörelser är hårdkodade och togs fram med hjälp av avläsning av aktuell vinkel för alla ledar för robotarmen. Robotarmen ställdes i ett önskat läge, sedan skickades alla värden från styrenheten till kommunikationsenheten där de sparades i en fil.

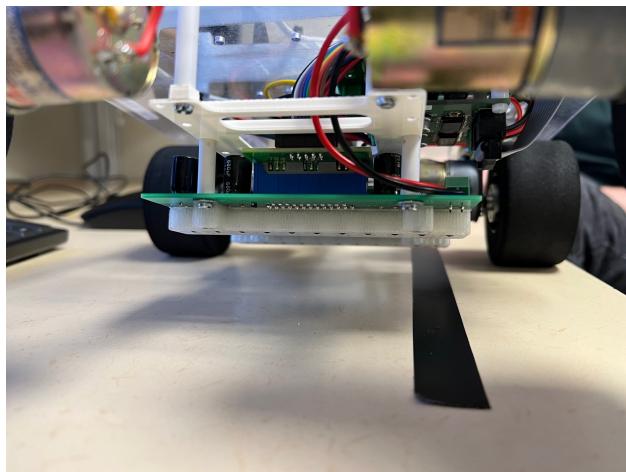
#### 5.3.6 *Manuell styrning*

Manuella styrningen hanteras till största del i en funktion som heter bluetooth-control-loop. Där hanteras samtliga data som skickas från PC:n för olika hanteringar. Beroende på mottagen data kommer man hamna i olika if-satser för olika hanteringar. Detta kan vara att PC:n önskar data från någon sensor- eller styrenhet, eller för styrning av robotplattformen eller armen. Se Exempelkod 16.

#### 5.3.7 *Reglering*

Regleringen för linjeföljningen sker genom en modifierad P-reglering. Algoritmen använder sig av fyra parametrar: front\_error, back\_error, Kp och Kd. Kp och Kd är konstanter som väljs genom testning. Variablerna front\_error och back\_error representerar avvikelsen från mitten för den främre respektive bakre reflexsen-

sorn. Se Exempelkod 5(Appendix A.3). Ett scenario som kan uppstå om man inte har med parametern  $K_p$  visas i Figur 4.



**Figur 4:** Illustration av parallellt fel.

Om regleringen endast består av:

$$\text{derivative} = K_D \cdot (\text{front\_error} - \text{back\_error}) \quad (3)$$

så kommer felet att bli noll trots att robotten inte är i mitten av tejpen och den kommer ej att regleras. Detta lösas genom:

$$\text{proportional} = K_p \cdot (\text{back\_error}) \quad (4)$$

som gör att robotten rör sig in mot mitten igen.

### 5.3.8 **Kortaste-Vägen-Algoritm**

För att robotten ska navigera i lagret krävs algoritmer som beräknar den kortaste vägen mellan olika mål samt översätter användarens input till styrbeslut som robotten kan tolka. Kortaste-Vägen-Koden är uppdelad i två filer, en som endast beräknar vägen till alla varor och en som tar hänsyn till hinder också.

Vid start används den första filen som beräknar den kortaste vägen från startpunkten till samtliga målnoder och därefter tillbaka till start. När robotten stöter på ett hinder aktiveras den andra filen. Denna tar bort den aktuella vägen mellan de två noder som hindret ligger mellan. Startpunkten uppdateras till den senast besökta korsningen, och en ny väg

beräknas från denna punkt till alla mål och avslutas i nod 1 (alltid placerad längst upp i vänstra hörnet).

För att beräkna snabbaste vägen mellan alla målnoder använder sig koden av en kombination av Bredden-Först-Sökning (BFS) och Held-Karp-Algoritmen. Vid första körningen av programmet matas följande in via användargräns-snittet:

- Lagerbredd och lagerhöjd,
- Ett dictionary där varje nyckel är ett heltalet från 1 till antalet målnoder,
- Värdet bakom varje nyckel är en lista som innehåller:
  - En enda nod, om det gäller startnoden,
  - Två noder, om det gäller en vara som befinner sig mellan två punkter.

När roboten stöter på ett hinder skickas ytterligare information till systemet:

- En lista med de två noder som hindret ligger mellan,
- Denna lista läggs till i en samlad lista över alla upptäckta hinder.

Vid detta tillfälle uppdateras startnoden till den senast besökta korsningen, och en ny vägberäkning initieras med hänsyn till de aktuella hindren.

Inledningsvis i båda fallen konstrueras en graf som representerar lagrets struktur se Exempelkod 11. Grafen kan betraktas som ett träd där varje nod motsvarar en korsning eller ett hörn i lagret. Denna graf implementeras som ett dictionary, där varje nod fungerar som en nyckel och dess värde är en lista över angränsande noder (grannar).

När roboten stöter på ett hinder aktiveras funktionen `remove_path`, vilken modifierar grafen genom att ta bort grann-relationen mellan de två noder som hindret ligger mellan. Detta innebär att den aktuella vägen inte längre betraktas som möjlig, vilket tvingar algoritmen att söka alternativa rutter.

När grafen skapats och eventuellt modifierats på grund av hinder, skapas en avståndsmatris, se Exempelkod 12. Den-na matris använder Bredden-Först-Sökning, se Exempelkod 13 för att beräkna avståndet mellan samtliga målpär i lagret. Den resulterande avståndsmatrissen skickas därefter till Held-Karp-Algoritmen, som används för att lösa det symmetriska Traveling Salesman Problem (TSP). Algoritmen beräknar den kortaste möjliga vägen som:

- Besöker varje målnod exakt en gång,
- Startar i en angiven startnod och slutar i en angiven slutnod (enligt filen som hanterar hinder),
- Alternativt både startar och slutar i nod 1 (enligt filen utan hinder).

Held-Karp-Algoritmen fungerar genom att iterativt undersöka alla möjliga kombinationer av besökta noder och suc-cessivt lägga till en obesökt nod i taget. Resultatet är:

- En lista som anger i vilken ordning målnoderna ska besökas,
- Den totala kostnaden för den optimala vägen.

Slutligen körs Bredden-Först-Sökning på nytt för att översätta denna besöksordning till en konkret sekvens av noder som roboten ska följa för att nå målnoderna i rätt ordning. När målnod nås anges detta i listan som 'goal'. Här har även specialfall och optimeringar tagits hänsyn till. Om roboten hittar två vägar som är lika långa men där en kräver en vändning och en inte gör det kommer den alltid välja vägen den inte behöver vända då det går fortare, se längre ner i Exempelkod 13.

För att översätta den genererade besökslistan med noder till styrbeslut som roboten kan tolka, används funktionen `path_to_instructions` som börjar med att översätta alla noder till koordinater, där till exempel 1 anges som (0,0), 2 som (0,1) och så vidare. Koordinaterna används sedan för att beräkna från vilket håll roboten kommer och vilket håll den ska åka för att komma till nästa nod. Dessa riktningar översätts till styrbesluten "höger", "vänster", "rakt", "vänd". När besökslistan anger 'goal' istället för en nod anges styrbeslutet `plocka`"i alla fall förutom det sista då lämnaläggs till allra sist. Styrbesluten avgör alltså vad roboten ska utföra i varje korsning alternativt plockstation längs vägen.

#### 5.4 PC

All kod på PC:n är kopplat till användargränssnittet (GUI:n). GUI:n är uppdelad i olika fönster som täcker separata krav, dessa är rutnät (bana) med knappar för att ändra storlek och antalet varor, datavisning, styrbeslut och knappar för manuell styrning. Utöver detta finns det även knappar för att ändra läge mellan autonomt och manuellt. Det finns även en knapp för att kalibrera linjesensor och en knapp i autonomt läge för att starta körning. Se Figur 8, 9 och 10 (Appendix A.4) för bilder på GUI:n i de olika lägena. Som visat på bilderna har olika designval lagts till såsom färgändring och textändring vid knapptryck. Datainhämtningen från de olika modulerna och styrbesluten startar då användaren trycker på knappen "Start". Se kapitel 5.3.3 för mer detaljerad beskrivning av hur kod för datainhämtningen ser ut. På samma sätt avbryts denna inhämtning då användaren trycker på samma knapp igen som nu istället säger "Avbryt". I Tabell 7 listas de programpaket som ingår i PC:n.

**Tabell 7:** Lista över programpaket för PC.

Programpaket	Egenskap	Antal
tkinter	Funktioner för GUI	1
time	Funktioner för tidsfördröjningar och tidsmätningar i koden	1
socket	Blåtandskommunikation	1
itertools	Funktioner för iteration	1
collections deque	Dubbelsidig lista	1

## 6 SLUTSATSER

### HUVUDSAKLIGA BIDRAG

Ett av de främsta bidragen för en fungerande lagerrobot är regleringen. En bra reglering avgör om resten av funktionerna kommer funka. Exempelvis måste roboten komma fram till en plockstation rakt för att den hårdkodade armen ska hamna rätt och ha möjlighet att plocka upp varan. Samma sak gäller vid svängar, med dålig reglering finns risken för att roboten inte hinner identifiera korsningarna.

Något som bidrog till bättre reglering och som implementerades ett tag in i projektet var att använda två reflexsensorer. I början användes endast en reflexsensor vilket gav en otillförlitlig reglering. Två reflexsensorer gjorde regleringen mer stabil.

Ett bidrag som blev en spontan lösning för sämre rotationer var eltejp på hjulen. På grund av friktion och annorlunda yta på de olika hjulen skapade detta en ostabil svängning. Eltejpen skapade därmed en jämnare yta och bättre och jämnt fördelad friktion.

### FÖRBÄTTRINGAR

Om mer tid hade funnits, hade gruppen försökt implementera mer generella lösningar. Det är en hel del hårdkodning i mjukvaran som inte riktigt är tillämpbar i industrin, men tillräckligt för projektet. Det uppstår en bugg ibland i reglering som är svår att återskapa och fixa. Gruppen tror att det hade gått att lösa genom att ha tre stycken linjesensorer.

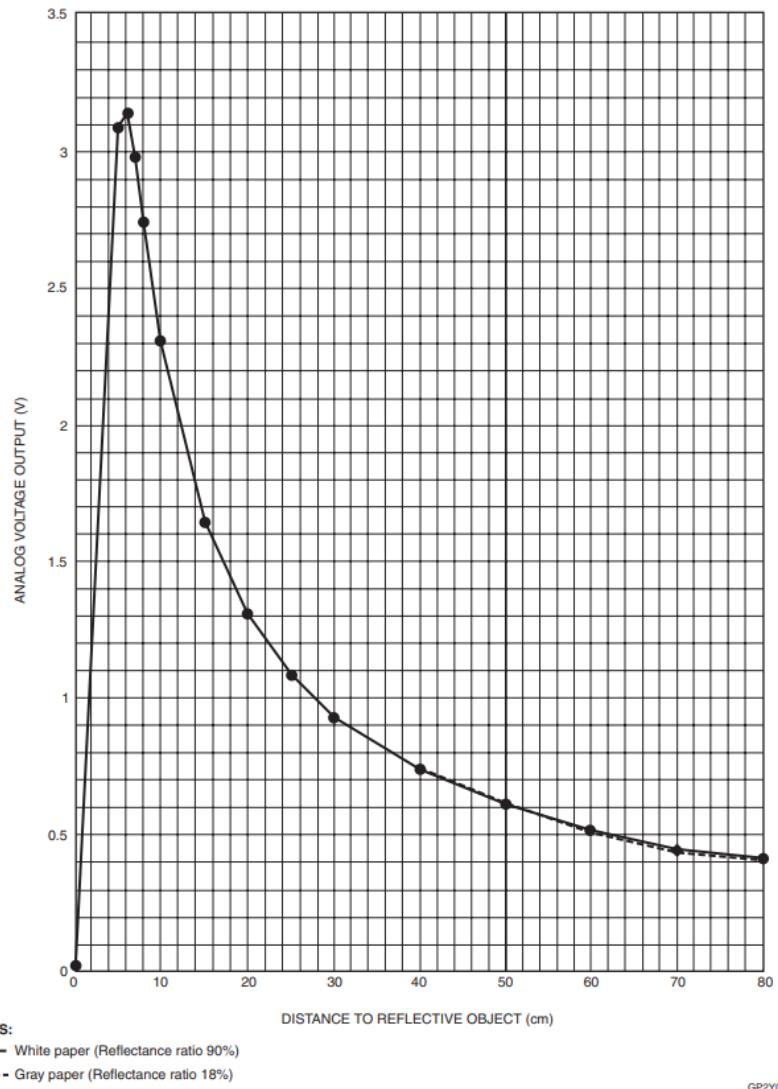
Det finns många värden som ska representera ett specifikt kommando eller som har en specifik betydelse, dessa värden har aldrig riktigt skrivits ner på ett samlat ställe under projektets gång vilket ledde till att dessa tog lång tid att sammansätta vid skrivandet av tekniska dokumentationen. Om projektet skulle göras om skulle detta kunna vara något att tänka på, så att alla värden finns samlade på samma ställe vilket också leder till en smidigare arbetsgång med mindre letande i koden.

## REFERENSER

- [1] ISY. *Reflektion*. Hämtad: 2025-04-15. [Online]. Tillgänglig: [https://da-proj.gitlab-pages.liu.se/vanheden/page/avr\\_raspberry/](https://da-proj.gitlab-pages.liu.se/vanheden/page/avr_raspberry/).
- [2] ROBOTIS. *AX-12A e-Manual*. Accessed: 2025-02-23. n.d. URL: <https://emanual.robotis.com/docs/en/dxl/ax/ax-12a/>.

## 7 APPENDIX

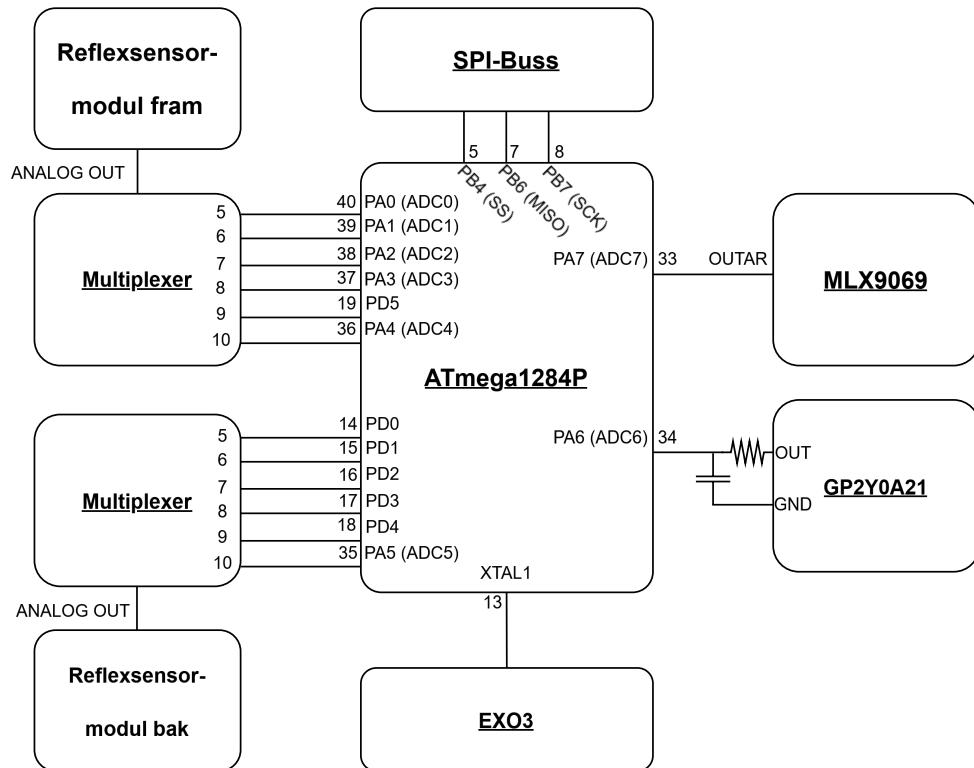
### 7.1 Grafer



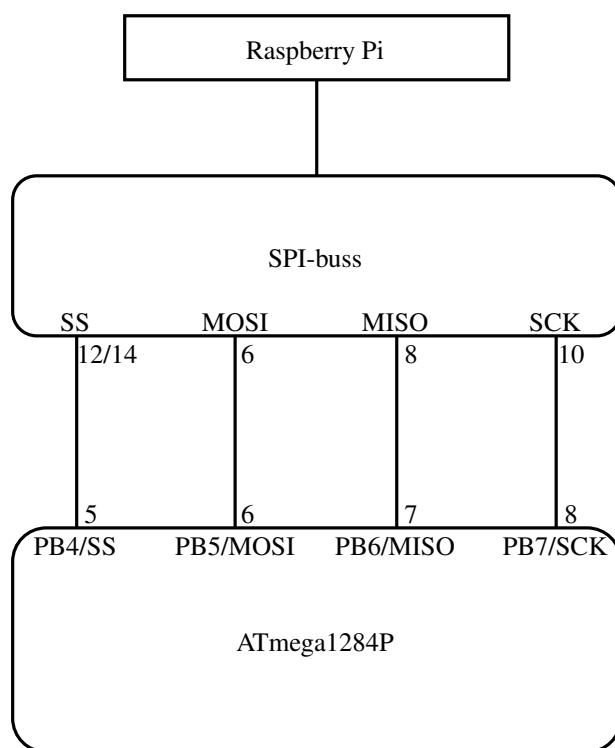
**Figur 5:** Spänningsvärden för avstånd.

## 7.2 Kopplingsscheman

Figur 6 visar kopplingsschemat för sensorenheten.



**Figur 6:** Kopplingsschema för sensorenheten.



**Figur 7:** Kopplingschema för SPI-bussen.

### 7.3 Exempelkod

**Listing 1:** Exempelkod för reflexsensormodulen.

```

1 int8_t read_line_back(int reflex_high)
2 {
3     int i;
4     volatile int8_t data;
5     volatile uint8_t indata = 0;
6     volatile int sum = 0;
7     volatile int sum_index = 0;
8     volatile int pivot = 0;
9
10    for(i = 1; i < 12; i++)
11    {
12        PORTD &= 0xF0;                                // Resets the 4 LSB bits in PORT A.
13        PORTD |= i;                                 // Set multiplexer to index i.
14        PORTD |= 0x10;                             // Start sensor.
15
16
17        indata = is_active_reflex(reflex_high);      // 1 if current sensor sees tape.
18        PORTD &= 0xEF;                            // Turn off sensor.
19
  
```

```

20     sum += indata;
21     sum_index += (i)*indata;
22 }
23
24 if (sum == 0)                                // Sees no tape.
25 {
26   pivot = 12;
27 } else {
28   pivot = (sum_index*2)/sum;                  // Center of mass calculation
29 }
30
31 return data = (int8_t)(pivot);              // Center of mass calculation
32 }
```

**Listing 2:** Exempelkod för styrenhetens mainloop.

```

1
2
3 int main(void)
4 {
5
6   DDRA = 0b1010;
7   PORTA = 0b0010;
8   DDRD = 0b11000110; // Sät TXD och TX enable till output och RXD till input
9   USART_Init(MYUBRR);
10  SPI_init();
11  init_pwm();
12
13 _delay_ms(200);
14 for(unsigned int i=1; i <= 7; i++)
15 {
16   set_speed(i, 80); // Sänk alla servons hastighet frutom gripklon
17 }
18
19 set_speed(8, 500); // Sätt gripklons hastighet snabbare än resterande servon
20
21 while (1)
22 {
23
24   if(current_action == 0x1)
25   {
26     drive_fwd();
27   }
28   else if(current_action == 0x2)
29   {
30     rotate_left();
31   }
32   else if(current_action == 0x3)
33   {
34     reverse();
35   }
36   else if(current_action == 0x4 )
37   {
38     rotate_right();
39   }
40   else if(current_action == FWD_LEFT)
41   {
```

```

42         fwd_left();
43     }
44     else if(current_action == FWD_RIGHT)
45     {
46         fwd_right();
47     }
48     else if(current_action == 0)
49     {
50         stop();
51     }
52     else if(current_action == 0x14 && current_action != last_action)
53     {
54         decrease_speed();
55     }
56     else if(current_action == 0x15 && current_action != last_action)
57     {
58         increase_speed();
59     }
60     // Reglering
61     else if(current_action == 0x30)
62     {
63         float reglercopy;
64         cli();
65         reglercopy = reglerstyr;
66         sei();
67
68         drive_and_turn(reglercopy);
69     }
70     // Servon
71     else if(current_action == 0x31)
72     {
73         counter += 1;
74         if(counter >= timertime)
75         {
76             add1degree_joint(current_joint);
77             counter = 0;
78             _delay_ms(50);
79         }
80     }
81     else if(current_action == 0x32)
82     {
83         counter +=1;
84         if(counter >= timertime)
85         {
86             sub1degree_joint(current_joint);
87             counter = 0;
88             _delay_ms(50);
89         }
90     }
91 }
92 }
```

**Listing 3:** Exempelkod hur styrenheten styr servon.

```

1 void USART_Transmit( unsigned char* data, unsigned int size )
2 {
3     PORTD |= 0b100;
```

```

4
5 ;
6 // Lgger data i en buffer och skickas
7 for(unsigned int i = 0; i < size; i++)
8 {
9   while ( !( UCSR0A & (1<<UDRE0)) )
10  ;
11  UDR0 = (unsigned char) data[i];
12 }
13
14 while ( !( UCSR0A & (1<<TXC0)) )
15 ;
16 UCSR0A |= 1<<TXC0;
17 }
18
19 void move_servo(unsigned int ID, unsigned int Angle)
20 {
21
22   unsigned int header = 0xFF; //B rja varje med 2st 0xFF
23   unsigned int Length = 0x5; // 2 + antal P
24   unsigned int Instruction = 0x03; // skriv
25   unsigned int P1 = 30; // Address att skriva till
26   unsigned char P2 = (unsigned char)Angle;
27   unsigned char P3 = (unsigned char)(Angle>>8);
28   unsigned int Checksum = ~ (ID + Length + Instruction + P1 + P2 + P3);
29
30   unsigned char data[] = {header, header, ID, Length, Instruction, P1, P2, P3, Checksum};
31   unsigned int data_size = sizeof(data) / sizeof(data[0]);
32   USART_Transmit(data, data_size);
33 }

```

**Listing 4:** Exempelkod för styrenhetens linjeföljning.

```

1 void drive_and_turn(float turnvalue)
2 {
3   PORTA = 0b1000;
4   speed = reglerspeed;
5   if(speed + turnvalue < 0xFF && speed + turnvalue > 0)
6   {
7     if(turnvalue > 0)
8     {
9       OCR2A = speed + turnvalue;
10    }
11   else if(turnvalue < 0)
12   {
13     OCR2A = speed + turnvalue * 1.25;
14   }
15   else
16   {
17     OCR2A = speed;
18   }
19 }
20 else if(speed + turnvalue > 0xFE)
21 {
22   OCR2A = 0xFE;
23 }
24 else if(speed + turnvalue < 0)

```

```

25  {
26    OCR2A = 0;
27  }
28
29
30
31 if(speed - turnvalue < 0xFF && speed - turnvalue > 0)
32 {
33   if(turnvalue < 0)
34   {
35     OCR2B = speed - turnvalue;
36   }
37   else if(turnvalue > 0)
38   {
39     OCR2B = speed - turnvalue * 1.25;
40   }
41   else
42   {
43     OCR2B = speed;
44   }
45
46 }
47 else if(speed - turnvalue > 0xFE)
48 {
49   OCR2B = 0xFE;
50 }
51 else if(speed - turnvalue < 0)
52 {
53   OCR2B = 0;
54 }
55
56 }

```

**Listing 5:** Exempelkod för regleringen.

```

1 def PDController(front_error, back_error, KP, KD):
2   derivative = KD * (front_error - back_error)
3   proportional = KP * back_error
4   output = proportional + derivative
5   return output

```

**Listing 6:** Exempelkod hur delsystem i sensormodulen initieras.

```

1 void init_IR()
2 {
3   ADMUX = (0<<REFS1) | (0<<REFS0) | (1<<ADLAR) | (1<<MUX2) | (1<<MUX1) | (0<<MUX0);           // AREF,
4   left-shift, ADC6.
5   ADCSRA = (1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
6   // Activate ADC, Prescaler 128.
7
8 void init_gyro()
9 {
10  ADMUX = (0 << REFS1) | (0<<REFS0) | (1<<ADLAR) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0);           // AREF,
11  left-shift, ADC7.

```

```

11    ADCSRA = (1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
12    // Activate ADC, Prescaler 128.
13 }
14
15 void init_line_front()
16 {
17    DDRA |= 0x0F;                                // Ports choice of sensor. Might change
18    DDRD |= 0x20;                                // Port for enable
19
20    ADMUX = (0<<REFS1) | (0<<REFS0) | (1<<ADLAR) | (1<<MUX2) | (0<<MUX1) | (1<<MUX0);           // AREF,
21    left-shift, ADC5.
22    ADCSRA = (1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
23    // Activate ADC, Prescaler 128.
24 }
25
26 void init_line_back()
27 {
28    DDRD |= 0x1F;                                // Ports for enable and choice of sensor.
29    ADMUX = (0<<REFS1) | (0<<REFS0) | (1<<ADLAR) | (1<<MUX2) | (0<<MUX1) | (0<<MUX0);           // AREF,
30    left-shift, ADC4. VIRA TILL 36
31    ADCSRA = (1<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);
32    // Activate ADC, Prescaler 128.
33 }
34
35 void init_interrupt()
36 {
37    EICRA |= (1<<ISC01) | (1<<ISC00);          // Interrupt on rising edge.
38 }
39
40 void init_SPI()
41 {
42    DDRB = (1 << DDB6);                          // Pin 7 (MISO) to output.
43    SPCR = (1 << SPIE) | (1 << SPE) | (0 << DORD) | (0 << CPOL) | (0 << CPHA);           // Activate SPI-buss.
44 }

```

**Listing 7:** Exempelkod för hur en Bluetooth-anslutning görs skriven på klienten (PC).

```

1 import socket
2
3 hostMACaddress = 'B8:27:EB:E9:12:27'
4 port = 4
5 global s
6 s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM) //create socket
7
8 s.connect((hostMACaddress, port))          // trying to connect

```

**Listing 8:** Exempelkod för hur en Bluetooth-anslutning görs skriven på Raspberry Pi.

```

1 import socket
2
3 hostMACaddress = 'B8:27:EB:E9:12:27'

```

```
4 port = 4
5 s = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM) //create socket
6 s.bind((hostMACaddress, port))    //Binding socket to address and port of the Raspberry Pi
7 s.listen(1)                      //Put socket in listening mode
```

**Listing 9:** Exempelkod för hur data skickas över Bluetooth från klienten (PC).

```

1 def sendbyte(byte):
2     try:
3         s.send(byte.to_bytes(1, 'big')) //Skicka 1 byte via socketen s
4     except Exception as e:
5         print(f"Fel vid sändning av byte: {e}")
6
7 def receive_data():
8     try:
9         data = s.recv(1) // Vänta på att ta emot 1 byte via socketen s
10        if data:
11            if len(data) == 1:
12                return data[0]
13            else:
14                return data
15        except Exception as e:
16            print(f"Fel vid mottagning av data: {e}")
17    return None
18
19
20 def send_and_receive(command):
21     try:
22         sendbyte(command)
23         received = receive_data()
24         return received
25     except Exception as e:
26         return
27
28 def data_loop(window):
29     data = bt.send_and_receive(command) // command = heltalet
30     data_list.append(data)
31     window.after(100, lambda: data_loop(window)) //lopa funktionen varje tiondelssekund

```

**Listing 10:** Exempelkod för hur data tas emot och sedan skickas tillbaka över Bluetooth för Raspberry Pi:n.

```

1 data = client.recv(1)
2 if data == "":
3     client.send(response.to_bytes(1, 'big')) //response = vald data att skicka tillbaka

```

**Listing 11:** Skapa graf - funktionen i snabbaste-vägen-koden.

```

1 def create_graph(width, height):
2     #creates a dictionary with all neighbours downward, right, left and over to all nodes
3     graph = {}
4     nx = width + 1
5     ny = height + 1
6     for y in range(ny):
7         for x in range(nx):
8             node = y * nx + x + 1
9             graph[node] = []
10            # right neighbour
11            if x < nx - 1:
12                graph[node].append(node + 1)
13            # down neighbour
14            if y < ny - 1:
15                graph[node].append(node + nx)

```

```

16     # left neighbour
17     if x > 0:
18         graph[node].append(node - 1)
19     # up neighbour
20     if y > 0:
21         graph[node].append(node-nx)
22 return graph

```

**Listing 12:** Create distance matrix funktionen i snabbaste-vägen-koden

```

1 def create_distancematrix(graph, nodes):
2     nr_nodes = len(nodes)
3     matrix = [[0]*nr_nodes for _ in range(nr_nodes)]
4     for i in range(nr_nodes):
5         for j in range(nr_nodes):
6             if i != j:
7                 path = bfs(graph, nodes[i+1], nodes[j+1], 0, 0)
8                 if i == 0:
9                     matrix[i][j] = len(path) - 2 if path else float('infinity')
10                else:
11                    matrix[i][j] = len(path) - 1 if path else float('infinity')
12
13 return matrix

```

**Listing 13:** Bredden-Först-Sökning i snabbaste-vägen koden

```

1 def bfs(graph, startnode, goalnode, next_node, last_node):
2
3     paths = []
4     for i in range(len(startnode)):
5         queue = deque([[startnode[i]]])
6         visited = set()
7         while queue:
8             path = queue.popleft()
9             node = path[-1]
10            if node in visited:
11                continue
12            visited.add(node)
13            for neighbor in graph[node]:
14                new_path = path + [neighbor]
15                if [node, neighbor] == mlnod or [neighbor, node] == goalnode or [neighbor] == goalnode:
16                    paths.append(new_path + ['goal'])
17                    queue.append(new_path)
18
19
20            if last_node == 0:
21                return min(paths, key=len)
22
23            if len(goalnode) == 1 and goalnode[0] in startnode and last_node != 1: #om varan ligger
24                j mte nod 1 och inte kommer fr n nod 1 ...
25                for i in startnod:
26                    if [i] != goalnode:
27                        return [goalnode[0], 'goal'] #... ska den ka ut
28
29            shortest_dist = min(len(l) for l in paths)
30            shortest_paths = [l for l in paths if len(l) == shortest_dist]

```

```

30
31   if len(shortest_paths) > 1:
32     for i in shortest_paths[:]:
33       if i[0] != next_node:
34         shortest_paths.remove(i)
35     shortest = min(shortest_paths, key=len)
36   if shortest[0] != next_node:
37     shortest = [next_node] + shortest
38
39   return shortest

```

**Listing 14:** Main-loopen i Raspberry Pi:n.

```

1 def main():
2   global nav_plan
3   nav_plan = 0
4   s = bt.init_bluetooth()
5   spi_styr, spi_sensor = spi.initSpi()
6
7   bt_thread = threading.Thread(target=bluetooth_listener, args=(s, spi_styr, spi_sensor),
8     daemon=True)
9   bt_thread.start()
10
11   KP, KD = 100, 100
12
13   while True:
14     old_Automatic = auto.Automatic
15     if auto.Automatic:
16       autonomous_loop(spi_styr, spi_sensor, KP, KD)
17       time.sleep(0.005)
18     if(old_Automatic == True and auto.Automatic == False):
19       spi.send_spi(spi_styr, 0)
20
21 if __name__ == "__main__":
22   main()

```

**Listing 15:** Exempelkod för Autonomous-loop i Raspberry Pi:n.

```

1 def autonomous_loop(spi_styr, spi_sensor, KP, KD):
2   global nav_plan
3   global new_plan
4   global goal_collected
5   global has_seen_roadmark
6   global nodeorder
7   global obstacles
8   global goods_deposited
9   try:
10     next_move = nav_plan[0]
11   except:
12     return
13   roadmark_status = auto.check_roadmark(spi_sensor)
14   m_lnoder = hm.get_m_lnoder()
15
16
17   if(auto.check_obstacle(spi_sensor)): //hinder
18     spi.send_spi(spi_styr, 0) //stanna
19

```

```

20     current_node, next_node = fw.find_location(nav_plan, nodeorder) //h mta aktiva noder
21     obstacles.append([current_node, next_node]) //l gg till hinder i lista
22     m_lnoder[1] = [current_node]
23     nav_plan, nodeorder = hm.update_path_obstacles(obstacles, m_lnoder) //ber kna ny v g
24     new_plan = True //skicka styrbeslut till PC
25     auto.rotate_right_180(spi_styr, spi_sensor) //rotera f r att undvika hinder
26
27 if(roadmark_status == 0):
28     has_seen_roadmark = False
29     auto.control_loop(spi_styr, spi_sensor, KP, KD)
30
31 elif(roadmark_status == 1): // dags att plocka vara t h ger
32     if(next_move == "plocka"):
33         goal_collected = True
34         spi.send_spi(spi_styr, 0) //stanna och plocka vara
35         auto.pick_right(spi_styr)
36         current_node, next_node = fw.find_location(nav_plan, nodeorder)
37         hm.remove_goal_node(current_node, next_node)
38         next_move = nav_plan.popleft()
39         if(nav_plan[0] == "v nd"):
40             auto.rotate_right_180(spi_styr, spi_sensor)
41             nav_plan.popleft()
42     else:
43         auto.drive_fwd(spi_styr) // K r f rbi plockstation

```

**Listing 16:** Exempelkod för bluetooth-control-loop i Raspberry Pi:n.

```

1 def bluetooth_control_loop(data, client, spi_styr, spi_sensor):
2     global new_plan
3     global nav_plan
4     global m_lnoder
5     global goal_collected
6     global goods_deposited
7
8     try:
9         if(data == "20"): // ndrar aktuell armled
10            data = client.recv(size)
11            try:
12                spi.send_spi(spi_styr, [0x20] + list(data))
13            except:
14                print("Invalid data 1")
15
16        elif data == "60": //IR-data
17            response = spi.send_spi(spi_sensor, 0)
18            client.send(response.to_bytes(1, 'big'))
19        elif data == "61": //Reflex-data
20            response = spi.send_spi(spi_sensor, 1)
21            client.send(response.to_bytes(1, 'big'))
22        elif data == "67": //kalibrera linjesensor
23            spi.send_spi(spi_sensor, 7)
24            time.sleep(0.01)
25
26        elif data == "81": //upplockade varor
27        if goal_collected:
28            client.send(b'\x01')
29            goal_collected = False
30        else:

```

```
31     client.send(b'\x00')
32
33 elif data == "82": //avbryta autonomt l ge i gui
34     if goods_deposited:
35         client.send(b'\x01')
36         goods_deposited = False
37     else:
38         client.send(b'\x00')
39
40 else:
41     try:
42         response = spi.send_spi(spi_styr, int(data, 16)) //manuell styrning
43     except:
44         print("Invalid data 2")
45     except:
46         print("Disconnected, looking for new socket")
```

## 7.4 Tabeller

**Tabell 8:** Lista över dataöverföring mellan Raspberry Pi och sensorenhet.

Data som Raspberry Pi skickar till sensorenheten	Betydelse
0	Skicka IR-data från sensorenheten till Raspberry Pi
1	Skicka främre reflex-data till Raspberry Pi
2	Skicka bakre reflex-data till Raspberry Pi
3	Starta gyro
4	Stoppa gyro
5	Skicka vinkel från nolläge för gyrot till Raspberry Pi
6	Skicka 0, 1, 2 eller 3 beroende på roadmark status till Raspberry Pi

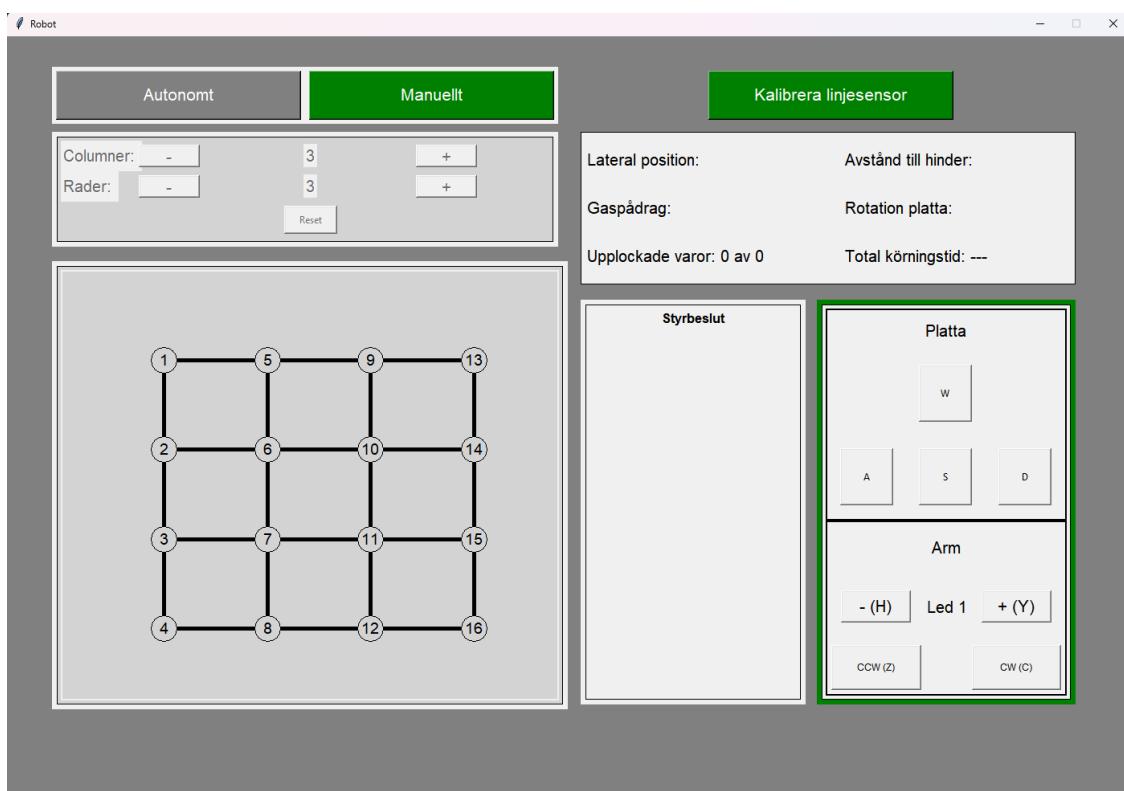
**Tabell 9:** Lista över dataöverföring mellan Raspberry Pi och styrenhet.

Data som Raspberry Pi skickar till styrenheten	Betydelse
0	Stanna
1	Kör framåt
2	Rotera vänster
3	Kör bakåt
4	Rotera höger
5	Kör vänster framåt
6	Kör höger framåt
20	Ändra aktuell armled
30	Meddela att efter detta värde kommer reglerinfo
31	Rotera gripklo moturs
32	Rotera gripklo medurs
40	Skicka gaspådrag vänster från styrenheten till Raspberry Pi:n
41	Skicka gaspådrag höger från styrenheten till Raspberry Pi:n
50	Meddela att efter detta värde kommer info om armens aktuella läge
60	Meddela att efter detta värde kommer info om hur armen ska flyttas

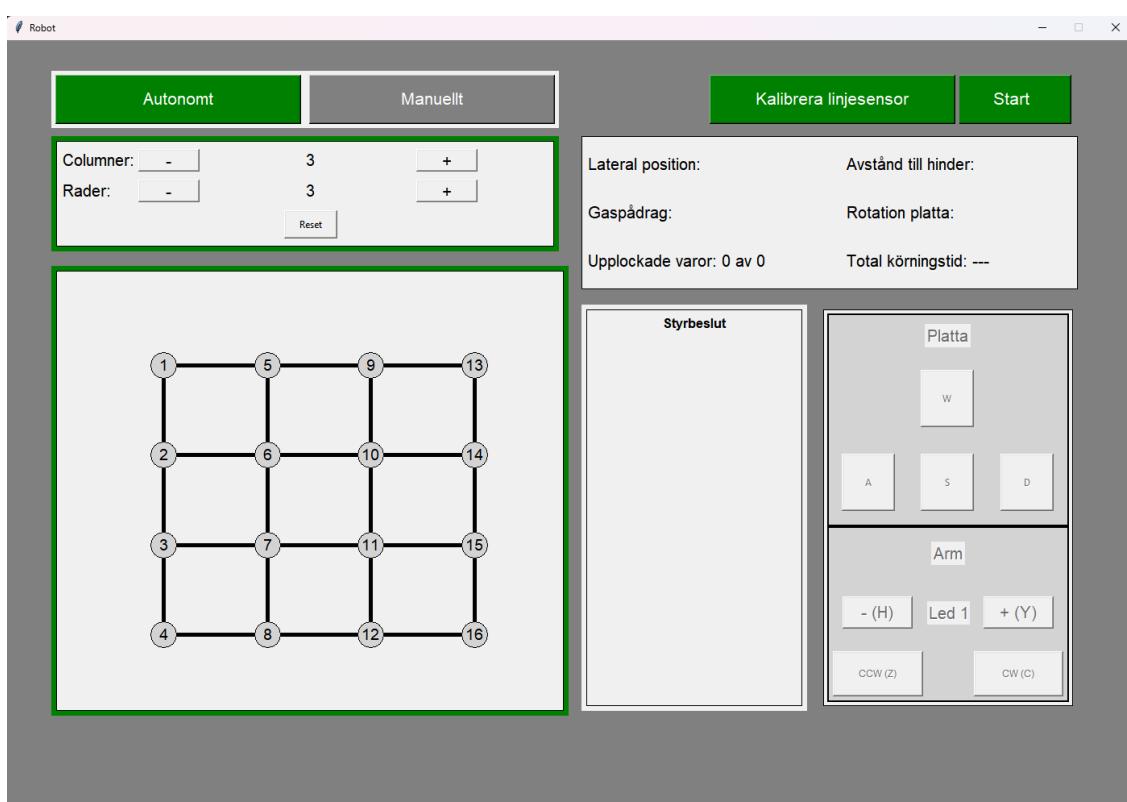
**Tabell 10:** Lista över dataöverföring mellan PC och Raspberry Pi.

Data som PC skickar till Raspberry Pi	Betydelse
0	Skicka vidare värdet 0 till styrenheten
1	Skicka vidare värdet 1 till styrenheten
2	Skicka vidare värdet 2 till styrenheten
3	Skicka vidare värdet 3 till styrenheten
4	Skicka vidare värdet 4 till styrenheten
5	Skicka vidare värdet 5 till styrenheten
6	Skicka vidare värdet 6 till styrenheten
20	Ändra aktuell armled
31	Skicka vidare värdet 31 till styrenheten
32	Skicka vidare värdet 32 till styrenheten
60	Skicka IR-data från Raspberry Pi till PC
61	Skicka reflex-data från Raspberry Pi till PC
62	Skicka gyro-data från Raspberry Pi till PC
63	Starta gyro
64	Stoppa gyro
65	Skicka gaspådrag höger från Raspberry Pi till PC
66	Skicka gaspådrag vänster från Raspberry Pi till PC
67	Kalibrera reflexsensor
70	Meddela att efter detta värde kommer lagerinfo
80	0 eller 1 skickas från Raspberry Pi till PC beroende på status på styrbeslut
81	0 eller 1 skickas från Raspberry Pi till PC beroende på status på upplockade varor
82	0 eller 1 skickas från Raspberry Pi till PC beroende på status autonom körning
99	Växla läge mellan autonomt och manuellt

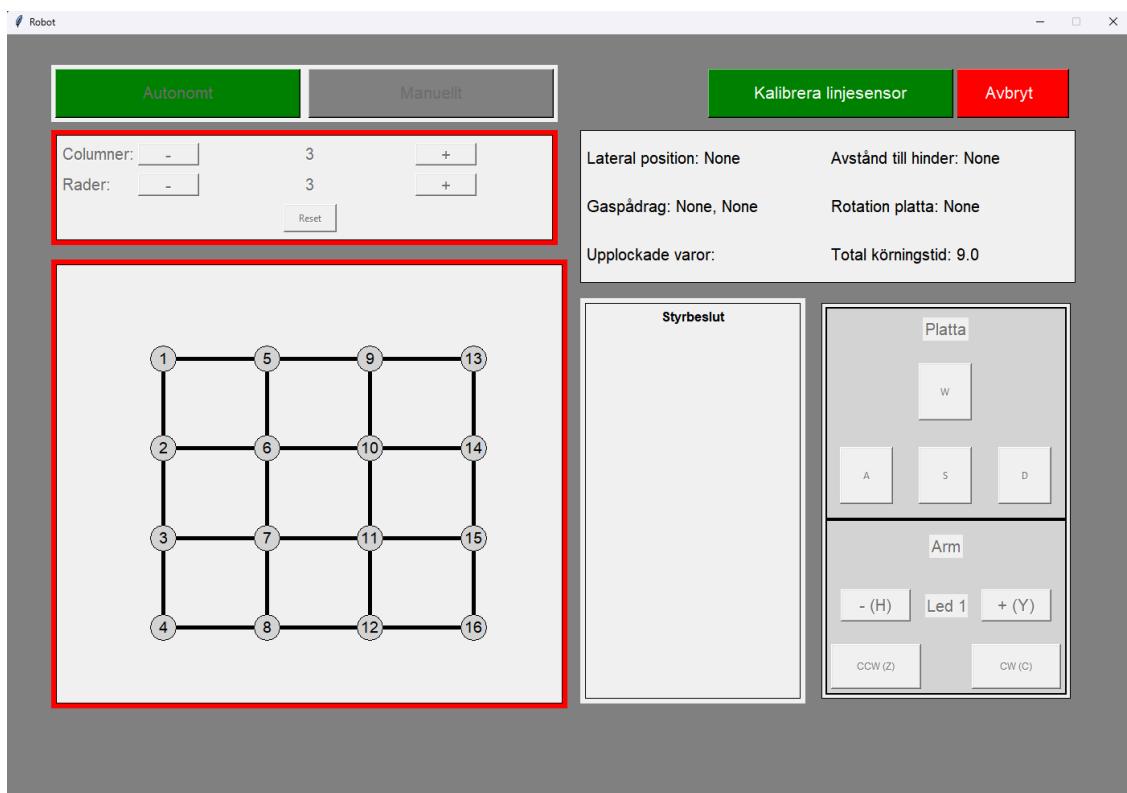
## 7.5 Övrigt



**Figur 8:** Användargränssnittet vid uppstart, även manuellt läge.



**Figur 9:** Användargränssnittet i autonomt läge.



**Figur 10:** Användargränssnittet vid start av körsättning i autonomt läge.