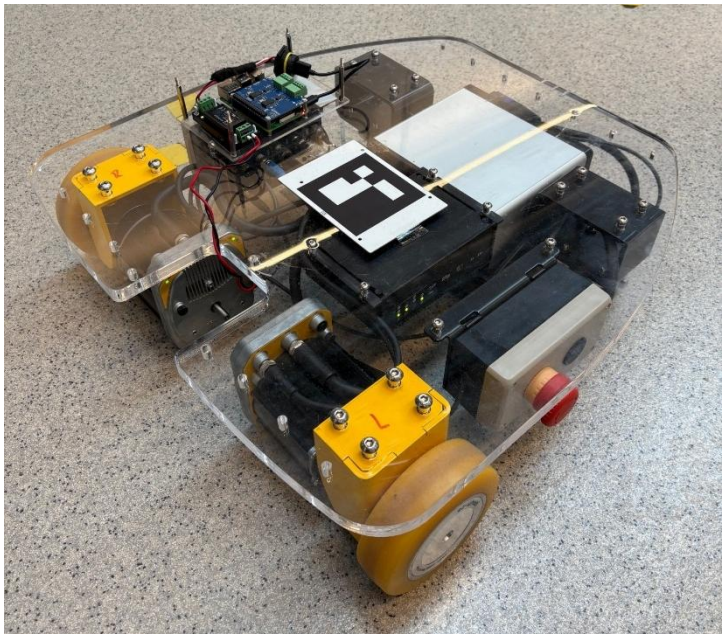
CHALMERS



Implementering av styrsystem för robot i GPSS-miljö

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

THEA NILSSON
TILDA POROPAT

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

EXAMENSARBETE INOM MEKTATONIK 2025

Implementering av styrsystem för robot i GPSS-miljö

THEA NILSSON

TILDA POROPAT

CHALMERS

Intuitionen för elektroteknik

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2025

Implementering av styrsystem för robot i GPSS-miljö

Examensarbete inom högskoleingenjörsprogrammet Mekatronik
THEA NILSSON
TILDA POROPAT

Handledare: Kristofer Bengtsson, Volvo Lastvagnar
Examinator: Veronica Olesen, Institutionen för elektroteknik

Institutionen för Elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telefon: +46 (0)31-772 1000

Erkännanden, dedikationer och liknande personliga uttalanden återspeglar författarens egna åsikter.

Förstasida: Bild på roboten [egen bild]
Göteborg, Sverige 2025

Förord

Detta projekt har genomförts som ett examensarbete för utbildningen i högskoleingenjör Mekatronik på Chalmers Tekniska Högskola. Examensarbete motsvarar 15hp av utbildningen på 180hp. Examensarbetet har genomförts på Volvo Lastvagnar.

Vi vill tacka vår handledare Kristofer Bengtsson som varit till stor hjälp under arbetets gång. Vi vill även tacka vår examinator och handledare på institutionen för elektronik på Chalmers, Veronica Olesen, för stöttningen och hjälpen med arbetets struktur.

Thea Nilsson och Tilda Poropat, maj 2025, Göteborg

Implementering av styrsystem för robot i GPSS-miljö

THEA NILSSON
TILDA POROPAT
Institutionen för elektroteknik
Chalmers Tekniska Högskola

Sammanfattning

Inom tillverkningsindustri är målet att utveckla och effektivisera processer. En metod som idag används mer och mer hos tillverkningsindustrier för att effektivisera och modernisera processer är automation. Automation ökar produktiviteten i processerna och leder till förbättrad noggrannhet och kvalité.

Arbetet har utförts på Volvo Lastvagnar som idag har börjat automatisera sitt materialflöde av artiklar från lagret ut till produktionslinjen. Detta sker med hjälp av autonoma transportrobotar som styrs via ett system som använder kameror i taket för att kunna identifiera robotens position och på ett säkert sätt lotsa robotarna fram genom fabriken. Arbetets syfte har varit att programmera och integrera en ny typ av autonom transportrobot i Volvos befintliga system. Denna rapport beskriver de metoder och tillvägagångssätt som använts för att programmera roboten.

En färdigutvecklad labbrobot, framtagen för utvecklingsändamål har använts. Dessutom har arbetet enbart utförts i labbmiljö. Lösningen har alltså inte implementerats i industrimiljö. Labbmiljön medför även andra avgränsningar så som ett begränsat antal kameror i taket och en begränsad yta att testköra på.

Det var möjligt att implementera labbroboten i Volvos system så att roboten på ett säkert sätt kunde köra en förbestämd rutt mellan punkter, bestämda av användaren, och samtidigt undvika hinder. Resultatet medför möjligheten att utöka användningen av förarlösa transportrobotar för olika applikationer och därmed effektivisera processer.

Nyckelord: Robot, förarlös transportrobot, AGV, navigationssystem, GPSS, automation, styrsystem.

Control System for GPSS Robot

THEA NILSSON
TILDA POROPAT
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In the manufacturing industry, the aim is to develop and make the processes more efficient. One method that is increasingly being used to modernize and improve efficiency in manufacturing is automation. Automation increases the productivity in the processes and leads to improved accuracy and quality.

This project was carried out at Volvo Trucks, which has begun automating its transportation of components to the production line. This is done using autonomous transportation robots, which navigates using cameras mounted in the ceiling. The robots can safely navigate through the factory. The purpose of this project was to programme and integrate an autonomous robot into Volvo’s existing system. This report describes the methods and approaches used to programme the robot in Python.

A fully developed robot, designed for development purposes, was used. Moreover, the project has been carried out in a lab environment and was not implemented in an industrial setting. The lab environment introduced limitations, such as a restricted number of ceiling cameras and a limited test area.

It was possible to integrate the lab robot in Volvo's system in a way that allowed it to safely drive a predetermined route between points, while avoiding obstacles.

The result makes it possible to expand the use of driverless transport robots in various applications, improving the efficiency.

Key words: Robot, Driverless Transportation Robot, AGV, Navigation system, GPSS, Automation, Control System.

Innehållsförteckning

Terminologi/ Förkortningar

GPSS. Generic Photo-based Sensor System

ROS. Robot Operating System

JSON. JavaScript Object Notation

Redis. Remote Dictionary Server

SSH. Secure Shell

AGV. Automated Guided Vehicle

VSC. Visual Studio Code

SWD. Safety Wheel Drive

CAN. Controller Area Network

1. Inledning

Företag inom tillverkningsindustrin jobbar ständigt med att effektivisera och utveckla sina tillverkningsprocesser. Detta är viktigt för att företagen ska vara konkurrenskraftiga. För att effektivisera tillverkningsprocesser har robotteknik och automation implementerats mer och mer i dagens tillverkningsindustrier. Detta leder till ekonomisk vinning på grund av förbättrad produktivitet, reducerade lönekostnader och minskad risk för mänskliga misstag. Det finns även andra fördelar så som högre säkerhet och jämnare kvalité.

Automatisering implementeras inte bara på själva tillverkningsprocessen utan kan användas på sidoprocesser så som den interna logistiken och materialflödet. Förarlösa transportfordon, så kallade AGV, *Automated Guided Vehicle,* används idag i stigande grad för att på ett mer effektivt sätt kunna transportera artiklar och material inom produktionen.

1.1 Bakgrund

I Volvos fabrik i Tuve tillverkas idag många olika modeller av lastbilar samtidigt. För att klara av detta är det viktigt att har rätt artiklar på plats vid produktionslinjen vid rätt tillfälle, vilket innebär stora logistikutmaningar [1]. För att underlätta och effektivisera denna process har Volvo själva tagit fram transportrobotar som automatiskt kan köra mellan lagret och produktionslinjen. Robotarna styrs via ett styrsystem som även det är framtaget av Volvo. Styrsystemet heter GPSS *Generic Photo-based Sensor System* som består av kameror som sitter installerade i taket av fabriken. Kamerorna läser av robotarnas positioner via en markör som sitter monterade på ovansidan av robotarna. Därefter guidas robotarna på ett säkert och effektivt sätt genom fabriken via GPSS. Anledningen till att Volvo själva valt att utveckla autonoma transportsystem var eftersom de ansåg att redan existerande och etablerade transportrobotar inte höll tillräckligt hög standard.

1.2 Syfte

Syftet med projektet är att implementera en ny autonom transportrobot i GPSS-miljön. Roboten som ska implementeras är en färdig labbrobot och syftet är att undersöka hur det är möjligt att implementera roboten i Volvos befintliga system och om detta i sin tur skulle leda till effektivare och billigare produktion.

1.3 Mål

Arbetets huvudmål är att utveckla ett navigeringssystem som med hjälp av robotens hjulodometri och kameraövervakningen från GPSS ska kunna bestämma robotens positionering utifrån GPSS:s koordinatsystem. Roboten ska även kunna förflytta sig mellan förbestämda punkter i GPSS samtidigt som den ska undvika hinder.

För att bryta ner detta mål har följande delmål satts upp:

- Roboten ska integreras i GPSS
- ROS kommunikation ska användas
- Roboten ska programmeras i Python
- Både manuell och automatisk styrning av roboten ska vara tillgängligt
- Roboten ska stanna vid hinder
- Robotens positionering ska ske utifrån både dess odometri samt kameror monterade i taket
- Roboten ska navigera utifrån kamerornas fasta koordinatsystem
- Datakommunikationen mellan roboten och GPSS ska ske utifrån ett redan existerande GPSS-protokoll
- Möjlighet att koppla in labbroboten i den verkliga fabriken i framtiden ska finnas, därför måste standarder följas.

1.4 Avgränsningar

Projektet avgränsas för att arbetet inte ska bli för brett och för att göra projektet mer precist. Typen av robot som skall användas i projektet är redan förbestämt, vilket innebär att arbetet inte kommer att ta hänsyn till hur en annan robot skulle implementeras i systemet. Projektet ska enbart utföras i labbmiljö, men implementeringen ska samtidigt följa förbestämda protokoll för att möjliggöra framtida implementering i fabriken. Det är även förbestämt att roboten ska programmeras i programmeringsspråket Python. Inga ekonomiska hänsynstaganden görs.

1.5 Precisering av frågeställningen

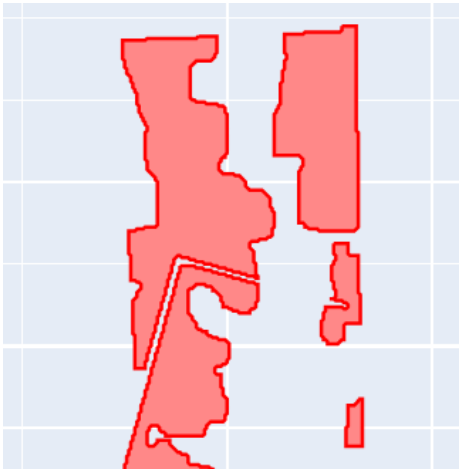
Roboten ska implementeras i GPSS-miljön. Vid implementation behöver roboten kommunicera med GPSS, hur sker den kommunikationen? Genom att använda sig av robotens hjulodometri och olika kameror ska positioneringen för roboten bestämmas. Hur ska navigeringen till målpunkten genomföras och med vilken precision sker navigationen? Roboten ska även undvika hinder som upptäcks av kamerorna, med vilken säkerhet undviker roboten dessa hinder?

2. Teknisk bakgrund

För att få en djupare förståelse för arbetet beskriver följande stycke de system och komponenter som har används under projektets gång. Nedan listas både hårdvara och mjukvara samt olika teorier som tillsammans bidragit till projektets funktionalitet.

2.1 GPSS

GPSS *Generic Photo-based Sensor System* är ett positioneringssystem som Volvo använder för att kunna styra autonoma transportrobotar i fabriken mellan produktionslinjen och lagret [1]. GPSS består av kameror installerade i taket på fabriken. Kamerorna kan i realtid via en markör som sitter monterad på roboten läsa av dess position. Kamerorna övervakar fabriksgolvet för att identifiera andra objekt så som människor och truckar. Med hjälp av informationen som kamerorna samlar in skapas en karta som visar en helhetsbild av fabriken i realtid, se figur 1. Kartan i sin tur används för navigering för de autonoma transportrobotarna [2]



Figur 1: Karta över fabriksgolvet, där det röda symboliserar hinder. Skärmavbild från GPSS

De autonoma transportrobotarna lotsas sedan utifrån denna information och får kommandon om vart de ska köra och när det är säkert att röra sig. Användaren kan via gränssnittet i GPSS se realtidskartan över fabriken. I gränssnittet är det även möjligt för användaren att ange målpunkter och körvägar samt hastigheter för de olika robotarna som styrs via systemet.

2.2 ROS

ROS *Robot Operating System,* är ett ramverk som tagits fram för att förenkla robotprogrammering genom att dela upp funktioner i noder, vilket betyder fristående och självständiga programenheter som enbart används för att lösa en specifik uppgift [3]. ROS används bland annat för kommunikation mellan olika delar hos roboten, för att till exempel skicka styrkommandon eller ta emot sensorinformation.

2.3 CAN

CAN *Controller Area Network,* används för att flera olika noder, exempelvis sensorer, ställdon eller kontrollenheter, ska kunna kommunicera tillsammans [4]. Noderna är kopplade till CAN-bussen där alla noder kan både skicka och ta emot meddelandena. När en nod skickar ett meddelande tas det emot av alla andra noder men med hjälp av informationen i meddelandet är det endast den avsedda mottagaren som svarar.

2.4 Raspberry Pi

Raspberry Pi är en liten enkortsdator som består av en processor med ARM-arkitektur [5]. En Raspberry Pi har inget eget minne så som en hårddisk. Den använder sig i stället av ett externt SD-kort eller ett microSD-kort för filhantering. En Raspberry Pi har även olika typer av USB-portar och micro-HDMI portar för att enkelt kunna interagera med den. Det går även fjärrstyra Raspberry Pi:n via en annan dator med hjälp av SSH *Secure Shell,* vilket är ett protokoll som möjliggör fjärranslutning mellan två datorer. Det är alltså via SSH möjligt att styra Raspberry Pi:n genom att köra kommandon via terminalen från sin egen dator. Operativsystemet hos en Raspberry Pi är oftast baserat på Linux.

2.5 Eazy wheel - SWD® Starter Kit

SWD® Starter Kit är en färdig labb-robot av märket Ez-wheel som består av två motorer som driver robotens framhjul, två nödstopp, ett för varje motor, en Linux dator, en säkerhetsscanner och ett uppladdningsbart 24 volts batteri. Linux datorn är en industriell dator som använder ROS och drivrutiner för att kontrollera robotens rörelse

genom att styra motorerna. Alla delar som tillhör roboten sitter fastmonterat under en plastskiva som utgör robotens chassi [6]. Även en tillhörande joystick finns för att kunna styra roboten manuellt.

Robot-kittet har tagits fram för att underlätta och snabba på utvecklingen av AGV:er hos företag. Roboten är konstruerad på ett sätt där driv- och säkerhetsfunktioner är integrerade direkt i hjulen vilket underlättar utvecklingen eftersom tekniska komponenter redan är samordnade och därmed minimeras komplexiteten i konstruktion och utveckling processen. Robotkittet underlättar även utvecklingsprocessen då hela plattformen alltså robotens olika delar redan existerar vilket gör att man endast behöver lägga fokus på programutvecklingen.

2.6 Proportionell reglering

Proportionell reglering är en linjär form av reglering [7]. Reglering sker proportionellt mot reglerfelet. En förstärkningsfaktor K_P multipliceras med systemets reglerfel e. Beroende på vilken förstärkningsfaktor som används reagerar systemet olika. Ett lågt K_P bidrar till ett stabilt men långsamt system medan ett högt K_P bidrar till ett instabilt system men med ökad snabbhet [7].

Ekvation (1) beskriver sambandet mellan, styrsignalen u, styrsignalens normalvärde u_0, förstärkningsfaktorn K_P och reglerfelet e.

$$u = u_0 + Kp \cdot e \qquad (1)$$

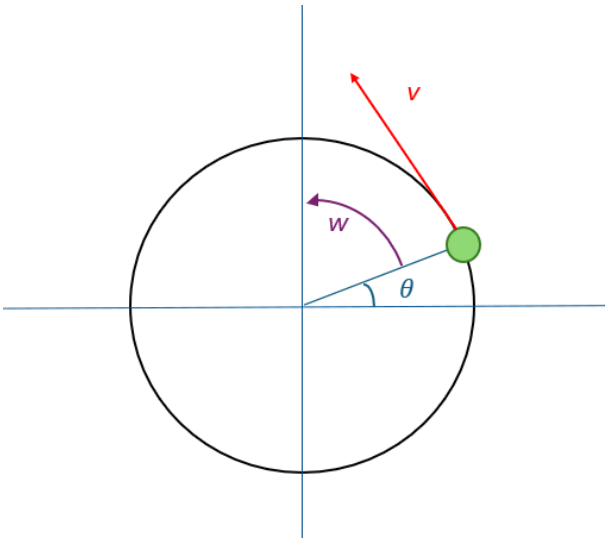
2.7 Odometri

Odometri är en intern form av positionering och lokalisering utan påverkan från yttre faktorer [8]. Hjulodometri beror helt och hållet på hjulen och är vanligt för tvåhjuls- och fyrhjulsfordon. För att möjliggöra användandet av odometrin behöver hjulen kunna manövreras oberoende av varandra i olika hastigheter och riktningar. Metoden är baserad på hjulkodare som känner av antalet varv varje hjul snurrat [9] . Antalet varv används sedan i en dynamisk modell för att avgöra den nuvarande positionen i förhållande till startpositionen.

En nackdel som förekommer vid användandet av odometri är positionsdrift. Över tid blir mätningarna mer felaktiga [8]. I miljöer där hård terräng eller glatta ytor förekommer skapas problem för odometrin. Odometrin kan inte ta hänsyn till hjulslirning. Den lämpar sig inte ensamt för långa och exakta lokaliseringssystem. En fördel med odometrin är att det är en relativ enkel och billig lokaliseringsteknik.

2.8 Twist

Twist är ett meddelande i ROS som ofta används för att styra en robot. Ett twist-meddelande består två delar, en vektor för vinkelhastighet och en vektor för linjärhastighet [10]. De två vektorerna består av 3 riktningar var. Via twist-meddelanden kan linjär- och vinkelhastighet bestämmas i ett tredimensionellt rum. Den linjära hastigheten beskriver hur fort förflyttningar sker från en punkt till en annan och mäts i m/s medan vinkelhastighet beskriver med vilken hastighet som rotationer sker och mäts i rad/s, se figur 2. Det är linjärhastigheten som styr robotens hastighet framåt och vinkelhastigheten som styr hastigheten roboten roterar med.



Figur 2: Visar skillnaden mellan linjärhastighet, som representeras av v i figuren, och vinkelhastighet, som representeras av ω i figuren.

2.9 JSON

JSON *Java Script Object Notation* är ett format för datautbyte. JSON är enkelt, lättläst och används ofta för att överföra data mellan en server och en webbapplikation samt för att strukturera data [11].

2.10 Redis

Redis *Remote Dictionary Server* är en minnesbaserad databas som ofta används för att lagra data tillfälligt, för att skapa en meddelandekö och som en databas för realtidsdata [12]. Redis är snabbt eftersom det körs direkt i RAM-minnet i stället för på hårddisken.

2.11 Sensorfusion

Sensorfusion är en vanlig metod som används hos bland annat självkörande bilar och förarlösa robotar för att kunna positionera fordonet korrekt, tolka dess omgivning och identifiera hinder [13]. Metoden går ut på att kombinera data från olika källor för att få en mer tillförlitlig och exakt uppfattning av exempelvis en position eller omgivning. De olika källorna kan bland annat utgöras av kamera, LiDAR, radar, GNSS och IMU. En vanlig kombination av källor är positionsdata från robotens egen odometri och från en kamera. Kombinationen av dessa källor används för att få en mer exakt och stadig uppskattning av dess läge.

2.12 Single Exponential Smoothing-filter

Single Exponential Smoothing, *SES,* är ett exponentiellt utjämningsfilter och är en väl använd metod för att förutse framtida värden [14]. En fördel med att använda denna metod är att det är enkelt att tillämpa och har en låg kostnad.

En viktig parameter i SES-metoden är utjämningskonstanten α som anger hur stor del det tidigare och det faktiska värdet ska påverka det slutgiltiga filtrerade värdet. Utjämningskonstanten påverkar noggrannheten och det är därför viktigt att välja ett lämpligt värde för utjämningskonstanten [14].

Det filtrerade värdet F_{t+1} bestäms med hjälp av utjämningsfaktorn, det nuvarande värdet X_t och det tidigare värdet F_t. En kombination av det tidigare och det nuvarande

värdet utgör det filtrerade värdet. Beroende på hur stor utjämningsfaktorn är kommer de olika värdena utgöra olika stor procentuell påverkan på det filtrerade värdet.

I ekvation (2) visas formeln som används i SES-metoden.

$$F_{t+1} = \alpha \cdot X_t + (1 - \alpha) \cdot F_t \qquad (2)$$

2.13 Kvaternioner

Kvaternioner q är en utvidgning av de komplexa talen och tillämpas i ett tredimensionellt rum [15]. Kvaternioner möjliggör en definition för kvoten mellan två vektorer i ett tredimensionellt rum och beskrivs som i ekvation (3), där a, b, c och d är reella tal och i, j och k är enhetsvektorer.

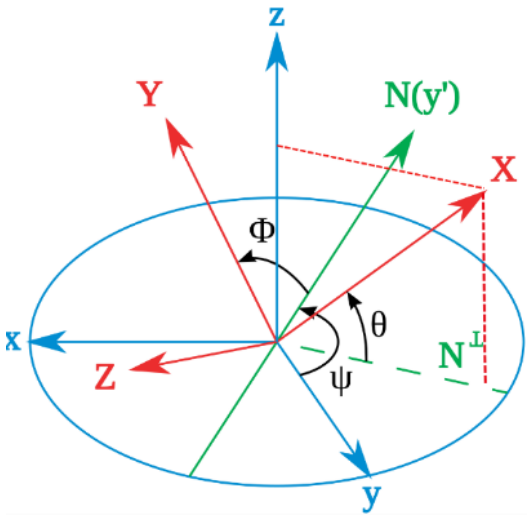
$$q = a + bi + cj + dk \qquad (3)$$

Kvaternioner kan användas för att representera tredimensionella rotationer.

2.14 Euler-vinklar

Euler-vinklar är tre vinklar som används för att beskriva rotationen av en stel kropp i förhållande till ett fast koordinatsystem [16]. Vinklarna kan definieras antingen genom geometri eller som en sammansättning av rotationer.

Euler-vinklarna representeras ofta av olika tecken, α eller ϕ benämner rotationen kring x-axeln, β eller θ benämner rotationen kring y-axeln och γ eller ψ benämner rotationen kring z-axeln, se figur 3. Dessa kan även benämnas som Yaw (vridning), Pitch (höjdjustering) och Roll (rullning) när de används för att beskriva rotationer och roteringar av objekt inom områden så som flygning, robotik och rymdteknik [16].



Figur 23[1]: Tydliggör vilka Euler-vinklar som relaterar till de olika axlarna i ett tredimensionellt koordinatsystem [16]

[1] Figur 3. "Eulerangles.svg" av Lionel Brits, licensierad under CC BY-SA 4.0, via Wikimedia Commons.

3. Metod

Metoden är uppdelad i 3 delar, de tekniska lösningar och principer som använts, systemets slutgiltiga uppbyggnad och sist test och verifiering.

Inledningsvis i avsnittet presenteras de tekniska lösningar som använts i arbetet kopplade till sitt respektive projektsteg. Därefter kommer ett stycke som beskriver hur dessa delar samverkar och hur hela systemet integreras, alltså hur robotens funktionalitet är uppbyggd. I slutet av metoden presenteras även de testmetoder som använts under projektets gång.

3.1 Tekniska lösningar och principer

De tekniska lösningarna och principerna som använts är uppdelat i följande 4 delar utifrån projektets projektsteg samt en inledande uppstart:

- Uppstart av robot
- Körning av robot via odometri
- Körning av robot via odometri i samverkan med GPSS
- Implementering av roboten i GPSS-miljön

Under varje projektsteg beskrivs de tekniska lösningar, principer och beräkningar som använts och implementerats för att uppnå målen. Strukturen på stycket medför en översikt där alla lösningar är koppade till sitt specifika projektsteg som i sig är kopplade till specifika program eller system. Uppdelningen utifrån projektstegen är även relevant då varje projektsteg byggde vidare på föregående projektsteg och krävde att förgående steg var slutfört innan det var möjligt att påbörja nästa.

3.1.1 Uppstart av robot

Första steget var att starta upp roboten och lära känna utrustningen och de olika program som skulle användas. Redan under detta steg valdes metoder för genomförandet.

3.1.1.1 Konfiguration

Konfiguration av motorerna behövdes för att koppla rätt motor till rätt hjul på roboten, alltså så att robotens dator vet vilken av motorerna som hör till vänster respektive höger sida. En mjukvaruuppdatering genomfördes därför på robotens dator.

3.1.1.2 Trasigt CAN- kort

De robotar som Volvo idag använder i GPSS använder sig av CAN-kommunikation och därför var tanken att denna robot också skulle använda sig av CAN-kommunikation. Det CAN-kort som från början satt monterat på Raspberry Pi:n och var avsett för att användas i arbetet visade sig inte fungera. Detta upptäcktes vid test-sändning av CAN-meddelanden mellan Raspberry Pi:n och en annan enhet som kunde hantera CAN genomfördes. När det upptäcktes att CAN-kortet var trasigt beslutades att CAN-kommunikationen skulle bytas ut mot ROS-kommunikation i stället. Detta beslut togs för att inte förlora tid på att invänta ett nytt CAN-kort och eftersom ROS redan användes på robotens dator.

3.1.2.3 Kommunikation mellan robot och Raspberry Pi

För att programmera roboten används en Raspberry Pi som kommunicerar med robotens egen dator. Det är sedan robotens egen dator som styr motorerna. Kommunikationen mellan Raspberry Pi:n och robotens dator är ROS-kommunikation och den kommunikation som sedan används mellan Raspberry Pi:n och GPSS är Redis-baserade meddelandekommunikation.

3.1.2 Körning av robot via odometri

I detta projektsteg låg fokus på att upprätthålla kontakt med roboten via ROS samt att etablera styrning för roboten som endast berodde på robotens egna hjulodometri.

3.1.2.1 Robotens odometri

När roboten startas upp har den koordinaterna (0,0), oavsett var i rummet den startas. Detta sker eftersom odometri-koordinaterna utgår från robotens startposition i stället för ett fast koordinatsystem.

Olika faktorer påverkar odometrin, såsom hjulens diameter och avståndet mellan hjulen. Hjulen är monterade i framkant på roboten vilket innebär att de koordinater som hämtas från odometrin är baserade på punkten mitt emellan hjulen i stället för robotens centrumpunkt. Robotens dator beräknar robotens koordinater och skickar sedan den information till en Raspberry Pi via en ROS-nod. Även rotationsvinkeln skickas från

robotens odometri. Vinklarna skickas i form av kvaternioner och omvandlas sedan av Raspberry Pi:n till Euler-vinklar.

3.1.2.2 Målposition

Körbanan till målpositionen beräknas utifrån den nuvarande positionen. Beräkningarna sker kontinuerligt under körningen för att kunna läsa av ändringar av målpositionen eller ändringen av robotens position i förhållande till målpositionen för att därefter kunna korrigera körbanan. Om en ändring av målpositionen sker förändras resultaten av beräkningarna vilket innebär att robotens körväg förändras.

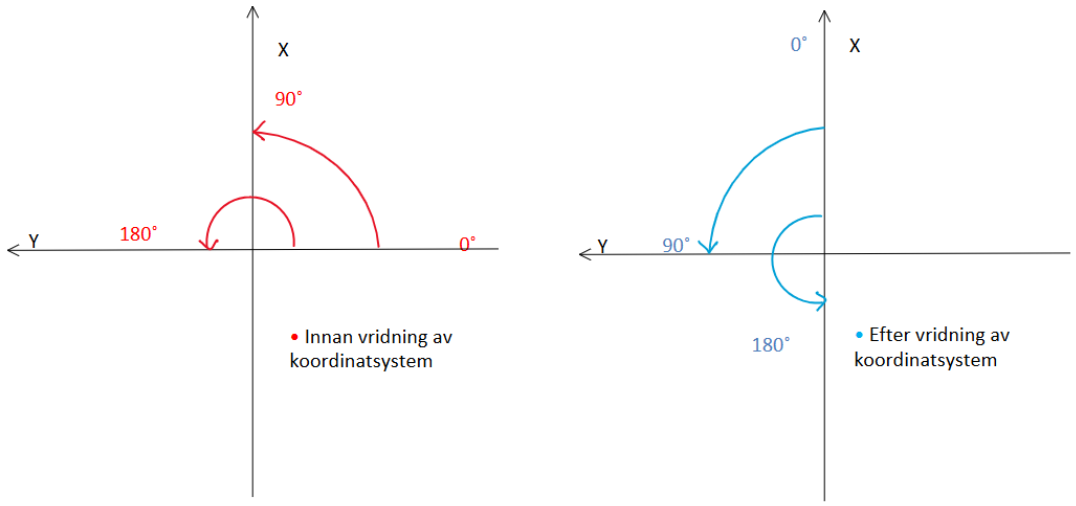
Avståndet till målpositionen samt vinkeln roboten behöver svänga för att nå målpositionen beräknas. Roboten kör den kortaste vägen till målpunkten och svänger i den riktningen som är närmast. Ekvation (4) visar hur avståndet till målpunkten räknas ut genom att använda robotens nuvarande koordinater och målkoordinaterna.

$$Avstånd\ till\ mål = \sqrt{(mål\ x - nuvarande\ x)^2 + (mål\ y - nuvarande\ y)^2} \quad (4)$$

Nuvarande koordinater och målkoordinaterna används även för att beräkna vinkeln som roboten behöver svänga för att nå målpunkten. I ekvation (5) beräknas målvinkeln för roboten.

$$Målvinkel = atan2\big((mål\ x - nuvarande\ x), (-mål\ y + nuvarande\ y)\big) \quad (5)$$

Atan2 beräknar en vinkel i förhållande till x-axeln i ett tvådimensionellt koordinatsystem, där positiva vinklar är ovanför x-axeln och negativa vinklar är under x-axeln. Då robotens x-koordinat representerar förflyttning framåt och y-koordinaten representerar förflyttning från höger till vänster har anpassningar gjorts i beräkningen för att vrida koordinaterna så att de stämmer överens med resterande koordinatsystem. Efter korrigeringen är vinklar till vänster positiva och vinklar till höger är negativa, se figur4.



Figur 4: Grafen till vänster visar hur vinklarna fungerade innan vridning av koordinatsystemet och grafen till höger visar efter vridning.

3.1.2.3 P-reglering

Styrningen för roboten p-regleras utifrån avståndet och vinkelfelet till målpunkten. Vinkelhastigheten påverkas av hur stort vinkelfel roboten har och den linjära hastigheten påverkas av avståndet till målet, desto närmare punkten roboten är desto långsammare kör den.

Robotens vinkelfel beror på den beräknade målvinkeln och robotens nuvarande rotation. Beräkningen beskrivs i ekvation (6).

$$Vinkelfel = målvinkel - nuvarande\ rotationsvinkel \qquad (6)$$

Tester genomfördes för att bestämma minimum och maximum tillåtna hastighet för roboten. Förstärkningsfaktorn beräknades därefter genom att använda minimum och maximum hastighet samt regleringsgränsen som avgör den maximala gränsen för att p-regleringen ska vara aktiv, se ekvation (7).

$$Förstärkningsfaktor = \frac{max.\ hastighet - min.\ hastighet}{regleringsgräns} = \frac{1.0-0.01}{65} = 0{,}015 \qquad (7)$$

När vinkelfelet är större än 65° svänger roboten med konstant hastighet tills vinkelfelet är mindre eller lika med 65°. Då kör roboten i stället framåt mot punkten samtidigt som

den svänger. Alltså regleras vinkelhastigheten när vinkeln är mindre eller lika med 65 grader, annars är vinkelhastigheten konstant.

Lägsta och högsta hastighet har bestämts utifrån vad som anses vara bäst lämpat. Om den högst tillåtna hastigheten är för stor påverkar det regleringen och bidrar till en mer ojämn styrning. Om den lägsta hastigheten är för låg rör sig inte roboten eftersom den påverkas av sin egen vikt.

Den linjära hastigheten bestäms i förhållande till avståndet från målpositionen, se ekvation (8).

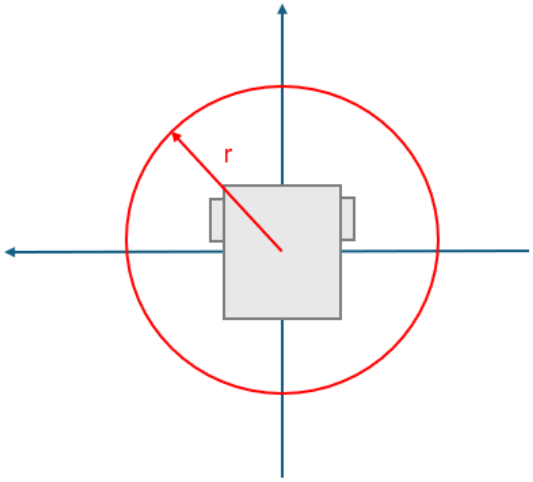
$$Linjär\ hastighet = Angiven\ hastighet \cdot Avståndet\ till\ målpositionen \qquad (8)$$

Den linjära hastigheten bestäms av avståndet till målpositionen och bör-värdet för hastigheten. Bör-värdet tas emot från GPSS. Utöver detta används en maximum- och minimum-gräns för den linjära hastigheten. Maximum gränsen är den angivna hastigheten från GPSS och minimumgränsen är 0 m/s.

P-regleringen bidrar till en effektiv styrning mot målpunkten genom att både köra framåt och svänga samtidigt. Genom att testa regleringen upprepade gånger beslutades det att förstärkningsfaktorn som beräknades med ekvation (8) skulle ändras och startvärdet för förstärkningsfaktorn blev i stället 0,0095.

3.1.2.4 Avvikelsetolerans

För att säkerställa att roboten träffar målpunkten fastställs en toleranszon. Storleken på toleranszonen bestäms av avvikelsetoleransen. Avvikelsetoleransen utgör radien från målpunkten som skapar toleranszonen, se Figur 5. Om roboten når denna zon är målpunkten uppnådd.



Figur 5: Visar toleranszonen som bestämts av toleransavvikelsen, r, i förhållande till att roboten befinner sig på den exakta målpositionen.

Avvikelseintervallet gör det enklare för roboten att nå målpunkten eftersom kravet på robotens precision minskar. Dessutom bidrar det till att styrningen av roboten blir mer jämn eftersom roboten nu inte behöver göra små justeringar för att hitta den exakta positionen. När roboten letade efter den exakta punkten innebar det att den utförde fler små svängningar som uppfattades som skakiga samt ineffektiva eftersom det tog tid för roboten att hitta den exakta positionen. Efter justeringar och tester av robotens navigeringsprecision är toleransintervallet bestämt till 15 cm.

3.1.3 Styrning av robot via odometri i samverkan med GPSS

Projektets andra fas handlade om att påbörja läsningen av protokollet från GPSS och via det styra roboten i kamerans koordinatsystem utifrån robotens odometri i kombination med positionen från kameran.

3.1.3.1 Offset för startposition

Kameran och robotens koordinatsystem har vid uppstart olika referensramar. Detta gäller både robotens x- och y-koordinater i rummet samt robotens rotationsvinkel, alltså den riktning roboten är vänd mot. Kameran har ett fast koordinatsystem i rummet

medan robotens koordinatsystem alltid definieras utifrån den position roboten startas i.

Kamerans och robotens koordinatsystem synkades genom att beräkna en offset, alltså skillnaden mellan robotens startposition enligt kameran och dess startposition enligt robotens egna odometripositioner. Denna offset för både x- och y-koordinaterna adderades sedan till i robotens position enligt odometrin, så att den kunde synkroniseras med kamerans koordinatsystem.

3.1.3.2 Filtrering av positionering

När både kamerapositionerna och odomertipositionerna användes i kombination för att styra roboten förekom det variationer i positioneringen i form av mätbrus. Bruset uppstod när man lade samman positionerna från de olika källorna då deras positioner inte överensstämde helt. Genom att använda ett Single Exponential Smoothing-filter, filtreras brus i positioneringen bort. Filtret användes genom att titta på den gamla och den nya positionen och sedan vikta vilken av dem som ska ha störst påverkan på den nya filtrerade position som sedan används genom att bestämma utjämningskonstanten.

3.1.3.3 Förflyttning av markör till robotens centrumaxel

För att kameran skulle kunna läsa av robotens position monterades en markör på robotens mittpunkt av chassit. Robotens rotationspunkt är däremot inte den samma som dess mittpunkt eftersom hjulen är monterade längst fram på roboten. Detta innebär att den axel som roboten roterar kring sitter mellan robotens hjul. Genom att inkludera en offset som representerade markörens positionering relativt robotens rotationsaxel möjliggjordes en justering av koordinaterna i både x- och y-led.

3.1.3.4 Låsning av vinkelfel

Roboten utförde stora riktningsförändringar när den befann sig nära sin målpunkt. Detta resulterade i en noggrannare positionering i förhållande till målpunkten vilket är positivt om man vill att roboten ska köra så precist som möjligt. När roboten skulle köra till många olika punkter efter varandra ledde det i stället till en ryckig körning. För att minimera detta låstes vikelfelet när roboten närmade sig målpunkten, vilket lede till en inte lika exakt positionering men mycket mjukare och mer stabil körning.

3.1.3.5 Tidsfördröjning av positionsinformation

Från att kameran läser av robotens position tills när den skickar i väg positionsinformationen hinner en liten tid gå. Detta innebär att när datorn tar emot positioneringen från kameran har roboten redan förflyttat sig från den specifika positionen. Beroende på hastighet och riktning kan positionen variera mer eller mindre.

Genom att veta hur länge sedan positionen skickades samt genom att beräkna hastigheten den kör i och därefter räkna fram ett deltavärde, både i x- och y-led, kan tidsfördröjningens resultat motverkas. Deltavärdet läggs på den positionen som senast togs emot från roboten för att få en mer exakt punkt för robotens aktuella position.

3.1.4 Implementering av roboten i GPSS-miljön

Projektsteget när roboten skulle implementeras i GPSS-miljön inledes med att via kod kunna ta emot målpositioner från GPSS. När det var implementerat skickades även robotens nuvarande position till GPSS för att på så sätt kunna tala om för systemet att roboten nått målpunkten.

3.1.4.1 Kommando

När roboten integreras i GPSS tar den emot koordinaterna till målpunkten via Redis. Ett startkommando skickas från GPSS till roboten. Startkommandot innehåller information som roboten använder för att ta sig till målpunkten. Kommandot talar om vilken sorts rörelse som roboten ska utföra. Roboten kan ta emot tre olika rörelsekommandon, *MoveTo*, *NoAction* och *Waiting*. För att styra roboten till en punkt används kommandot *MoveTo*. När roboten inte ska göra någonting används kommandot *NoAction* och om det finns ett hinder framför roboten ska kommandot *Waiting* användas. Kommandot innehåller även information om hastigheten som roboten ska köra samt en tidstämpel för när kommandot skickades. Kommandot uppdateras flera gånger under robotens körning för att justera robotens väg.

3.1.4.2 Status

Efter att roboten tagit emot ett kommando och påbörjar sin körning kommer den skicka kontinuerliga statusuppdateringar via Redis till GPSS. Informationen som skickas via statusmeddelandet innefattar robotens aktuella koordinater och vinkel, IP-adress och område för roboten samt nuvarande vinkelhastighet och linjär hastighet.

Denna information skickas kontinuerligt för att systemet ska ha en uppfattning om robotens positionering och hur den rör sig mot målpunkten. Även en tidsstämpel skickas med i statusmeddelandet för att indikera var roboten befinner sig vid specifika tillfällen.

3.1.4.3 Svar

När roboten nått målpunkten kommer roboten skicka ett svar till GPSS som indikerar att roboten nått dit den ska köra. Svaret skickas precis som kommandot och statusen via Redis. Svaret innehåller det kommando som roboten har genomfört, tidsstämpeln för när kommandot togs emot samt att den innehåller en tidsstämpel från när roboten nått sin slutgiltiga position. När roboten inte är på målpositionen, skickas tidstämpel i svaret som tom för att hänvisa att positionen inte är nådd än.

3.1.4.4 Undvika hinder

När det finns ett hinder framför roboten innebär det att den stannar tills hindret är avlägsnat. Detta sker genom att ett kommando som kallas *Waiting* skickas till roboten när kamerorna känner av ett hinder. När roboten tar emot *Waiting* kommandot sätts den linjära hastigheten och vinkelhastigheten till 0 vilket i sin tur innebär att roboten stannar.

3.2 Beskrivning av slutgiltigt systemet och styrsystemets funktionalitet

Det slutgiltiga systemet och dess funktionalitet utgörs av implementeringen av roboten i GPSS och sammanfogningen av de olika delfunktionerna till en enda helhet. När integreringen med roboten är genomförd, alltså när kommandon, status och svar skickas mellan roboten och GPSS är det via GPSS-gränssnittet som de olika punkterna anges.

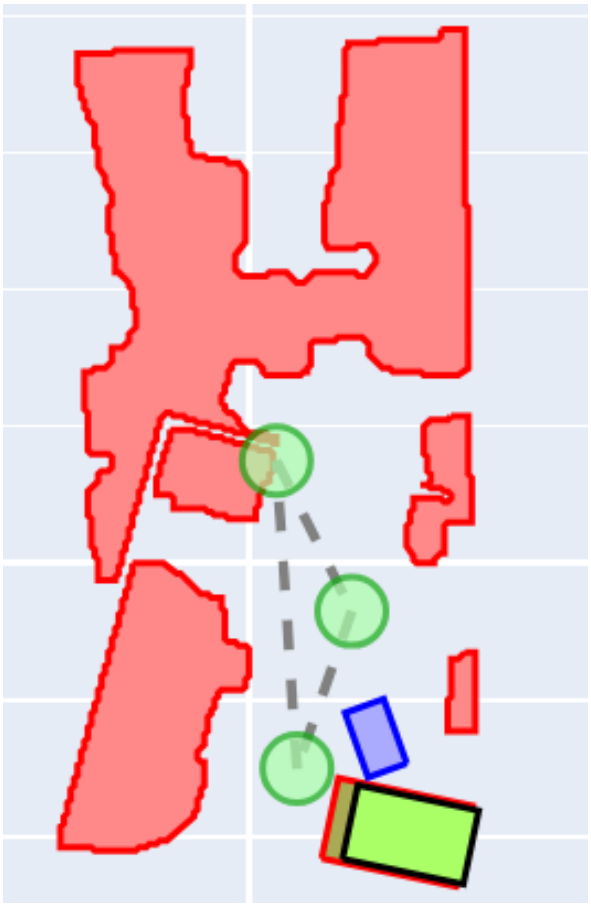
För att bestämma en bana som roboten ska köra på skapas *waypoints* som används för att navigera roboten. Flera punkter kan skapas för att utgöra körbanan, se figur 7, exempelvis en plats för upphämtning av en produkt, en plats för avlämning av en produkt och en plats för att ladda. Ordningen och positionen för punkterna bestäms i GPSS-gränssnittet, se figur 6. I gränssnittet bestäms även vilken sorts rörelse som roboten ska utföra samt med vilken hastighet och vilka koordinater den ska köra till.

WAYPOINTS TABLE OVERVIEW

Id	Area	Radius	Final angle	x	y
wp4	campx	0.50		0.70	0.00
wp3	campx	0.50		0.40	1.50

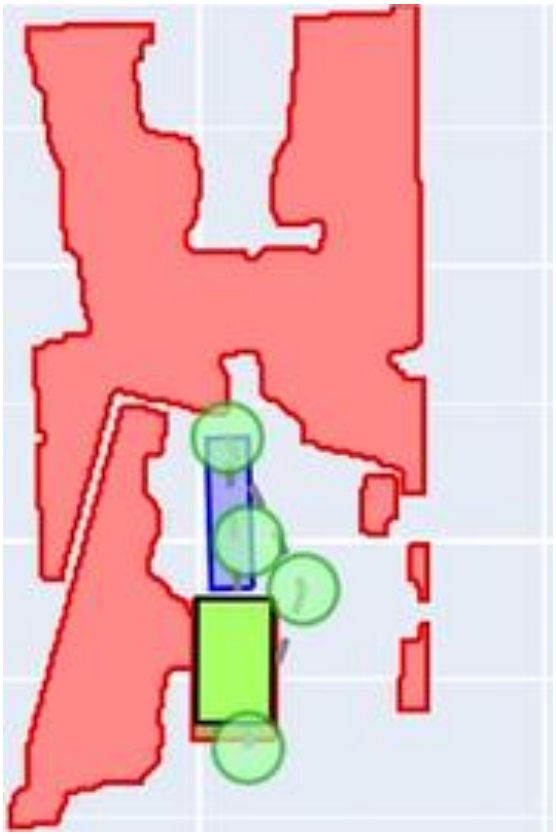
Figur 6: Tabell där position för waypoints bestäms i GPSS, Skärmavbild från GPSS

I gränssnittet kan man även se robotens rutt och aktuella körning visualiserat i en karta, se figur 7. Mellan varje *waypoint* skapas en tillfällig och rörlig *waypoint*. Roboten tar emot en temporär punkt från GPSS. Den temporära punkten är en punkt mellan roboten och målpunkten. När roboten närmar sig den temporära punkten förflyttas den. Roboten når alltså aldrig fram till punkten innan den förflyttar sig. Enda gången roboten når punkten är när den nått den slutliga målpunkten.



Figur 7: Robotens körbana mellan waypoints (de gröna punkterna). Skärmavbild från GPSS

För att undvika hinder får roboten information av kameror ifall det befinner sig något hinder framför roboten. Kamerorna i taket söker av all yta i rummet för att identifiera hinder. Alla hinder identifieras och ritas upp på en karta (rött), se figur 8. För hinderidentifiering används en ruta (blå) med start en meter framför roboten och som täcker robotens körbana, se figur 8. Användaren kan själv bestämma hur långt i körbanan hinderidentifierings-rutan ska sträcka sig och även om den enbart ska vara identifierad till nästa waypoint eller om den ska fortsätta förbi. Denna ruta symboliserar robotens närmsta körväg. Finns det inte något hinder i rutan framför roboten skickar GPSS kommandot *MoveTo* till roboten och roboten kör då framåt. Om det däremot skulle befinna sig ett föremål i rutan framför roboten betyder det att det befinner sig ett hinder i robotens körbana och då skickas i stället kommandot *Waiting* och roboten står still tills hindret har flyttat på sig. I figur 8 symboliserar den gröna fyrkanten roboten, de gröna punkterna de bestämda waypointsen och den sträckande linjen mellan waypointsen symboliserar robotens körbana.



Figur 8: Bild på blå rutan som identifierar hinder framför roboten. Skärmavbild från GPSS

3.4 Test och verifiering

Vid utveckling av robotens funktioner var testning en stor del av arbetet för att försäkra att robotens styrning och navigering fungerade som förväntat.

3.4.1 Testmetoder

Eftersom arbetet till största del har inneburit att skriva kod har små tester genomförts kontinuerligt. Vid stora och komplexa projekt som innefattar mycket programmering är det viktigt att ofta och i små etapper testa koden. Kontinuerlig testning av koden i samband med att ny kod har tillkommit är viktigt eftersom det blir mycket enklare att identifiera exakt var ett fel kan ha uppstått, felsökningen blir alltså både enklare och mer effektiv. På grund av detta är den testmetod som använts mest i projektet att kontinuerligt genomföra små tester.

I samband med alla projektstegsavslut har även ett större test genomförts för att säkerhetsställa att kraven för projektsteget har uppfyllts och att det är möjligt att gå vidare till nästa steg. Vid varje projektavslut har även resultatet visats upp och presenterats för handledaren för att säkerhetsställa att det varit godkänt.

3.4.2 Testförhållanden

Alla tester har utförts i en labbmiljö med bra förhållande. De yttre faktorerna i labbmiljön så som underlaget och ljuset har varit bra. De enda begränsningarna kopplade till labbmiljön var att det enbart fanns 2 GPSS-kameror och ytan i labbmiljön var begränsad. Nackdelen detta medförde var att roboten ej kunde testas på att köra långa avstånd.

3.4.3 Verifiering av test

Många tester som utfördes gjordes ett upprepat antal gånger för att säkerhetsställa att resultatet varit det samma varje gång och aldrig berott på slumpen. Tester utfördes även med varierade förutsättningar, till exempel när roboten skulle navigera till bestämda punkter startades roboten alltid med varierade startvinklar och positionering. Olika målpositioner angavs också för att konstatera att roboten alltid lyckades navigera rätt oavsett vart den skulle köra.

Tester som genomfördes för att justera värden för en förbättrad körning loggades och data antecknades för att kunna analysera resultaten och via det bestämma det mest optimala värdena.

4. Resultat

I följande kapitlet presenteras arbetets resultat med fokus på navigationsprecision, rörelse och styrning. Det presenteras hur parametrar har bestämts utifrån olika tester och hur det har påverkat det färdiga systemet och dess beteende.

4.1 Navigationsprecision

Flera olika metoder användes för att utveckla och förbättra robotens navigering. Nedan beskrivs hur metoderna påverkat roboten och dess funktionalitet. Dessutom beskrivs hur metoderna har använts och vilka värden, begränsningar och prioriteringar som metoderna medfört.

4.1.1 Avvikelsebegränsningar

För att bestämma den nödvändiga toleransen som avgör hur nära målpunkten roboten måste komma, genomfördes tester och olika faktorer vägdes mot varandra. En liten tolerans innebär att roboten hade kommit nära punkten samtidigt som det innebär att robotens navigering måste vara mer exakt för att träffa rätt punkt. En hög tolerans innebär att roboten enkelt navigerade och kvitterade sin målpunkt. Dock innebär det även att felmarginalen för den faktiska målpunkten och där roboten stannar är stor.

Att sänka toleransen prioriterades. Genom att sänka toleransen kör roboten nu med en stor säkerhet till målpunkten. När toleransen sänktes för mycket blev det däremot svårare att träffa punkten vid vissa sträckor, exempelvis korta sträckor med stora vinklar. För att försäkra att roboten inte missar punkten valdes ett toleransvärde som var litet nog för att med säkerhet vara nära punkten men stor nog för att undvika onödiga fel. Toleransen som används är 15cm. När toleransen var 5 cm uppstod felet med att roboten missade punkten frekvent och när toleransen var 10 cm missade den ibland punkten.

4.1.2 Filterspecifikationer

När roboten skulle styras via odometripositionen i kombination med kamerapositionen implementerades ett filter. Filtret användes för att minska stora skillnader mellan positionerna från hjulodometrin och från kameran. I filtret behövdes en utjämningsfaktor bestämmas som skulle avgöra hur mycket man skulle lita på respektive position när den nya filtrerade positionen skulle beräknas. Detta valdes efter en del beräkningar men framför allt genom att testa olika värden tills robotens körning var optimal. Den utjämningsfaktor som valdes var 35%. Detta innebär att den nya gemensamma positionen skulle bestå till 35% av kamerapositionerna och 65% av odometri positionerna. Detta valdes eftersom odomertipositionerna är stabilare och uppdateras mer ofta. Kamerapositionen är däremot väldigt exakta men uppdateras mer sällan och kan därmed leda till hackigare körning. Om roboten styrs till 100% via kamerapositionen upplevdes rörelsen hackig då koordinaterna sällan uppdateras, detta eftersom roboten vid varje ny positionering försökte kompensera för ett felaktigt vinkelfel. Resultatet blev då en svängande körning kring sin tänkta bana. Om roboten däremot enbart körde via odometrin körde roboten mer följsamt men den tenderade i stället att drifta i koordinatsystemet. Viktfaktorn på 35% ledde till att den nya viktade positionen gav en stabilare positionering då två positioneringar från olika källor ger bättre noggrannhet än en ensam.

4.1.3 Begränsning av vinkelfel

För att precisera och effektivisera navigeringen låstes vinkelfelet när roboten var tillräckligt nära målpositionen. Om roboten är 35 centimeter från målpositionen sätts vinkelfelet till 0 vilket innebär att roboten kommer fortsätta köra i aktuella riktningen.

Genom testning av olika värden för avståndet bestämdes det att 35 centimeter var mest lämpad och gav önskat resultat. När avståndet var för stort innebar det att roboten oftare missade punkten eftersom det faktiska vinkelfelet var stort nog att påverka körningen av roboten. När avståndet var för litet var den sista biten av körningen väldigt ryckig då roboten hela tiden försökte kompensera och reglera för vinkelfelet.

Det finns flera faktorer som påverkar vid vilket avstånd som vinkelfelet bör låsas, ett exempel är radien hos toleransavvikelsen, alltså hur nära punkten roboten måste träffa. Ytterligare en faktor som påverkar avståndet är robotens hastighet, både den

linjära hastigheten och vinkelhastigheten påverkar styrningen av roboten. Dessutom påverkar förstärkningsfaktorn som sköter p-regleringen även avståndet. Låsningen av vinkelfelet prioriterades lägre än de ovanstående faktorerna vilket innebär att det var viktigare att dom andra faktorerna var bra inställda. Därefter bestämdes ett passande avstånd för att låsa vinkelfelet vilket.

4.2 Rörelse och styrning

För att optimera robotens rörelse och styrning applicerades olika metoder. Nedan listas de metoder som krävde olika variabler för att kunna tillämpas, samt hur dessa olika variabler fastställdes.

4.2.1 Hastighetsavgränsningar

I GPSS kan användaren bestämma i vilken linjär hastighet roboten ska köra. Både den linjära hastigheten och vinkelhastigheten beräknades sedan i koden med hjälp av hastigheten angiven i GPSS för den specifika sträckan. Men detta kunde bidra till sämre styrning när hastigheterna var antingen för höga eller för låga. Därför implementerades begränsningar på dessa hastigheter.

4.2.1.1 Linjärhastighet

Maxvärdet för den linjära körningen sattes till samma hastighet som den förbestämda hastigheten i GPSS. Denna hastighet förhindrades alltså inte och detta betyder att man litar på att användaren inte skriver in en alldeles för hög hastighet. Minimumhastigheten för den linjära hastigheten sattes till 0 m/s. Detta innebär alltså att ingen nedre begränsning har införts men det gör det möjligt för roboten att köra väldigt långsamt som den ska göra när den närmar sig målet.

Det var inte bara min- och maxvärden som implementerades för att kontrollera robotens linjära hastighet. En annan metod som implementerades för att korrigera robotens hastighet var att den saktade ner när den närmade sig sin målposition. Detta gjordes genom att multiplicera den angivna hastigheten från GPSS med avståndet till målet. Detta innebar att när roboten var mindre än en meter från målet saktade den in för att kunna anpassa farten och lättare träffa målet. Men roboten körde aldrig snabbare än den angivna hastigheten eftersom maxvärdet tidigare begränsades till den fart som är bestämd i GPSS.

4.2.1.2 Vinkelhastighet

För att sätta en maximum- och minimumgräns för vinkelhastigheten gjordes tester för att avgöra inom vilket hastighetsspann roboten kunde utföra körningen på ett kontrollerat och effektivt sätt. Om den maximala hastigheten var för hög bidrog det till att robotens körstil blev mer osäker, robotens svängar blev hastiga. Ännu en bidragande faktor till beslutet om den maximala vinkelhastigheten var att om roboten i framtiden har en last som eventuellt inte är fastspänd på roboten ska det inte finnas utrymme för godset att glida av roboten.

Anledningen till att den minimala hastigheten inte är noll är på grund av p-regleringen. I vissa fall bidrar p-regleringen till att vinkelhastigheten blir väldigt liten. I dessa fall innebär det att roboten inte svängde för att hastigheten var för liten i förhållande till robotens egen vikt. Efter tester bestämdes det att den maximala hastigheten var 1.0 rad/s medan den minimala hastigheten var 0.01 rad/s.

5. Diskussion och slutsats

För att sammanfatta och utvärdera implementeringen av roboten i det befintliga GPSS diskuteras olika perspektiv utifrån robotens resultat. Systemets begränsningar och tillförlitlighet, måluppfyllelse samt eventuella framtida förbättringar nämns för att ge ökad insikt och förståelse i roboten. Dessutom diskuteras roboten utifrån olika etiska och hållbarhetsperspektiv.

5.1 Systemets begränsningar och tillförlitlighet

Nedan beskrivs de begräsningar som påverka robotens funktionalitet och tillförlitlighet i verkliga miljöer.

5.1.1 Förändringar i omgivningen

Roboten har god förmåga att undvika hinder tack vare hinderidentifieringen från kamerorna i taket. Systemet upptäcker i god tid rörliga hinder och roboten stannar med god marginal. För att minimera risken för att roboten skulle råka krocka med något objekt har man gjort området som upptäcker hinder relativt stort. Detta kan dock vara en begränsning. Om roboten rör sig på trånga utrymmen i fabriken och en pall eller något annat objekt placeras några centimeter fel skulle det kunna innebära att roboten inte kör vidare då dess väg är blockerad.

En annan begränsning vid hinderidentifiering är att roboten stannar och väntar tills planerade vägen är fri även om den har stor yta för att köra om vid sidan av hindret. Systemet kan alltså inte räkna ut en ny rutt trots att ytan för det finns. Roboten kör endast den banan som användaren bestämt.

Ojämna eller hala golvytor är också en begränsning som kan påverka rörelseprecision och bidra till bland annat slirning. Detta kan självklart ha negativ påverkan på navigationen då positioneringen från hjulodometri kan bli missvisande och felaktig. Men eftersom positioneringen redan bestäms utifrån både hjulodometri och kamerapositionen tas detta problem redan hänsyn till.

5.1.2 Labbmiljö

Roboten testkördes endast i labbmiljö vilket i sin tur innebar begränsade förhållanden i form av yta och hinder. Andra fel som ännu inte kunnat undersökas på grund utav labbmiljön kan komma att upptäckas vid tester om roboten skulle integreras i det verkliga systemet. Något som ej undersökts ännu är hur integrationen av mer än en robot skulle påverka systemet. En annan funktion som inte testats till följd av labbmiljön är robotens funktion vid hämtning och avlämning. Som en följd av den begränsade ytan har inga tester genomförts där roboten navigerar stora ytor och långa sträckor. Dessutom finns fler källor till oväntade hinder i den verkliga miljön jämfört med labbmiljön vilket innebär att testerna för hinder inte är på samma nivå som den verkliga miljön.

5.1.3 Kombination av odometri och kamera

Anledningen till att både hjulodometri och kameror används är för att de kompletterar varandras egenskaper på ett gynnsamt sätt. Som nämnt ovan skiftar hjulodomterins koordinater vid slirningar. Längre körsträckor kan även medföra skiftningar i koordinatsystemet. För att motverka detta används kameror som rättar till koordinaterna från hjulodometrin. Anledningen till att inte endast kamerorna används är för att de inte skickar robotens koordinater i realtid och inte heller skickar konstanta uppdateringar av koordinaterna. Under tiden som kamerorna inte skickar nya koordinater används hjulodometrin. Detta är anledningen till att både odometrin och kamerorna är viktiga och bidrar till robotens navigering.

5.2 Måluppfyllelse och analys ut av frågeställningen

Ett mål var att robotenen skulle kunna integreras i Volvos GPSS. Detta mål är uppfyllt då roboten kan ta emot kommandona *MoveTo, Waiting* och *NoAction* från GPSS. Roboten kan även skicka meddelande till GPSS. Den skickar kontinuerligt sin status alltså sin nuvarande position och den skickar även ett svarsmeddelande när den nått en målposition och därmed är klar med ett kommando. Detta meddelande innehåller dess position och tidstämpel samt det kommando roboten har avklarat.

Ett annat mål var att roboten skulle kunna styras både manuellt och automatiskt. Detta är uppnått då det både är möjligt att köra roboten via dess joystick och via kommandon i GPSS. Joysticken har högre prioritet än kommandona i GPSS viket gör att man när som helst kan ta över styrningen och köra roboten manuellt. När roboten är i sitt automatiska läge är den förprogrammerad att köra en specifik bana. Banan är uppbyggd av w*aypoints* som roboten kör till i den ordningen som är bestämt av användaren. Om ett rörligt hinder så som en människa eller en truck blockerar robotens väg mellan två *waypoints* får den ett kommando *Waiting* vid detta kommando stannar roboten och väntar på att vägen blir fri igen och fortsätter då att köra. Detta betyder att även målet om att stanna vid hinder är uppnått.

Målet om att roboten ska använda en kombination av både hjulodometrin och positioner från kameror monterade i taket uppnåddes. När en kamera skickar en position till roboten används den för att ange robotens aktuella position. Kamerorna uppdateras inte konstant utan har en liten fördröjning vilket innebär att mellan att roboten tar emot en position från kameran tills att den skickar en ny uppdateras inte positionerna från kameran även om roboten rör på sig. Då används i stället positionen från robotens odometri medan roboten väntar på en uppdaterad position från kameran. En annan anledning till att använda både kameror och odometri är för att odometrin på egen hand kan vara instabil och osäker på grund av exempelvis slirningar men även för att odometrin inte har ett fast koordinatsystem. Odometrin utgår ifrån robotens startpunkt medan kamerorna har ett fast koordinatsystem. Roboten navigerar därför via kamerornas koordinatsystem.

Kommunikationen mellan roboten och GPSS sker utifrån det förbestämda protokollet. Den data som roboten tar emot och skickar är förbestämt. Roboten tar emot ett kommando och skickar sedan kontinuerligt sin status till GPSS och när målpunkten är uppnådd skickas sedan ett svar.

CAN-kommunikation skulle använts från början men eftersom CAN-kortet var trasigt beslutades det därför att användas ROS-kommunikation. Målen korrigerades alltså under arbetets gång och ROS-kommunikation ersatte CAN-kommunikation. Bytet från CAN till ROS hade inte någon större påverkan på resultatet men genomförandet förändrades. Fördelen med ROS jämfört med CAN var att det var ROS kommunikationen som användes på roboten från början vilket gjorde det enkelt att implementera och det var även relativt enkelt att använda.

Roboten undviker hinder med stor säkerhet. I GPSS är det förbestämt hur stor ytan som upptäcker hinder skall vara samt var den ska vara placerad i förhållande till roboten.

5.3 Framtida förbättringar och möjligheter

En förbättringsmöjlighet är att roboten haft en egen sensor för hinderidentifiering. Detta var planen med arbetet från början men i och med att scannern som tillhörde robotkittet kopplades ur fanns inte tiden att koppla in den igen. Det gjordes ett försök att återigen koppla in den med det lyckades inte och tiden att felsöka för att få det att fungera fanns inte och därmed beslutades att scannern inte kulle användas. Om roboten haft en sensor monterad skulle hinderigenkänningen bli ännu bättre då den snabbare hade kunnat identifiera plötsliga hinder utan tidsfördröjning och ytterligare en nivå av säkerhet skulle förhindra krock med hinder. En förbättringsmöjlighet är därför att återuppta användandet av säkerhetsscannern.

Ytligare en förbättringsmöjlighet för arbetet hade varit ett starkare batteri vilket skulle innebära längre drifttid om roboten i framtiden ska implementeras i fabrik. Eventuellt kan detta medföra att förändringar behöver göras i robotens kod gällande exempelvis den linjära hastigheteten och vinkelhastigheten eftersom robotens vikt förändras vilket i sin tur leder till att det kan kräva större hastigheter på motorerna för att driva roboten.

Dessutom skulle ett tyngre batteri eventuellt innebära att tyngdpunkten för roboten förändras vilket kan påverka robotens körning.

Slutligen hade ytterligare en förbättring varit att implementera en funktion för att rotera på plats. Kommandot som GPSS skickar till roboten kan bestå av olika rörelser, i detta fall är *NoAction*, *MoveTo* och *Waiting* implementerade. Däremot finns fler kommandon i GPSS, varav ett av dom är *Rotate* vilket säger till roboten att rotera ett visst antal grader på stället. Detta är bra att ha ifall roboten förväntas köra på trånga platser eller utrymmen. Dock förflyttas roboten när den roterar, då den inte kan rotera på plats. Detta eftersom robotens hjul inte sitter i mitten av roboten vilket i sin tur innebär att roboten snurrar runt axeln mellan motorerna i stället för att snurra runt sin egen axel.

Som tidigare nämnt har roboten inte implementerats eller testats i kritisk miljö än. Men detta är möjligt då alla nödvändiga funktioner som behövs finns hos roboten. Om man i framtiden implementerar roboten i fabriken eller i annan applikation kommer det att kunna leda till en ökad användning av förarlösa transportrobotar i industrier. Förarlösa transportrobotar befinner sig i städning utveckling och har stor positiv påverkan på att effektivisera transporten i fabriker.

5.4 Hållbarhetsaspekter med etik och miljö

Ur ett hållbarhetsperspektiv bidrar implementeringen av roboten i en fabrik till minskad resursförbrukning samt effektivare produktion. Genom att använda robotar skulle en mer jämn prestanda bli resultatet. Det hade varit färre avbrott i produktionen vilket i sin tur leder till mindre resursanvändning i form att energiåtgång samt drifttid. Samtidigt skulle det innebära att färre produkter kasseras eftersom roboten inte har en mänsklig faktor som spelar in i resultatet. Detta leder även till en effektivare produktion samt en bättre kontrollerad resursförbrukning. Om roboten underhålls och sköts förlängs dess livslängd i jämförelse med om den ej underhålls. Ur ett hållbarhetsperspektiv är det därför viktigt att underhålla robotarna för att slippa ersätta dom ofta.

Ur ett etiskt perspektiv skulle roboten bidra till att människor behöver utgöra färre tunga lyft och förflyttningar. Det skulle underlätta slitsamt och monotont arbete. Detta i sin tur leder till en bättre arbetsmiljö för personalen. Däremot kan en överflödig automatisering av en industri leda till nerskärningar av personalen eller att arbetsroller byts ut. Detta innebär att nya roller skapas och personal kan behövas bytas ut.

Referenser

[1] *Automated transporters introduced at Volvo Group powered by AI and computer vision.* [Film]. Sverige: Volvo Group, 2024.

[2] Volvo grups AB, ”AI transporters push the boundaries of modern manufacturing,” 28 11 2024. [Online]. Available: https://www.volvogroup.com/en/news-and-media/news/2024/nov/ai-modern-manufacturing.html?utm_source. [Använd 04 05 2025].

[3] Open Source Robotics Foundation. (n.d.), ”ROS - Robot Operating System,” [Online]. Available: Robot Operating System. [Använd 02 04 2025].

[4] ”Controller Area Network (CAN) Protocol Overview,” 30 05 2025. [Online]. Available: https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/controller-area-network--can--overview.html?srsltid=AfmBOoo4cCQYTMAvyKJyxVH7D2AKjBSKX2VYOV1FSFilf3487QWrGMb_#section--640856706. [Använd 04 06 2025].

[5] Wikipedia. (n.d.), ”Raspberry Pi,” [Online]. Available: https://sv.wikipedia.org/wiki/Raspberry_Pi. [Använd 02 04 2025].

[6] EZ-Wheel. (n.d.)., ”EZ-Wheel – Electric wheel solutions,” [Online]. Available: https://www.ez-wheel.com/en. [Använd 02 04 2025].

[7] B. Thomas, i *Modern Reglerteknik*, Liber, 2021, pp. 54-58.

[8] S. A. Mohamed, H. Tenhunen, J. Plosila, T. Westerlund, M.-H. Haghbayan och J. Heikonen, ”A Survey on Odometry for Autonomous,” *IEEE Access,* vol. 7, pp. 97466-97486, 2019.

[9] *Modern Robotics, Chapter 13.4: Odometry.* [Film]. Northwestern Robotics, 2017.

[10] Abed, ”ROS Notes,” [Online]. Available: https://abedgnu.github.io/Notes-ROS/chapters/ROS/00_getting_started/overview.html. [Använd april 2025].

[11] ProgrammeraPython.se, ”JSON för nybörjare: En komplett guide till att hantera JSON i Python,” [Online]. Available: https://www.programmerapython.se/json-for-nyborjare-en-komplett-guide-till-att-hantera-json-i-python/#google_vignette. [Använd april 2025].

[12] CODEMA, ”Vad är Redis,” [Online]. Available: https://codema.io/sv/tekniker/redis. [Använd april 2025].

[13] G. M. Smith, ”What is Sensor Fusion?,” 15 05 2024. [Online]. Available: https://dewesoft.com/blog/what-is-sensor-fusion. [Använd 28 05 2025].

[14] M. H. P. Swari, I. P. S. Handika och I. K. S. Satwika, ”Comparison of Simple Moving Average, Single and Modified Single Exponential Smoothing,” i *2021 IEEE 7th Information Technology International Seminar (ITIS)*, Surabaya, Indonesien, 2021.

[15] ”Quaternion,” Wikipedia, 02 maj 2025. [Online]. Available: https://en.wikipedia.org/wiki/Quaternion.

[16] ”Euler angles,” Wikipedia, 15 mars 2025. [Online]. Available: https://en.wikipedia.org/wiki/Euler_angles.

[17] A. Deep, M. Mittal och v. Mittal, ”Application of Kalman Filter in GPS Position Estimation,” i *IEEE 8th Power India International Conference (PIICON)*, New Delhi, Indien, 2018.

[18] R. Ranjan, Y. Jung, D. Shin, S. Kim, C.-H. Kim, S. Lee och J. Kye, ”Improving Indoor Positioning Systems with UWB and Filtering Techniques: A Comparative Analysis,” i *The 23rd International Conference on Control, Automation and Systems (ICCAS 2023)*, Yeosu, Korea, 2023.

[19] T. Yuldashev och S. Andrey, ”Basics of PID Controllers: Design, Applications, Advantages & Disadvantages,” Integra Sources, 26 april 2023. [Online]. Available: https://www.integrasources.com/blog/basics-of-pid-controllers-design-applications/.

[20] J. Singh, *Circular Motion (Uniform and Non-Uniform),* 2022.