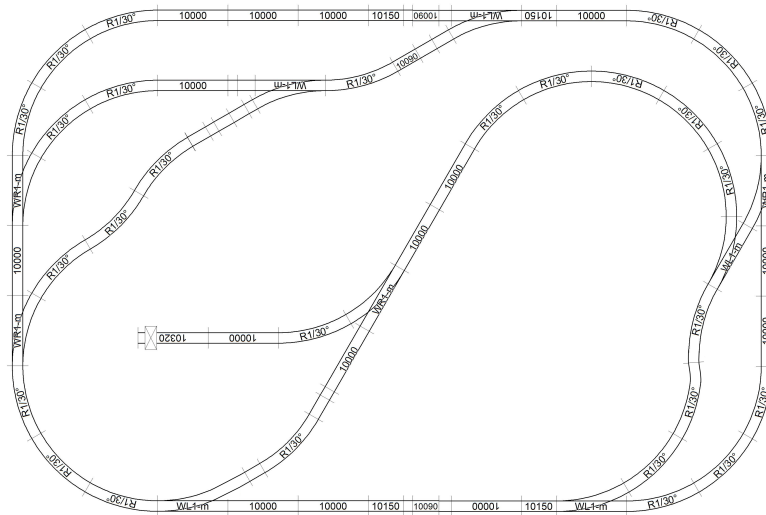




Realtidsnavigering av autonom inspektionsrobot i komplexa rälsnätverk

En småskalig implementation

Kandidatarbete inom Maskinteknik, Teknisk fysik och Automation och Mekatronik.

Alva Andreasson
Jacob Engström
Gustav Eweman
Alexander Nilsson
Emil Sitell
Isac Åkerlund

INSTITUTIONEN FÖR MEKANIK OCH MARITIMA VETENSKAPER

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

KANDIDATARBETE I MASKINTEKNIK, TEKNISK FYSIK OCH
AUTOMATION OCH MEKATRONIK.

Realtidsnavigering av autonom inspektionsrobot i komplexa järnvägsnät

En småskalig implementation

Alva Andreasson
Jacob Engström
Gustav Eweman
Alexander Nilsson
Emil Sitell
Isac Åkerlund



CHALMERS

Institutionen för Mekanik och Maritima Vetenskaper
Avdelningen för fordonsteknisk och autonoma system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025

Realtidsnavigering av autonom inspektionsrobot i komplexa järnvägsnät
En småskalig implementation

Alva Andreasson

Jacob Engström

Gustav Eweman

Alexander Nilsson

Emil Sitell

Isac Åkerlund

© Alva Andreasson, Jacob Engström, Gustav Eweman,
Alexander Nilsson, Emil Sitell, Isac Åkerlund, 2025.

Handledare: Vivien Lacorre

Examinator: Krister Wolff

Kandidatarbete 2025

Institutionen för Mekanik och Maritima Vetenskaper

Chalmers Tekniska Högskola

SE-412 96 Göteborg

Telefon +46 31 772 1000

Omslagsbild: Digital representation av layouten för det utvecklade rälssystemet.

Typsatt i L^AT_EX
Göteborg 2025

Realtidsnavigering av autonom inspektionsrobot i komplexa järnvägsnät
En småskalig implementation
Alva Andreasson, Jacob Engström, Gustav Eweman,
Alexander Nilsson, Emil Sitell, Isac Åkerlund
Institutionen för Mekanik och Maritima Vetenskaper
Avdelningen för fordonsteknisk och autonoma system
Chalmers Tekniska Högskola

Sammanfattning

Projektet har utvecklat den befintliga testbädden med tillhörande småskalig robot som finns på Chalmers som en del av projektet FP3 Iam4rail där Chalmers är en del av bidragsgrupp WP18. Det befintliga rälsnätverket utvidgades med mer räls samt växlar utrustade med servos för automatisk växling. Projektet integrerade även ett befintligt kamerasystem från Qualisys i testbädden för att använda som ett lokalt positioneringssystem av en robot som rör sig längs rälsen. Den befintliga koden för att styra den småskaliga roboten kompletterades med mjukvara för att kunna planera en rutt och navigera i rälsnätverket. Ruttplaneringen bygger på A* som kompletterats för att kunna hantera att delar av rälsnätverket kan vara otillgängliga under vissa tider. Detta för att simulera tågtrafik. Styrningen av roboten och kommunikationen mellan delsystem sker med ROS samt MQTT. Systemet i sin helhet fungerar. I försök förekom det att växlar ej klarar av att utföra en växling och fastnar i ett felaktigt läge men detta till trots kan roboten efter ett avslutat projekt självständigt planera och utföra rutter med hög tillförlitlighet i rälsnätverket.

Nyckelord: Automation, Navigering, Lokalisering, LPS, Robotik, Järnväg, Ruttplanering, ROS, Testbädd, Optimering.

Förord

Denna rapport presenterar resultatet av vårt examensarbete som genomfördes vid institutionen för mekanik och maritima vetenskaper vid Chalmers tekniska högskola under våren 2025 på avdelningen för fordonsteknik och autonoma system med Krister Wolff som examinator och Vivien Lacorre som handledare.

Författarnas tack

Vi vill rikta ett stort tack till vår handledare Vivien Lacorre för hans ovärderliga vägledning, engagemang och stöd genom hela projektets gång. Hans expertis inom ämnet gjorde arbetet möjligt att genomföra. Vi vill även tacka vår examinator Krister Wolff för hans kunskaper och vägledningen genom projektets gång som hjälpt oss slutföra detta kandidatarbete. Ett särskilt tack riktas till institutionen för mekanik och maritima vetenskaper vid Chalmers tekniska högskola som tillhandahållit oss resurser och en arbetsmiljö som möjliggjort detta arbete. Till sist vill vi tacka våra familjer och vänner som varit ett ständigt stöd under arbetets gång.

Alva Andreasson, Jacob Engström, Gustav Eweman, Alexander Nilsson, Emil Sitell, Isac Åkerlund, Göteborg, maj, 2025

Akronymer

Nedan är listan över akronymer som har använts genom hela denna rapport listade i alfabetisk ordning:

ATC	Automatic Train Control
CBTC	Communications-Based Train Control
ERTMS	European Rail Traffic Management System
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input Output
LPS	Lokalt Positionerings System
MPIR	Multi Purpose Inspection Robot
MQTT	Message Queuing Telemetry Transport
QTM	Qualisys Track Manager
ROS	Robotic Operating System
SDK	Source Development Kit

Figurer

1.1	Bild på det ursprungliga rälsnätverket, bestående av ett stickspår som syns i bildens nedre del samt en kort sträcka med parallella spår som syns i bildens övre del.	3
2.1	I a syns den Raspberry Pi som hanterar logiken och kommunikationen för växlarna. En av de mikrokontrollers som användes för att styra respektive växel syns i b	7
2.2	En av de kameror som användes i projektet syns i figur a , upphängningen gör att kameran kan justeras i både rotation och infallsvinkel. I figur b syns de markörer som QTM använder för att positionera en kropp. . . .	9
3.1	Översiktlig systembeskrivning med de ingående styrenheterna. Robotens navigeringsnod hämtar sin information från LPS och använder denna för att ta beslut som den sedan skickar till rälsnätverket och robotens huvudnod. All kommunikation mellan dessa olika system sker över ROS medan kommunikationen inom rälssystemet sker via MQTT.	11
3.2	I figuren presenteras det utbyggda rälsnätverket. I den digitala representationen är även de olika primärsegment identifierade och numrerade. . .	12
3.3	Exempel på en motväxel och en medväxling, typen av växel bestäms av riktningen som växeln passeras.	13
3.4	Den vänstra växeln är den ursprungliga och har en utstickande arm för manuell styrning. Den högra är den utvecklade designen som bibehåller den tekniska lösningen utan den manuella styrningen men med monteringsplats för en servomotor.	13
3.5	Uppdaterad växeln med och utan lock.	14
3.6	Tvärsnitt på växel.	15
3.7	Kopplingsschema för ett trafikljus.	16
3.8	Strömmen för trafikljusen vid olika GPIO signaler.	16
3.9	Skiss över kamerauppställningen i labbet. Totalt användes sex kameror i projektet.	17
3.10	Placeringen av de reflektiva markörerna på roboten. Notera hur markörerna sitter asymmetriskt kring robotens centrum för att säkerställa att avstånden mellan varje par av markörer är unikt.	18
3.11	Algoritmen hittar pågående bokningar i tidtabellen vid $t = t_3$ och lägger till ytterligare evaluering av samma segment vid $t = t_5$. En ytterligare evaluering kommer läggas till i $t = t_8$, eftersom t_8 är nästa tidpunkt då ett angränsande segment blir ledigt efter t_5	21

3.12	Flödesschema för robotens navigeringsnod som sköter kommunikation kring växling och motorstyrning.	23
3.13	När roboten färdas över växeln mellan segment 1 och segment 2 kommer segment 3 vara det närmsta segmentet under delar av övergången. Genom att roboten utgår från att den färdas längs sin rutt kan den undvika att felaktigt identifiera vart den är på väg.	24
4.1	Bild på den slutgiltiga testbädden.	27
4.2	I figuren är områden där QTM har försämrat prestanda markerade. I de markerade områden är täckningen en markör 60 %, vilket är lägre än snittet för hela rälsnätverket.	29
4.3	Roboten ska färdas från segment 1 till segment 2 över växeln i a . På grund av att växeln ställs fel – se b – navigerar roboten in på segment 3, då den upprepade gånger befinner sig på segment 3 identifierar den att det rör sig om en felnavigering. Den avbryter sin tidigare rutt och planerar en ny mellan segment 3 och segment 2. Växlarna ställs om efter den nya rutten i c respektive d och roboten ankommer på segment 2 i e	30

Innehåll

Akronymer	ix
Figurer	xi
1 Inledning	1
1.1 Bakgrund	1
1.1.1 Positionering och ruttplanering inom järnvägssektorn	1
1.1.2 Skalmmodell för autonom felsökning	2
1.1.3 Utgångspunkt för projektet	3
1.1.4 Samhälliga aspekter	3
1.2 Syfte	4
1.3 Avgränsningar	4
2 Teori	7
2.1 Styrning av delsystem	7
2.2 Algoritm för ruttoptimering med reserverade rälssegment	8
2.3 Positionering i testbädden	9
3 Design och implementation	11
3.1 Utbyggt rälsystem	12
3.2 Smarta växlar	12
3.2.1 Mekanisk modifiering av växlar	13
3.2.2 Elektronik och styrsystem	15
3.2.3 Trafikljus och styrlogik	15
3.3 Integrering av ett positioneringssystem	16
3.4 Digital representation av rälsen	18
3.5 Implementation av tidsberoende ruttplanering	20
3.6 Beslutsfattning hos roboten	22
4 Resultat	27
4.1 Rälsnätverket	27
4.2 Positioneringssystemet	28
4.3 Realtidsnavigering i rälsnätverket	29
5 Diskussion	31
5.1 Utvecklingen av testbädden och ruttplanering	31
5.2 Framtida utvecklingsmöjligheter inom projektet	32
6 Slutsatser	35

Litteraturförteckning

37

1

Inledning

Tågtrafiken är en av grundpelarna i transportsektorn för såväl pendlare, resenärer och godstransporter. För att den ska fungera tillförlitligt behöver rälsen underhållas och som ett hjälpmedel för detta inleddes ett projekt vid Chalmers tekniska högskola under år 2020. Projektet var en del det större projektet FP3-IAM4RAIL där Chalmers bidrar till underprojektet WP18 [1]. Projektet syftar till att utveckla en autonom robot som ska kunna genomföra inspektion av rälsen och benämns Multi Purpose Inspection Robot (MPIR) [2]. År 2022 testkördes MPIR på en testbana utanför Västerås men på grund av det geografiska avståndet mellan Västerås och Chalmers har man även byggt en skalmodell av MPIR, i rapporten kallad *roboten*, med ett tillhörande rälsnätverk. Detta för att kunna utföra tester av mjukvara utan att resa till testbanan i Västerås.

Målsättningen med detta projekt var att bygga ut och utveckla den småskaliga testbädd som finns i Chalmers lokaler. Rälsnätverket skulle bli mer komplext, växlarna styrs digitalt och roboten kommunicera med ett centralt trafikledningsprogram. Ett lokalt positioneringssystem (LPS) har implementeras för att kunna agera som motsvarighet till Global Navigation Satellite System (GNSS) som används på MPIR. Roboten skulle även utrustas med mjukvara för att kunna planera en rutt genom ett rälsnätverk och kunna ta hänsyn till att olika rälssegment kan vara reserverade av andra fordon under vissa tidsintervall. Roboten ska sedan kunna navigera autonomt genom rälsnätverket på ett tillförlitligt sätt.

1.1 Bakgrund

I denna del presenteras den relevanta bakgrunden för projektet. Fokus ligger på hur det svenska rälsnätverket fungerar i dagsläget samt vilka konsekvenser en ökad automation har för järnvägssektorn. Avsnittet motiverar även behovet av en småskalig testbädd samt redogör stadiet på den befintliga testbädden som var projektets utgångspunkt.

1.1.1 Positionering och ruttplanering inom järnvägssektorn

Inom järnvägssektorn är det övergripande problemet att etablera och upprätthålla en hinderfri tågväg för olika aktörer att färdas på mellan två förutbestämda punkter. För att kunna bestämma positionen för de olika aktörerna på rälsen används flera olika metoder. I Sverige är den vanligaste metoden spårledning, men även andra metoder som axelräknare, radioblockering och Communications-Based Train Control (CBTC) används [3]. Dessutom använder Trafikverket GNSS för att i realtid bevaka var tågen befinner sig [4].

Spårledning bygger på att både rälsen och hjulaxlarna är strömförande. Genom att lägga en spänning över de två rälsstyckena kan man mäta om kretsen sluts och detta sker endast då ett tåg kommer in på rälssegmentet. Detta sedan användas för positionering [3]. En liknande metod används för en axelräknare, där man mäter förändringen av ett pålagt magnetfält när en hjulaxel passerar in och ut ur avgränsade rälssegment [3]. Både radioblockering och CBTC använder trådlös kommunikation för att positionera tåg [3]. Gemensamt för de vanligaste metoderna – spårledning och axelräkning – är att de använder separata rälssegment som bara tillåter ett tåg i taget för att ge goda säkerhetsmarginaler [3]. Detta kommer att ligga till grund för hur reserverade spårsegment behandlas under projektet.

En stor del av den svenska tågtrafiken planeras manuellt men det sker i kombination med tågskyddssystemet Automatic Train Control 2 (ATC)[5]. ATC-systemet möjliggör att man kan framföra olika sorters fordon på en järnväg utan risk för trafikolyckor som kollisioner eller urspårningar [5]. Det svenska ATC-systemet fungerar genom att järnvägsspåren är utrustade med sändare som kommunicerar med antenner på lokomotiven [5]. Detta gör det möjligt att förmedla exempelvis den rådande högsta tillåtna hastigheten för att tågen ska kunna beräkna bromssträckor och därmed hålla sig på behörigt avstånd till andra tåg [5]. Ett nytt signalsystem vid namn European Rail Traffic Management System (ERTMS) håller på att implementeras och syftar till att sammanföra Europas rälsnätverk och förenkla transporten med tåg mellan olika länder [6]. ERTMS fungerar genom att en trafikledare planerar rutter för alla tåg, sedan kan systemet kontrollera att spåret är fritt från andra fordon och att alla växlar är i rätt läge längs dess rutt [6].

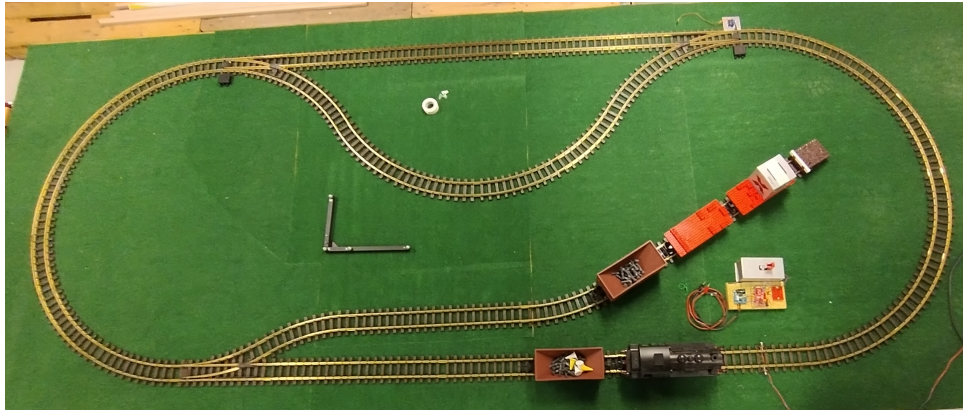
1.1.2 Skalmodell för autonom felsökning

Underhållet av rälsnätverket är en omfattande process som är tidskrävande och svårplanerad. Ett av områdena som tar mycket tid är att undersöka möjliga fel i rälsen. Detta har hittills i stor utsträckning gjorts manuellt då inga tillförlitliga automatiserade lösningar har funnits. På grund av detta har stort fokus lagts på att optimera hur arbetet genomförs. Planeringsproblemen är dock matematiskt bevisat svårlösta [7], och kostnaden för autonoma system har drastiskt sjunkit under de senaste årtiondena [8], vilket medför att det blir alltmer genomförbart att automatisera vissa delar av felsökningsprocessen med en flotta av rälsgående autonoma testriggar. Chalmers har börjat utveckla en sådan testrigg, i nuläget i syfte att lokalisera och identifiera fel i rälsen – eventuella autonoma underhållsåtgärder utförda av samma rigg ligger längre fram i tiden.

Eftersom den fullskaliga modellen måste testas på en dedikerad testbana – för närvarande en testbädd i Västerås – blir testerna både kostsamma och tidskrävande att genomföra. För att minimera antalet fullskaliga tester som krävs samt maximera effektiviteten av de tester som utförs, har en småskalig testbädd inomhus byggts som implementerar samma system i mindre skala. Detta gör det möjligt att testa integrationen av algoritmer samt undersöka att roboten har det önskade beteendet i liten skala. All funktionalitet kan inte testas genom denna metod men det är ett kostnadseffektivt sätt att öka produktiviteten och minska ledtiden. Av denna anledning är det naturligt att ett ruttplaneringsystem först implementeras i liten skala där man enkelt kan göra ändringar i rälsnätverket samt införa störningsmoment i form av andra tåg och oväntade hinder.

1.1.3 Utgångspunkt för projektet

På Chalmers fanns det en befintlig uppställning bestående av ett rälsnätverk och en skalmodell av MPIR som detta projekt utgick ifrån. Rälsnätverket var en enkel slinga med ett kort parallellt spår samt ett stickspår med ett stopblock, se Figur 1.1. Det tåg som syns på bilden är ett modelltåg som kan styras manuellt. Hela rälsnätverket är byggt i skala 1:22,5 som även betecknas G-skala.



Figur 1.1: Bild på det ursprungliga rälsnätverket, bestående av ett stickspår som syns i bildens nedre del samt en kort sträcka med parallella spår som syns i bildens övre del.

Den befintliga roboten som användes i detta projekt är baserad på ett modelltåg i G-skala. Den är utrustad med en Raspberry Pi som agerar som dator samt en mikrokontroll i form av en Arduino. Roboten var även utrustad med takometer som kan mäta antal varv hjulen har roterat samt två avståndsgivare. Den befintliga mjukvaran på roboten är baserad på Robot Operating System version 2 och kan ta emot kommandon att köra framåt, bakåt eller stanna, teori om dessa delar finns att läsa i avsnitt 2.1. Det fanns även funktionalitet på roboten för att stanna när en avståndsgivare ger ett värde under ett givet tröskelvärde. Monterade i labbet fanns även sex stycken Qualisys Pro Reflex kameror, med dessa är det möjligt att positionera markörer i rummet vilket förklaras i detalj avsnitt 2.3.

1.1.4 Samhälliga aspekter

Då detta projekt syftade att utveckla en testbädd, vilken inte har lika påtagliga samhälliga implikationer, behandlar detta avsnitt de aspekter som det överliggande WP18-projektet påverkar.

En viktig fråga när det gäller projekt som detta är hur automationen påverkar människors jobb. Inspektion av räls är i dagsläget ett personalintensivt arbete, och projektets mål är att denna process skall automatiseras och effektiviseras med hjälp av en robot. Den generella trenden inom flera delar av jobbmärknaden idag är att arbetsuppgifter automatiseras och utförs av robotar, vilket har lett till att flera politiska ledare och ekonomer har argumenterat att automationen är en bidragande faktor till att arbetslösheten fortsätter växa [9]. Genomgående är att automatiseringen främst förändrar vilka jobb som finns, att praktiska och manuella arbetsuppgifter byts ut mot tekniska och designfokuserade [9]. Då projektet kan leda till förlust av jobb så är detta viktigt att ha i åtanke under projektets fortsatta utveckling.

Det är även viktigt att diskutera vem som har ansvaret vid en olycka som involverar autonoma fordon. I takt med att det blir allt mer förekommande har det funnits behov att ta fram riktlinjer för hur man ska förhålla sig till detta problem. Den mest uppenbara lösningen är att lägga ansvaret hos tillverkarna då de har designat och utvecklat systemet som har lett till en olycka [10]. Om det var något fel eller ett designbeslut i systemet, som i vissa situationer tenderar att orsaka olyckor, så kan ansvaret läggas på producenten då de kan argumenteras att de borde upptäckt dessa brister innan produkten går till produktion [10]. Men det finns såklart en annan sida av frågan. Om man i alla situationer lägger hela ansvaret hos tillverkaren riskerar man att hämma utvecklingen av nya lösningar då de negativa konsekvenserna förknippade med olyckor gör det för riskabelt att investera i [10].

En positiv samhällspåverkan av projektet är möjligheten att minska utsläppen inom tågsektorn men även transportsektorn i stort. Vidareutveckling av MPIR så den enklare kan identifiera fel med rälsen kan leda till att reparationer genomförs snabbare, vilket bidrar till att tågen förbrukar mindre energi vid drift men även att rälsen kräver mindre underhåll. Detta eftersom skador i rälsen kan leda till ett ökat slitage på tågen och om skador på rälsen upptäcks tidigare skulle det således minska behovet av omfattande reparationer. Ytterligare kan tekniken bidra till färre förseningar, då fel kan åtgärdas innan de orsakar driftstörningar. Minskade förseningar gör tåg till ett mer pålitligt transportmedel, vilket skulle kunna leda till att fler väljer tåg framför bil – en ytterligare faktor till att projektet skulle kunna bidra till minskade utsläpp.

Ytterligare en viktig del med att tåg kommer i tid är dess viktiga del inom samhället och dess avgörande roll för många människors vardag. Det är essentiellt att alla har samma möjlighet att ta sig till de platser de behöver. För flera yrken kan en försening på endast fem minuter innebära betydande konsekvenser som inom vård, industri och service, till skillnad från kontorsarbete där arbetet ofta är mer flexibelt och arbetspassen inte är lika bundna till specifika tider. Därför är en pålitlig tågtrafik som kommer fram i tid nödvändigt för ett väl fungerande samhälle där alla har samma förutsättningar.

1.2 Syfte

Syftet med detta kandidatarbete var att utveckla den småskaliga testbädden i Chalmers lokaler. Gruppens arbete inkluderade att få roboten att kunna lokalisera sig och självständigt navigera i ett rälsnätverk genom flera styrbara växlar till en specifik punkt. Navigeringen skulle kunna ske i nätverk där vissa sträckor kan vara upptagna av andra fordon. Positioneringen skulle ske med hjälp av ett LPS baserat på ett kamerasystem från Qualisys. Detta innebar att roboten skulle kunna planera och genomföra rutter där den åkte från en given start till ett givet mål. Projektet syftade till att möjliggöra mer avancerade algoritmtester och vidareutveckling av roboten och därmed även MPIR i framtiden.

1.3 Avgränsningar

Projektet består i att utveckla testbädden med fokus på att introducera ett ruttplaneringsverktyg som använder sig av ett LPS. GNSS användes ej då det inte lämpar sig för projekt på denna skala. Integrationen av positioneringssystemet syftade till att kunna

positionera ett fordon i rälsnätverket samt fungera som stöd för att manuellt kunna föra in rälsens position till en digital modell. Ruttplaneringen skulle kunna hantera planering för en autonom robot som navigerar rälsnätverket med begränsad tillgänglighet av rälssegment samt korrekt hantera fall där ingen väg till målet går att hitta. Algoritmen skulle kunna ta fram rutter som undviker de reserverade rälssegmenten samt inkludera pauser där roboten väntar på att rälssegment ska bli lediga. Då MPIR endast rör sig på rälsnätverk – som i regel är glest i termer av grafer – samt att MPIR kommer att lösa uppgifter i relativt begränsade delar av nätverket har tidskomplexiteten ej varit centralt. Algoritmen skulle endast kunna hantera i förtid bokade rälssegment – inga realtidsavvikelser från detta schema.

Lösningarna som togs fram syftade till att vara applicerbara vid utvecklingen av MPIR men ska i första hand ses som en prototyp som förbättrar arbetsflödet vid den småskaliga testbädden. Ingen testning eller integration mot den fullskaliga roboten gjordes – endast utveckling den småskaliga testbädden. Dessutom behövde vissa etiska och samhällsliga aspekter tas i åtanke under projektets gång. Dessa behandlas i underavsnitt 1.1.4.

2

Teori

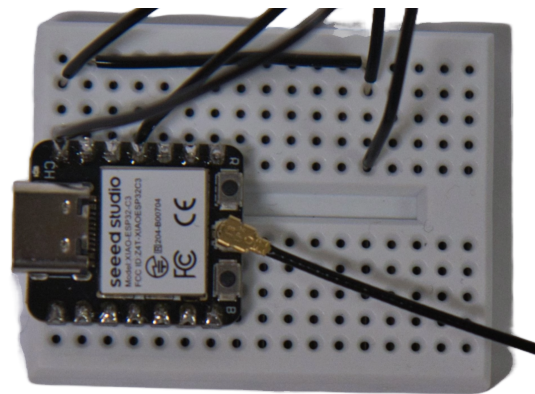
I detta kapitel presenteras den teoretiska bakgrunden som ligger till grund för projektet. Fokus ligger på de olika tekniska system som använts och hur dessa fungerar.

2.1 Styrning av delsystem

Att dela upp systemet i delsystem och möjliggöra kommunikation mellan dem är en viktig del av projektet. För att kunna styra olika delsystem kommer en liten dator i form av en Raspberry Pi att användas, se Figur 2.1a. Den har alla nödvändiga funktioner av en vanlig dator men är mer kompakt [11]. För den mer hårdvarunära styrningen kommer en mikrokontroller användas, se Figur 2.1b. En mikrokontroller är ett litet datorkort som kan programmeras och användas för att köra olika applikationer [12]. I detta projekt kommer mikrokontrollern Sseed Studio XIAO ESP32C3 att användas, framöver kallad ESP32.



(a) Raspberry Pi.



(b) Sseed Studio XIAO ESP32C3 (ESP32).

Figur 2.1: I a syns den Raspberry Pi som hanterar logiken och kommunikationen för växlarna. En av de mikrokontrollers som användes för att styra respektive växel syns i b.

För att olika delsystem ska kunna kommunicera användes Robotic Operating System version 2 (ROS ¹) [13] och nätverksprotokollet Message Queuing Telemetry Transport

¹ROS version 2 är uppföljaren till version 1 som ofta förkortas på samma vis. Detta projekt använder endast version 2.

(MQTT). MQTT är ett lättviktigt protokoll för att styra och kommunicera med uppkopplade enheter och bygger på en publicerings-/prenumerationsmodell [14].

Precis som MQTT grundar sig ROS på en publicerings-/prenumerationsmodell med hög modularitet, där ett stort antal små separata program – så kallade noder – kan utföra uppgifter och kommunicera oberoende av varandra. Detta är mycket kraftfullt för denna typ av projekt då det tillåter effektiv kommunikation och samspel mellan robotens olika sensorer, kamerapositioneringssystemet och växelstyrningen.

Inom robotik används ROS som ett väletablerat utvecklingsramverk för robotapplikationer, som i över ett decennium har utvecklats kontinuerligt av en global användargemenskap genom öppen källkod[13]. Detta har lett till en kontinuerlig förbättring av systemet och skapat ett omfattande bibliotek av färdiga kodmoduler som förenklar utveckling och implementering i nya projekt. ROS betraktas idag som industristandard inom robotik och utgör grunden för mycket av den moderna robotforskningen. Dess breda plattformstöd – inklusive Linux och Windows – gör det dessutom särskilt relevant och användbart för projektet [13].

2.2 Algoritm för ruttoptimering med reserverade rälssegment

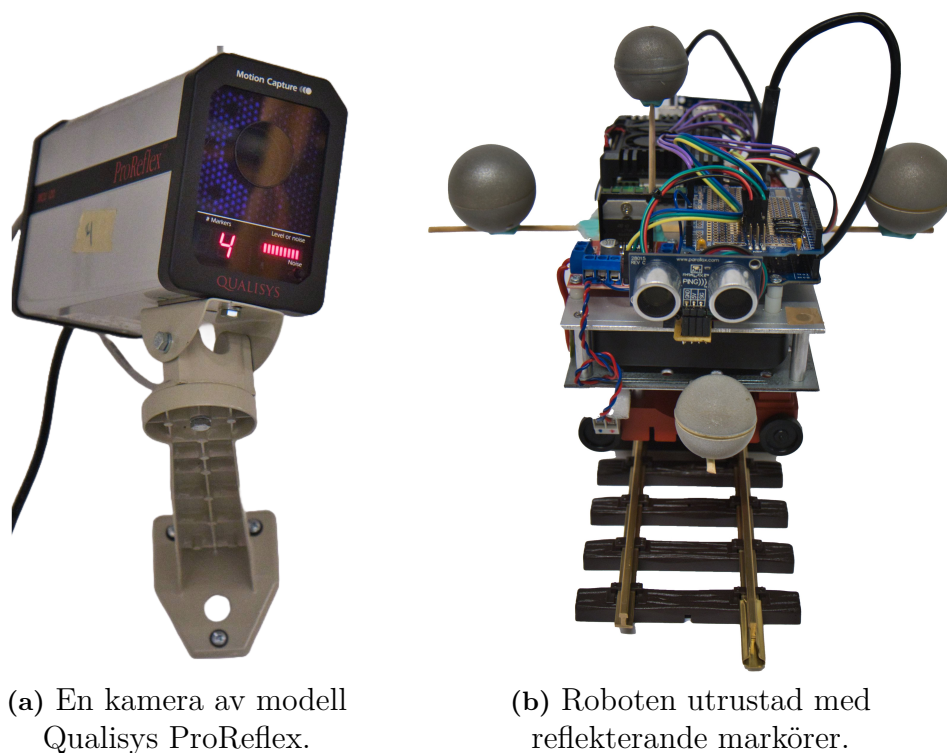
Ett kritiskt problem med ruttplaneringen är att rälssegment kan vara reserverade av andra tåg och är därför otillgängliga för roboten. Problem av denna typ – tidsberoende ruttoptimeringar – är bevisat svårlösta i det allmänna fallet och det kommer således inte gå att hitta en enkel, effektiv algoritm [15]. Olika befintliga algoritmer har studerats för att lösa detta system och dessa är ofta fokuserade på att lösa problemet för vägnätverk [16], men då dessa uppvisar många liknande karakteristiker som järnvägar är detta en lämplig utgångspunkt.

Då de tidsberoende kostnaderna i detta arbete endast används för att markera reserverade segment är möjligt att använda den euklidiska sträckan mellan två punkter som en estimering av den faktiska sträckan. Detta reducerar problemet och möjliggör användandet av algoritmer som bygger på A^* [17] och man undviker att använda de mer involverade algoritmer som beskrivs i [16]. För att fungera i denna applikation måste A^* -algoritmen modifieras för att kunna hantera tidsberoende kostnader. Detta ökar komplexiteten då det kräver att en rutt inte bara definieras av vilka noder som besöks, utan även vid vilka tidpunkter dessa besöktes. Däremot kan den ytterligare tidsinformationen tillåta roboten att vänta genom att ha segment som är kopplade till sig själva vid senare tidpunkter. Detta tillåter komplexa beteenden i algoritmen, som att åka in på ett sidospår och vänta på att noderna blir lediga innan algoritmen fortsätter.

Den höga tidskomplexiteten anses acceptabel av två anledningar: för det första skulle roboten navigera i relativt små delar av ett redan simpelt rälsnätverk, jämfört med exempelvis vägnätverk. För det andra arbetar MPIR på tidsskalor av flera sekunder, om inte minuter, vilket gör att även en relativt långsam algoritm lär hinna beräkna färdigt innan roboten behöver börja köra.

2.3 Positionering i testbädden

På grund av projektets skala är det ej aktuellt att använda ett globalt positioneringssystem – likt GNSS – eftersom ett sådant system har för låg precision. På grund av detta behöver ett LPS användas. Det LPS som implementerades i projektet bygger på triangulering av infrarödreflekterande markörer utplacerade i laborationsmiljön. Ett system för att genomföra denna typ av triangulering är Qualisys Track Manager (QTM)[18]. Det är en mjukvara som använder kameror känsliga i det infraröda spektrumet för att identifiera markörerna. Kamerorna som användes i projektet var av modell Qualisys ProReflex, se Figur 2.2. Efter kalibrering kan mjukvaran positionera en markörs tredimensionella position genom att använda kamerabilden från flera kameror och jämföra markörens storlek och position i respektive bild.



(a) En kamera av modell Qualisys ProReflex.

(b) Roboten utrustad med reflekterande markörer.

Figur 2.2: En av de kameror som användes i projektet syns i figur a, upphängningen gör att kameran kan justeras i både rotation och infallsvinkel. I figur b syns de markörer som QTM använder för att positionera en kropp.

Tidigare studier har genomförts för att utvärdera QTM och då har systemet med hög tillförlitlighet lyckats identifiera markörer placerade på ett rörligt fordon [19]. Om en samling punkter som sitter på samma föremål – exempelvis en robot – ska följas kan en *kropp* definieras. Detta gör det möjligt att följa kroppens förflyttning i alla frihetsgrader vilket möjliggör bestämning av både position och rotation. Det ökar även redundansen då det endast krävs tre punkter för att räkna ut hela kroppens position och orientering.

Algoritmen som QTM använder har strikta krav för att kunna identifiera en kropp utifrån markörer. För att underlätta för systemet nämns det i [19] därför två viktiga saker att

tänka på vid deras montering. Det första är att försöka placera markörerna på ett sådant sätt att varje par av markörer har ett unikt avstånd mellan varandra. Detta underlättar för algoritmen att identifiera vilken markör i kroppen som hittats. Den andra är att kamerorna har svårt att särskilja markörer som ligger nära eller i linje med varandra. Om detta sker leder det till att markörens storlek och således position beräknas felaktigt. För att undvika detta bör antalet markörer på kroppen minimeras – dock krävs fortfarande minst tre – samt att de ska placeras långt ifrån varandra och i olika plan [19]. Även att använda markörer i olika storlekar nämns som ett sätt att lösa detta problem [19].

Även i de fall QTM ej identifierar en kropp kommer systemet fortfarande att försöka positionera de markörer som tillhör kroppen. Utifrån dessa punkter är det möjligt att approximeras kroppens position. I det enklaste fallet kan positionen approximeras som medelvärde av de markörer som blivit identifierade. Detta ger en relativt god approximation av var kroppen befinner sig, även om detta gör att man inte kan avgöra kroppens rotation. Denna metod kan dock leda till snabba variationer i den beräknade positionen då den blir beroende på exakt vilka punkter som identifieras i just det ögonblicket. En alternativ metod är att försöka implementera en algoritm som efterliknar den QTM använder, som utifrån markörernas avstånd försöker identifiera vilka markörer det rör sig om och med dessa beräkna kroppens mittpunkt. Genom att implementera en egen version av algoritmen skulle det vara möjligt att anpassa den till problemets specifika parametrar och på så sätt kunna uppskatta kroppens position även då QTM inte lyckats.

Användningen av QTM ger en absolut position i rummet men det är även möjligt att räkna ut en relativ position. Detta kan användas för att beräkna en position utifrån en tidigare känd position, så kallad *dödräkning*. Med denna metod kan positionen beräknas även om kroppen försvinner ur kamerornas synfält. För att kunna implementera dödräkningen behöver kroppens hastighet vara känd. Med detta kan nästa position vid t_n approximeras med

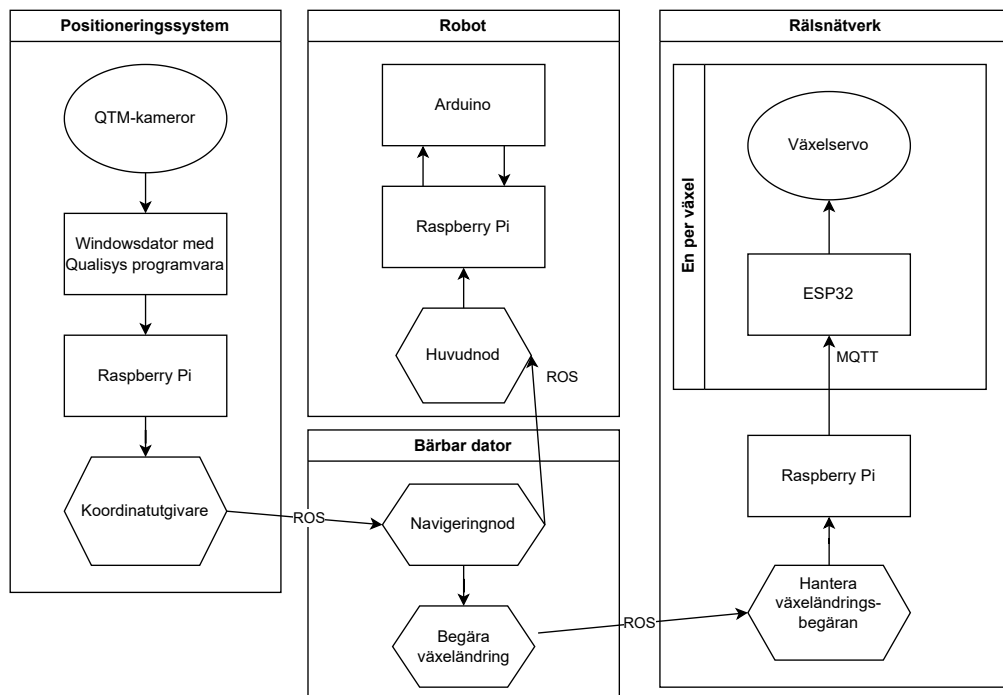
$$\mathbf{r}_{\text{kropp}}(t_n) = \mathbf{r}_{\text{kropp}}(t_{n-1}) + \mathbf{v}_n \cdot (t_n - t_{n-1}),$$

där $\mathbf{r}_{\text{kropp}}(t)$ är kroppens position vid tiden t och $\mathbf{v}_{\text{kropp}}(t)$ kroppens hastighet vid tiden t . I den aktuella applikationen är hastigheten okänd och endast farten kan bestämmas med hjälp av en takometer. För att komma runt detta problem kan olika metoder användas. En metod bygger på att man sparar en serie av tidigare kända punkter och att man utifrån dessa extrapolerar vilken riktning kroppen färdas. Detta kan ge en acceptabel approximation ifall det rör sig om korta tidsintervall eller relativt raka sträckor, men den klarar ej av partier med större riktningsförändringar. En annan metod utnyttjar det faktum att det i detta projekt är känt att kroppen rör sig längs en räls, vilket kan användas för att förbättra approximationen. Under antagandet att sträckan som kroppen förflyttats måste ske längs den aktuella rälsen kan man få en bättre bild av positionen, som dessutom klarar av att hantera kurvor. Denna metod kräver dock mer information om kroppens position än vad QTM har tillgång till och den skulle behöva integreras helt och hållet i den specifika applikationen vilket ställer högre krav på utvecklingen.

3

Design och implementation

Detta projekt består av användning av mätutrustning och sedan utveckling av både hårdvara och mjukvara. Systemet som har utvecklats består av flera datorer och processer, som kommunicerar via ROS och MQTT. I Figur 3.1 ses ett översiktligt diagram över vilka datorer och mikrokontrollers som ingår i systemet och i stora drag hur de kommunicerar. De rälsystem som fanns vid projektets början, se Figur 1.1, har byggts ut och utrustats med styrbara växlar och trafikljus. Mjukvara för att styra växlarna och en struktur för att representera rälsnätverket digitalt har tagits fram och med hjälp av ett positioneringssystem har en digital representation av rälsnätverkets geometri genererats. En algoritm har skapats som räknar ut den tidsmässigt kortaste rutten för roboten att ta sig till ett specifikt segment av rälsen, sedan har ett program skrivits som skickar dess instruktioner till roboten och växlarna så den kan utföra dem.



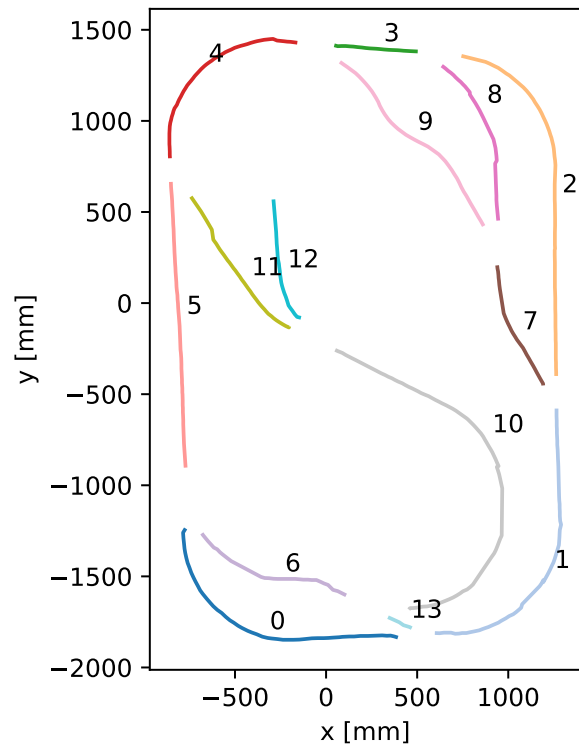
Figur 3.1: Översiktlig systembeskrivning med de ingående styrenheterna. Robotens navigeringsnod hämtar sin information från LPS och använder denna för att ta beslut som den sedan skickar till rälsnätverket och robotens huvudnod. All kommunikation mellan dessa olika system sker över ROS medan kommunikationen inom rälsystemet sker via MQTT.

3.1 Utbyggt rälssystem

Det befintliga rälssystem som fanns innan projektets start byggdes ut och komplexiteten i systemet ökades genom att införa fler delsträckor och växlar. Längre partier med parallella spår lades till för att testa robustheten hos positioneringssystemet. I Figur 3.2a ses det utvecklade rälsnätverket som är något större än den tidigare rälsuppställningen med en ungefärlig dimension på $2\text{ m} \times 3\text{ m}$. Ytterligare ökade denna förändring komplexiteten genom att antal växlar gick från fyra till nio. För navigeringssyften hade alla enskilda rälssegment fått namn under projektets gång. Dessa kan ses i Figur 3.2b.



(a) Det fysiska rälsnätverket i labbet.



(b) En digital inläsning av rälsnätverket.

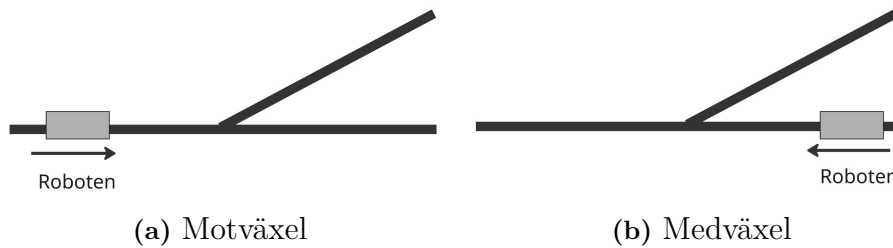
Figur 3.2: I figuren presenteras det utbyggda rälsnätverket. I den digitala representationen är även de olika primärsegment identifierade och numrerade.

3.2 Smarta växlar

För att styra fordon på rälsen modifierades växlarna så att de kunde styras elektroniskt från en central dator. Detta löstes genom att montera en servo tillsammans med en mikrokontroller vid varje växel. Servon ansvarar för den mekaniska rörelsen av växeln, medan mikrokontrollern hanterar styrsignaler och kommunikation med det centrala systemet. Det nya spåret innehåller nio växlar och varje växel har varsin mikrokontroller, servo och

tre trafikljus för att både kunna styra växeln och visa med en röd respektive grön ljusdiod att spåret är öppet i bestämd färdriktning.

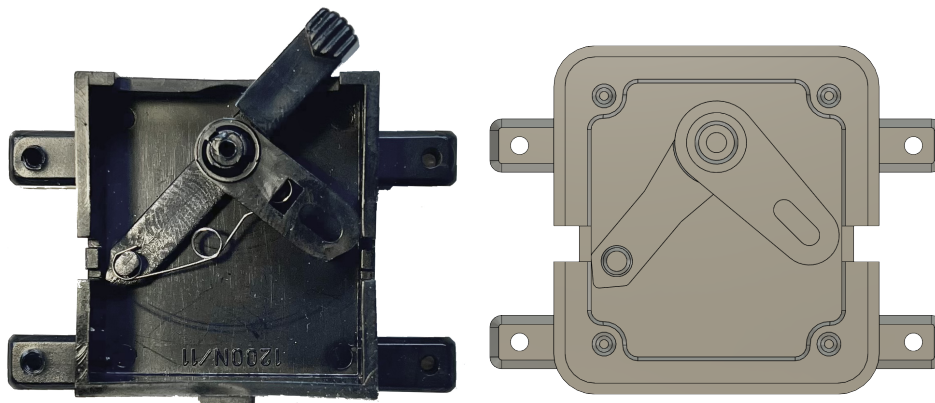
Typen av växeln benämns olika beroende på riktningen som en växel passeras. Om fordonet färdas mot förgreningen kallas det för *motväxel*, och på motsvarande sätt kallas det *medväxel* när fordonet färdas längs med förgreningen, se Figur 3.3.



Figur 3.3: Exempel på en motväxel och en medväxling, typen av växel bestäms av riktningen som växeln passeras.

3.2.1 Mekanisk modifiering av växlar

I sitt originalutförande styrs växeln manuellt genom en mekanisk spak som genom ett enkelt mekaniskt länkeage förflyttar en växeltunga mellan två spårlägen. Denna rörelse tillåter tåg att fortsätta rakt fram eller växla till ett sidospår. Växeltungan hålls i position med hjälp av en fjädermekanism i växeln som ger ett tydligt och stabilt läge i båda riktningarna. Fjädermekanismen tillåter även att ett tåg som kör medväxel kan passera växeln utan att spåra ut då växeltungorna kan fjädras tillbaka om växeln är felaktigt ställd.



(a) Manuell växel från Märklin.

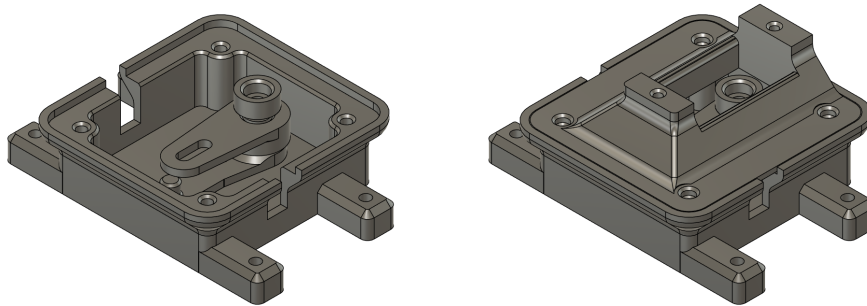
(b) Uppdaterad växel.

Figur 3.4: Den vänstra växeln är den ursprungliga och har en utstickande arm för manuell styrning. Den högra är den utvecklade designen som bibehåller den tekniska lösningen utan den manuella styrningen men med monteringsplats för en servomotor.

För att möjliggöra automatisk styrning krävdes att växlar kunde styras av ett ställdon.

För att bibehålla funktionaliteten hos den manuella växeln valdes därför att efterlikna samma utförande på det mekaniska länkaget samt att återanvända den inbyggda fjädern, se Figur 3.4. En mikro servo av modellen MG90S valdes för sin enkelhet, låga kostnad och lämpliga dimensioner.

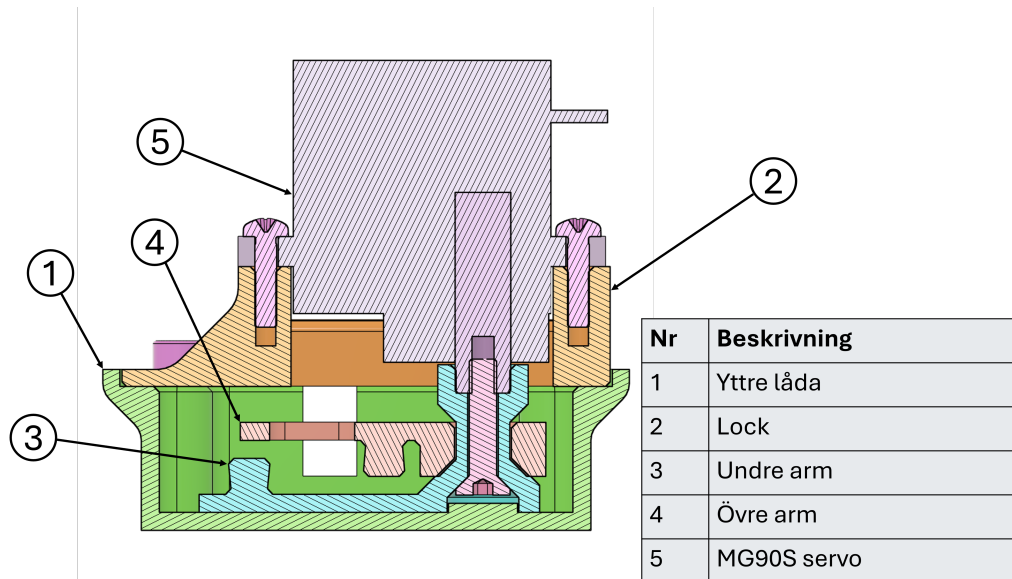
Konstruktionen av den nya växelmekanismen inleddes med att en noggrann CAD-modell av MG90S-servon togs fram. Därefter mättes och modellerades växels ursprungliga delar. Den färdiga modellen kombinerades med servon i en skyddande låda som passade växels infästningspunkter, se Figur 3.5.



(a) Uppdaterad växel utan lock. (b) Uppdaterad växel med lock.

Figur 3.5: Uppdaterad växeln med och utan lock.

Fjädermekanismen från originalväxeln återanvändes i den modifierade versionen för att bevara växels låsfunktion. Servon (5) i Figur 3.6 är monterat med en undre arm (3) som roterar 90° åt respektive håll, vilket möjliggör växling mellan de två lägena. Den övre armen (4) rör sig då i motsatt riktning, där fjädern hjälper till att låsa växeltungan i rätt position. Se Figur 3.6 för en genomskärning av konstruktionen, där även den yttre lådan (1) och locket (2) framgår.



Figur 3.6: Tvärsnitt på växel.

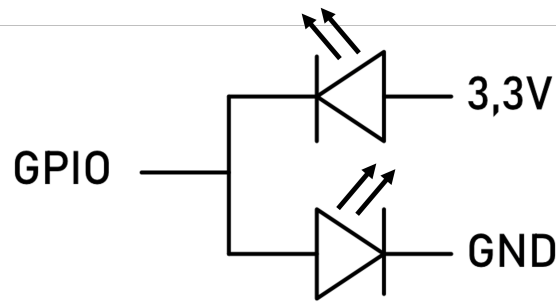
3.2.2 Elektronik och styrsystem

För styrningen av varje växel används en ESP32. Den valdes tack vare sin kompakta formfaktor, låga strömförbrukning, inbyggda WiFi-modul och tillräckligt många General Purpose Input Output-anslutningar (GPIO) för att styra både ett servo och tre trafikljusdioder. Varje mikrokontroller är monterad direkt vid sin växel och ansvarar för att tolka inkommande styrkommandon och utföra motsvarande mekaniska rörelser.

För att möjliggöra trådlös kommunikation mellan växlarna och en Raspberry Pi användes kommunikationsprotokollet MQTT. Protokollet valdes på grund av dess låga systembelastning, tillförlitlighet och publicerings-/prenumerationsmodell som lämpar sig väl för distribuerade inbyggda system med begränsade resurser. Varje växel i nätverket är utrustad med en ESP32 som fungerar som MQTT-klient. Dessa mikrokontroller är enbart prenumeranter och lyssnar på varsin individuella kanal. På Raspberry Pi:n körs två ROS-noder, en som väntar på förfrågan om att ställa ljus och en som väntar på förfrågan om växling. När en förfrågan kommer ifrån en annan ROS-nod skickas ett MQTT-meddelande på rätt kanal så den önskade ESP32:n kan ställa om växeln eller ljusen.

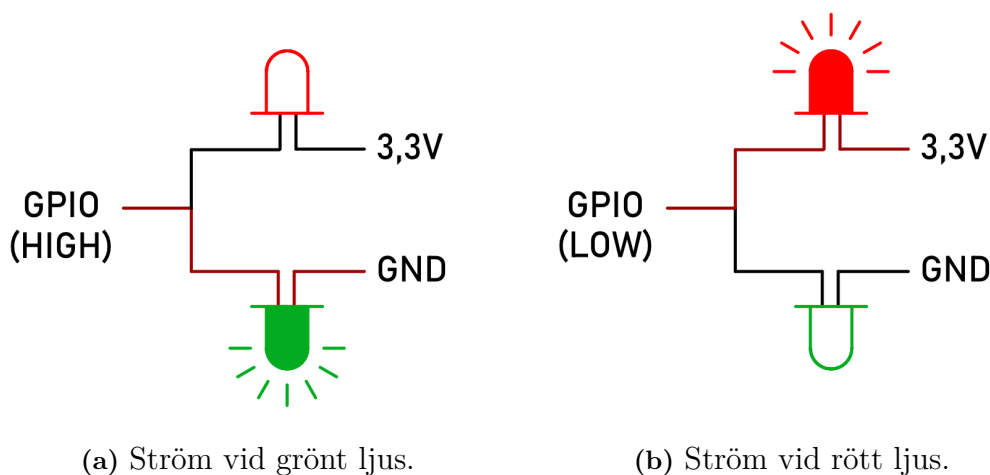
3.2.3 Trafikljus och styrlogik

Trafikljusen har ett rent visuellt syfte och indikerar i vilken riktning tåg är tillåtna att köra. Tre trafikljus placerades vid varje växel för att indikera respektive körriktning. Om en ljusdiod i en växel är grön då ett tåg närmar sig den, är de resterande röda.



Figur 3.7: Kopplingsschema för ett trafikljus.

För att minska behovet av flera GPIO-anslutningar användes en kopplingslösning där varje ljusdiod (röd och grön) delar samma GPIO. Den röda ljusdiodens katod är kopplad till 3,3V, medan dess anod går till GPIO:n. När GPIO:n är satt till LOW (0 V) lyser den röda ljusdioden. Den gröna ljusdioden är kopplad tvärtom, vilket gör att den lyser när GPIO:n är satt till HIGH (3,3 V). Se Figur 3.7 och Figur 3.8.



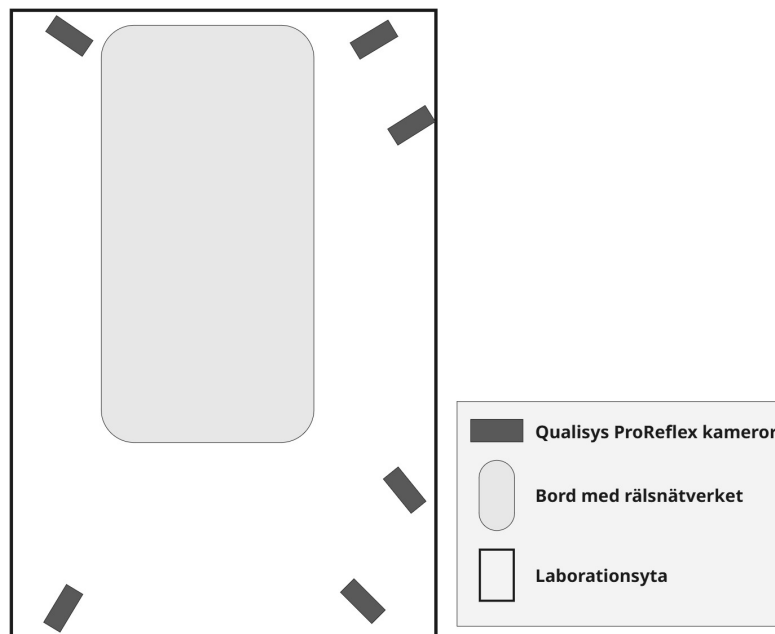
Figur 3.8: Strömmen för trafikljusen vid olika GPIO signaler.

Trots färdig hårdvarukonstruktion prioriterades inte programmeringen av trafikljusen, vilket innebar att deras funktion inte implementerades i den slutgiltiga lösningen.

3.3 Integrering av ett positioneringssystem

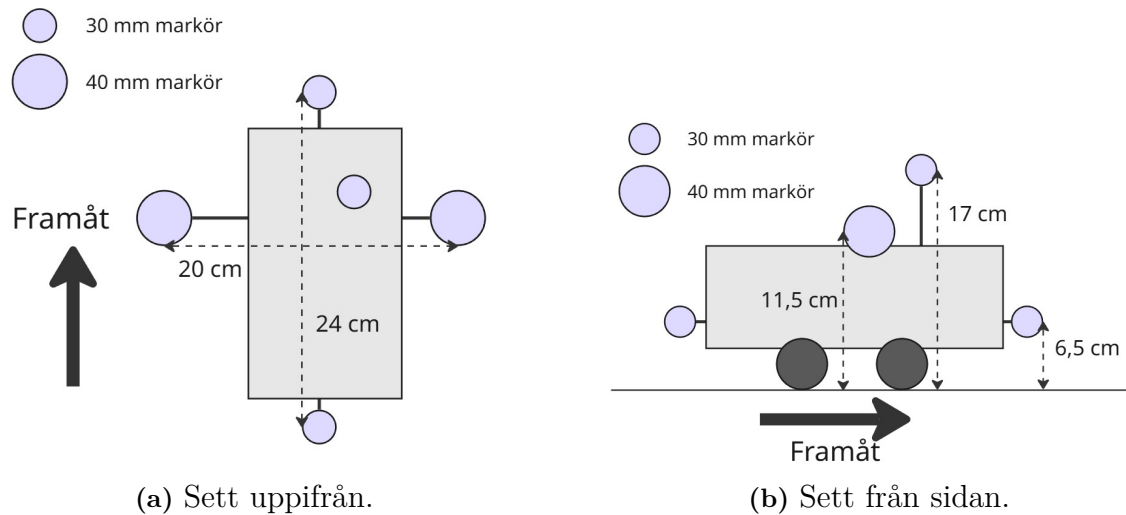
För att kunna positionera roboten samt läsa in rälsens position användes ett LPS som bygger på Qualisys ProReflex-kameror och totalt användes sex kameror placerade runt rälsnätverket enligt Figur 3.9. Kamerornas upphängning var fastmonterad i labbet men genom att justera riktning och infallsvinkel var det möjligt att erhålla täckningen av hela rälsnätverket. Genom att utrusta roboten med flera reflektiva markörer var det möjligt att bestämma dess absoluta position i rummet. Placeringen av markörerna syftade till att avstånden mellan alla par av markörer skulle vara unikt, i enlighet med teorin i avsnitt 2.3. Detta uppnåddes genom att markörerna placerades asymmetriskt kring robotens centrum och i flera olika höjder, se Figur 3.10. För att undvika det andra problemet som nämns

i avsnitt 2.3 – att QTM felaktigt slår samman olika punkter – användes markörer med både 30 mm och 40 mm diameter [19].



Figur 3.9: Skiss över kamerauppställningen i labbet. Totalt användes sex kameror i projektet.

Robotens position lästes in i realtid via kamerasystemet och via QTM identifierades de markörer som konstituerar roboten. Genom att använda Qualisys egna Python Source Development Kit (SDK) [20] kunde QTM strömma data direkt över nätverket till en mottagande enhet. I detta fall användes en Raspberry Pi, som med hjälp av SDK:n kommunicerade med en Windows-dator via ett lokalt nätverk, där QTM kördes och styrde kamerorna. När QTM sände data, tolkades och avkodades koordinatinformationen av programvaran på Raspberry Pi:n. Därefter publicerades dessa koordinater i realtid till en specifik ROS-kanal, vilket möjliggjorde kontinuerlig spårning av robotens position.



Figur 3.10: Placeringen av de reflektiva markörerna på roboten. Notera hur markörerna sitter asymmetriskt kring robotens centrum för att säkerställa att avstånden mellan varje par av markörer är unikt.

Roboten definierades som en kropp i QTM, och kroppens position strömmades över nätverket. Eftersom QTM har strikta krav för att identifiera en kropp behövde implementationen hantera att kroppens position stundtals försvann. Då detta skedde identifierades fortfarande en del av kroppens punkter, och dessa strömmades parallellt med kroppens data. När kroppens definition försvann i programmet och inte var tillgänglig för SDK:n användes istället en algoritm som beräknade kroppens position utifrån punkternas medelvärde.

Även om kamerorna som användes har uppvisat stor noggrannhet i tidigare tester [19] var det fortfarande intressant att studera metoder för att öka tillförlitligheten. Särskilt intressant var att undersöka metoder för att hantera störningar i kamerasystemet som uppkom av andra reflekterande material eller skymd sikt. Dessa problem återspeglar de begränsningar som finns för GNSS och som MPIR behöver kunna hantera. Därför påbörjades en utveckling av en positionsavgörare som med hjälp av den inbyggda takometern kompletterade LPS. Med takometern var det möjligt att räkna ut tågets position baserat på hur många varv hjulen roterat. Genom att räkna ut förhållandet mellan de steg takometern uppmätt mot avståndet roboten hade färdats togs förhållandet på 11,22 mm/steg fram. Med detta förhållande var det möjligt att avgöra robotens förflyttade sträcka helt oberoende av LPS.

3.4 Digital representation av rälsen

För att kunna representera rälsen digitalt skapades en datastruktur för att hantera rälsen och de olika kopplingarna. Datastrukturen bygger på ett par enkla principer som grundar sig i att rälsen delas in i *segment* där varje segment motsvarar ett rälsavsnitt som enbart kan ockuperas av ett fordon åt gången. Dessa segment består i sin tur av en samling punkter som representerar rälsens faktiska utformning för att på ett korrekt sätt ta hänsyn till kurvor. Rälsen i sig saknar en egendefinierad riktning, men för att kunna särskilja de

olika ändarna av rälsen väljs den ena änden godtyckligt till start och den andra till slut. Ordningen på punkterna i segmentet definieras för att den ska gå från startpunkten till slutpunkten. Varje segment sparar sedan vilka segment som är kopplade till respektive ände och utifrån det kan rälsnätverket byggas upp och traverseras likt en graf. Notera att växlarna aldrig explicit definieras i datastrukturen och de representeras istället av noder med flera utgående kanter.

För att praktiskt kunna översätta ett större rälsnätverk med denna metod användes *primärsegment* som motsvarar rälsavsnitt som saknar växlar eller andra förbindelser med andra spår. Dessa primärsegment och deras relationer till de andra primärsegmenten definieras sedan med hjälp av en grannlista. Eftersom detta arbete ej tagit fram ett verktyg för att automatiskt ta fram grannlistan är detta ett effektivt sätt att minimera arbetet att digitalisera en ny konfiguration då även ett relativt stort nätverk ofta består av få växlar. I ett verkligt tågnät delas även rälssegment utan växlar upp i mindre segment för att tillåta flera tåg att åka efter varandra. Denna karakteristik var möjlig att återskapa i detta projekt genom att primärsegmentet delas upp ytterligare utifrån en önskad längd på segmenten. I projektet sattes en undre gräns på längden av ett rälssegment till två robotlängder, vilket i detta fall motsvarade 50 cm. Ett primärsegment delas aldrig in i segment kortare än denna distans för att garantera ett lämpligt säkerhetsavstånd.

I praktiken implemeterades datastrukturen som en `geopandas.GeoDataFrame` där varje segment motsvarar en rad och identifierades med sin index. I Tabell 3.1 ses ett exempel på raden för segment 7 ur Figur 3.2b, segmentet startar i segment 1 varefter det förgrenar sig i en växel till segment 8 respektive 9. Geometrin för respektive segment sparas som en `shapely.LineString` bestående av ett antal koordinatpunkter längs rälsen, för att minimera upprepade beräkningar sparas även segmentets längd.

Tabell 3.1: Exempel på ett rälssegment sparad som en `geopandas.GeoDataFrame`, för varje segment beräknas längden och sparas för att minimera beräkningar i ruttplaneringsalgoritmen. Varje segment sparar även ett unikt schema för dess reservering.

Index	Start	End	Geometri	Längd [mm]	Schema
7	1	8, 9	$[(91, -82), \dots, (87, -75)]$	453	14:07-14:23

För att entydigt kunna översätta en förgrening från grannlistan till verkligheten är deras inbördes relation definierad enligt Definition 1. Med denna definition identifieras det korrekt att segment 7 i Tabell 3.1 kopplar till segment 9 genom en växel ställd åt höger.

Definition 1 (Entydighet av riktning vid en förgrening) *Vi en sammankoppling av fler än två segment definieras riktningarna utifrån det segment som förgrenas. Stående i det förgrenade segmentet, i riktning mot förgreningen, är segmentet utgående åt vänster det första segmentet och segmentet utgående åt höger det andra segmentet i förteckningen.*

Det är även viktigt att den digitala representationen kan korrekt hantera både med- och motväxlar, se Figur 3.3. Även om riktningen inte nödvändigtvis ändras då en medväxel passeras är det viktigt att det ställs i rätt läge för att inte riskera urspårning och slitage.

Då den digitala representationen sparar i vilken ände olika segment är kopplade går det att räkna ut typ av växel utifrån vilken rutt roboten kommer färdas.

3.5 Implementation av tidsberoende ruttplanering

Ett av de huvudsakliga målen med projektet var att roboten autonomt ska kunna navigera mellan två olika punkter i ett rälsnätverk. Att genomföra detta i praktiken omfattar många olika komponenter och i detta avsnitt behandlas den mjukvara som tagits fram för att planera vilken väg som roboten ska ta.

Att planera en rutt definieras i detta fall som att ta fram en serie av segment som roboten behöver röra sig över för att ta sig från startpunkten till slutpunkten. Med grund i den datastruktur som diskuteras i avsnitt 3.4 kan rälsnätverket representeras av en graf, och således kan problemet lösas med grafsökning. Rutten som planeras ska minimera restiden samt undvika befintlig trafik på rälsnätverket. En vanlig algoritm för grafsökning är A^* [17], som byggdes ut för att hantera tidsdimensionen i problemet. För att kunna implementera denna algoritm kommer följande frågor besvaras:

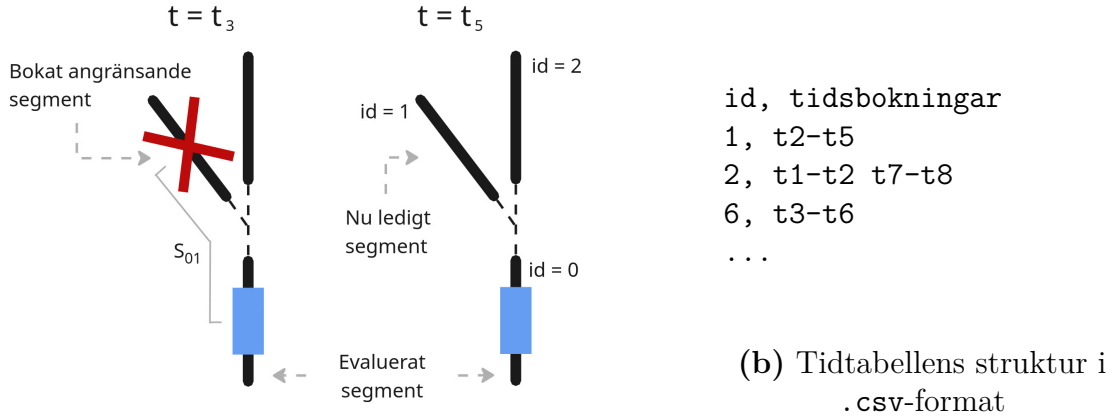
- Hur definieras en nod i grafen?
- Hur är noderna sammankopplade?
- Hur beräknas kostnaderna mellan noderna?

Då algoritmen utöver att hitta den rutt som minimerar restiden även ska undvika den befintliga tidsberoende trafiken, behöver den ta hänsyn till att delar av rälsnätverket kan vara uppbokade under vissa tidsintervall. En nod definieras därför av både ett segment och tidpunkten då algoritmen planerar att ankomma till segmentet.

Ett viktigt beteende för en tidsberoende grafsökning är att roboten behöver kunna stanna och vänta på ett segment – exempelvis ett sidospår – för att invänta annan trafik på rälsen. Detta mer komplexa beteende gör att algoritmen kan hantera utmanande nätverk som är mer verklighetstroga, men innebär att varje segment behöver kopplas till sig självt i sökningen utöver de länkar som beskrivs i avsnitt 3.4. Med benämningarna i Tabell 3.1 fås en nods angränsande noder således av

$$\text{angränsande} := \{s \mid s \in \text{Start} \cup \text{End} \cup \text{Index}\} \quad .$$

Notera att kopplingen tillbaka till samma *segment* leder till en ny *nod*, då roboten genom att följa denna länk väntar en stund och därför anländer i samma segment vid en senare tidpunkt.



(a) Återevaluering av ett segment med bokad granne vid senare tidpunkt.

Figur 3.11: Algoritmen hittar pågående bokningar i tidtabellen vid $t = t_3$ och lägger till ytterligare evaluering av samma segment vid $t = t_5$. En ytterligare evaluering kommer läggas till i $t = t_8$, eftersom t_8 är nästa tidpunkt då ett angränsande segment blir ledigt efter t_5 .

När algoritmen väntar på ett segment, gör den det så lång tid det tar för nästa angränsande segment att bli ledigt i schemat (se rad 15 i algoritm 1). I Figur 3.11 motsvarar detta att den väntar från $t = t_3$ till $t = t_5$. På så vis undviker den att lägga till onödigt många framtida tidpunkter att evaluera till sökningen, som exempelvis $t = t_4$ då situationen inte har förändrats. En bieffekt av detta är att roboten väljer att vänta så nära in på bokade segment som möjligt, eftersom väntandet bara kan inträffa då minst ett angränsande segment har en pågående eller framtida bokning inom tidsramen för sökningen.

Kostnaden mellan noder definieras i tiden det tar att färdas mellan nodernas mittpunkter. Detta gör det möjligt för grafsökaren att korrekt bestämma ankomsttiden vid nya noder. För att bestämma restiden utifrån den kända sträckan mellan segment approximeras robotens hastighet med en medelhastighet. Denna hastighet antas var konstant genom hela grafsökningen och är baserad på empiriska data. Med en konstant hastighet v_m erhålls restiden $t_{i,j}$ mellan två segment i och j av

$$t_{i,j} = \frac{s_{ij}}{v_m} = \frac{l_i/2 + l_j/2 + l_{i,j}}{v_m},$$

där l_i och l_j är längden på segment i respektive j och $l_{i,j}$ är avståndet mellan de båda segmentens ändpunkter (se sträckan s_{01} i Figur 3.11). För att algoritmen ska undvika reserverade segment läggs endast de noder som motsvarar lediga segment-tidspar till bland de möjliga rutterna. Detta går att likna med att kostnaden till otillgängliga segment är oändligt hög, och att algoritmen avfärdar rutten som skulle ta oändlig tid på grund av denna kostnad.

Dessa kostnader summeras längs den rutt som når en given nod tidigast för att ge restiden från startnoden till den givna noden. I A^* uppdateras varje nods tidigaste ankomsttid löpande varje gång en snabbare väg dit hittas, och det antas att varje delsträcka som kan ersättas med en snabbare väg fortfarande kommer nå målet [17]. Detta gestaltas av funktionen

$$f(n) = g(n) + h(n) ,$$

där $g(n)$ är den hittills snabbaste funna vägen från start till en given nod n , och h är en heuristikfunktion som ger minsta möjliga tid som måste återstå från n till målet. h utgår i detta projekt från den euklidiska sträckan mellan punkterna för att beräkna restiden. A* tar den nod som har lägst f -värde då den ska välja en ny nod att evaluera, vilket leder till att den söker genom de snabbaste kandidatrutterna först. Detsamma är även sant i robotens modifierade version, men eftersom tidsdimensionen är strängt ökande genom hela sökningen kan algoritmen aldrig hamna i situationen att en snabbare rutt till en tidigare evaluerad nod hittas. Notera även att algoritm 1 använder (*segment, tid*)-notation, då segment och tidpunkt är de två datum som utgör en nod.

Algoritm 1 TIDSBEROENDE-A*(start, slut, t_0 , v_m , rälsnätverk)

```

1: Input: start, slut,  $t_0$ ,  $v_m$  och rälsnätverk
2: Output: rutten med kortast restid från start till slut.

3: function FINN-TIDSBEROENDE-RUTT(start, slut,  $t_0$ ,  $v_m$ , rälsnätverk)
4:    $Q := \{(\text{start}, t_0)\}$ 
5:    $g(\text{start}, t_0) := 0$ 
6:    $f(\text{start}) := h(\text{start})$ 
7:    $\forall n : n \neq \text{start} \implies f(n) := \infty$ 

8:   while  $Q \neq \emptyset$  do
9:      $\text{segment}_{\text{nu}}, t_{\text{nu}} := \min_{(n,t) \in Q} (f(n))$ 
10:     $Q := Q \setminus \{\text{segment}_{\text{nu}}\}$ 
11:    if  $\text{segment}_{\text{nu}} = \text{slut}$  then return rutt(segmentnu,  $t_{\text{nu}}$ )
12:    Låt grannar vara mängden med alla angränsande segment till segmentnu.
13:    for granne in grannar do
14:      if granne = segmentnu then
15:         $t_{\text{ankomst}} := t_{\text{nu}} + \text{nästa\_lediga\_starttid}(\text{grannar} \setminus \{\text{segment}_{\text{nu}}\})$ 
16:      else
17:         $t_{\text{ankomst}} := g(\text{segment}_{\text{nu}}, t_{\text{nu}}) + \text{restid\_mellan}(\text{segment}_{\text{nu}}, \text{granne})$ 

18:      if bokad(granne,  $t_{\text{ankomst}}$ ,  $v_m$ ) or  $t_{\text{ankomst}} > \text{deadline}$  then continue
19:      else
20:         $\text{rutt}(\text{granne}, t_{\text{ankomst}}) := [\text{rutt}(\text{segment}_{\text{nu}}, t_{\text{nu}}), (\text{granne}, t_{\text{ankomst}})]$ 
21:         $g(\text{granne}, t_{\text{ankomst}}) := t_{\text{ankomst}}$ 
22:         $f(\text{granne}) := t_{\text{ankomst}} + h(\text{granne})$ 
23:         $Q := Q \cup \{(\text{granne}, t_{\text{ankomst}})\}$ 

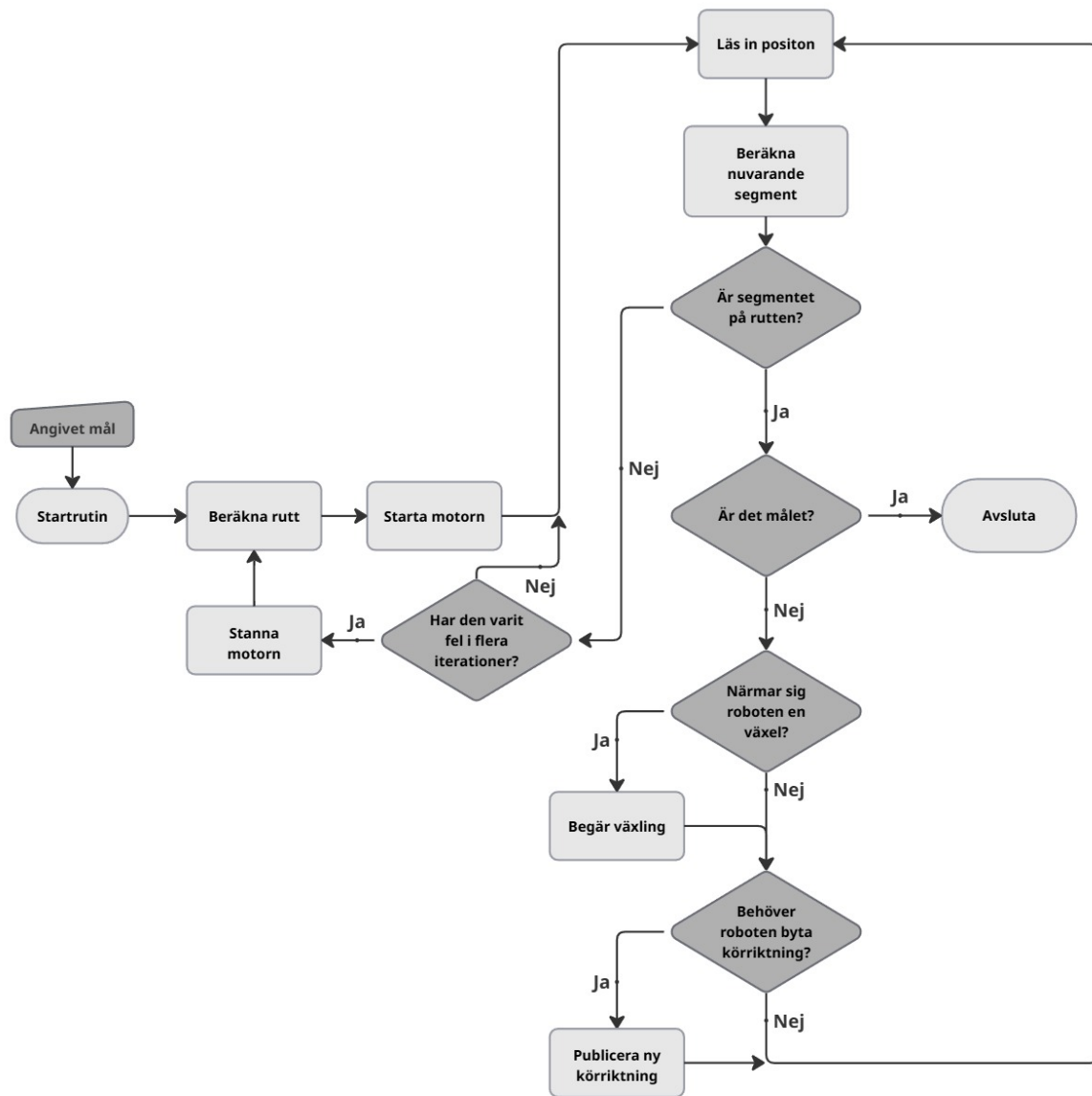
```

3.6 Beslutsfattning hos roboten

Roboten sätter samman alla de olika delarna som nämnts tidigare i kapitlet för att kunna navigera autonomt i rälsnätverket. Detta görs genom en navigeringsnod i ROS som matas med relevant information för att fatta beslut. Denna nod initialiseras och

startar upp noder för att kunna lyssna på koordinater, begära växlingar samt skicka motorinstruktioner.

Med alla ingående noder initialiserade påbörjas beräkning av rutt mellan den nuvarande positionen och den av användaren angivna slutpositionen, enligt algoritmen som beskrivs i avsnitt 3.5. Utifrån den beräknade ruten identifieras de växlar som kommer att passeras och till vilka lägen de behöver ställas. Efter detta kan roboten påbörja sin körning och dess beteende styrs av en loop som beskrivs av flödesschemat i Figur 3.12.



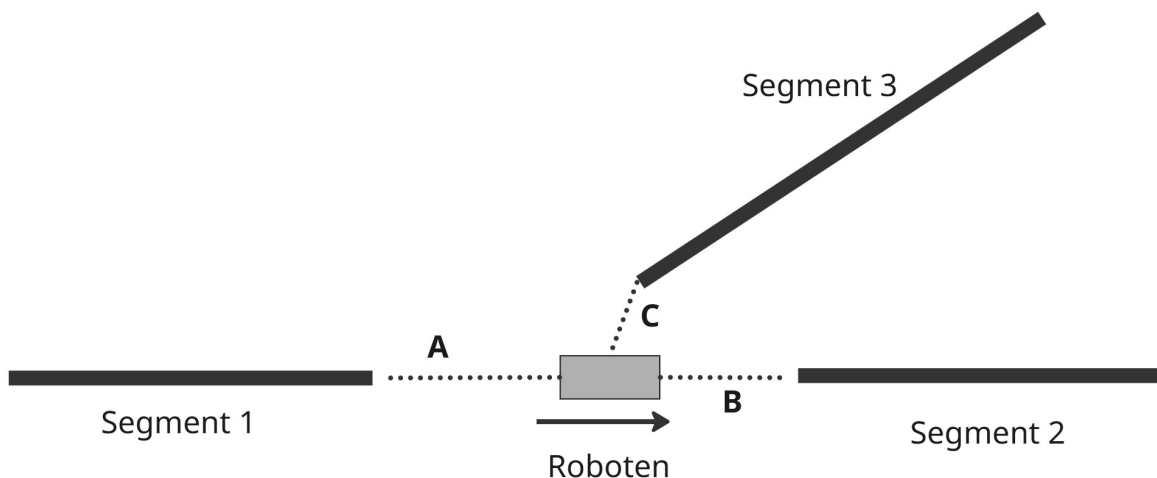
Figur 3.12: Flödesschema för robotens navigeringsnod som sköter kommunikation kring växling och motorstyrning.

Varje iteration i loopen börjar med att roboten läser in sin position och försöker översätta denna till sin faktiska position på rälsen. Med denna information börjar den med att undersöka ifall positionen är tillåten och om den inte är det påbörjas ett felsökningsprotokoll. Detta tillåter den att avgöra ifall det är en tillfällig feltolkning av LPS-data eller

en faktisk felkörning som kräver en omplanering av ruten. Om robotens position var tillfredsställande kontrollerar den först ifall den ankommit målet – varefter den avslutar – och annars kontrollerar den ifall den behöver begära en växling eller ny motorriktning. Efter att möjliga förfrågningar gjorts startar loopen om.

En av de stora utmaningarna för roboten är steget att korrekt bestämma sin position på rälsen. Även om det är möjligt att lokalisera roboten i rummet med hög noggrannhet är det svårt att på ett tillförlitligt sätt projicera den punkten till ett rälssegment. För att lösa detta problem användes en kombination av olika metoder. Först och främst sparas en serie av de senaste segmenten som roboten tror att den befunnit sig på, och dess slutgiltiga position bestäms sedan av en majoritetsröstning bland de identifierade segmenten. Detta minimerar risken att enstaka felpositioneringar får konsekvenser för robotens navigering i stort. Under projektets gång identifierades 15 positioner som en lämplig avvägning mellan att få en stabil position och låg latens.

Även med denna metod för att säkerställa vilket segment roboten befinner sig på är det möjligt att roboten felaktigt identifierar segmentet i övergångsområden. Eftersom växlarna aldrig explicit definieras i den digitala representationen befinner sig roboten i en gråzon på själva växeln. På dessa partier är det möjligt att det närmsta segmentet ges av ett angränsande segment som ej är en del av den beräknade ruten. En skiss av fenomenet ses i Figur 3.13 där längden $C < B < A$ och roboten anser således att den är på segment 3. Detta löses genom att roboten har som utgångspunkt att den färdas längs med sin angivna rutt och att den därför bara accepterar ett segment längs med ruten.



Figur 3.13: När roboten färdas över växeln mellan segment 1 och segment 2 kommer segment 3 vara det närmsta segmentet under delar av övergången. Genom att roboten utgår från att den färdas längs sin rutt kan den undvika att felaktigt identifiera vart den är på väg.

Med ett sådant system är det särskilt viktigt att roboten kan hantera felnavigeringar och detta görs genom att den sparar de tillfällen då dess position ligger utanför ruten. Om detta sker fler gånger i följd än ett fördefinierat tröskelvärde kommer roboten att stanna och försöka beräkna en ny rutt utifrån sin nya position. Detta gör att den klarar av att hantera mer utmanande situationer där exempelvis växlar hamnat i fel läge.

Roboten som användes är symmetrisk i bemärkelsen att den saknar en definierad fram-

riktning och motorn är lika stark i båda riktningar. Detta är en av dess stora styrkor då den klarar av att byta riktning utan att behöva vända. För att kunna navigera korrekt är det dock avgörande att roboten alltid vet vilken riktning den rör sig längs rälsen samt vilken polaritet motorn har. På grund av loopar i rälsen, se segment $\{5, 6, 10, 11\}$ i Figur 3.2, går det inte att definiera en entydig riktning utifrån rälsegmenten. Loopen gör att roboten kan passera samma segment åt olika håll givet samma polaritet på motorn. I arbetet löstes detta problem genom att roboten alltid initialiseras i en förutbestämd riktning och sedan sparar roboten i vilken riktning den färdades då den ankommer till ett nytt segment. På detta sätt kan den entydigt avgöra sin nuvarande riktning samt räkna ut ifall den behöver byta körriktning för att fortsätta följa sin rutt.

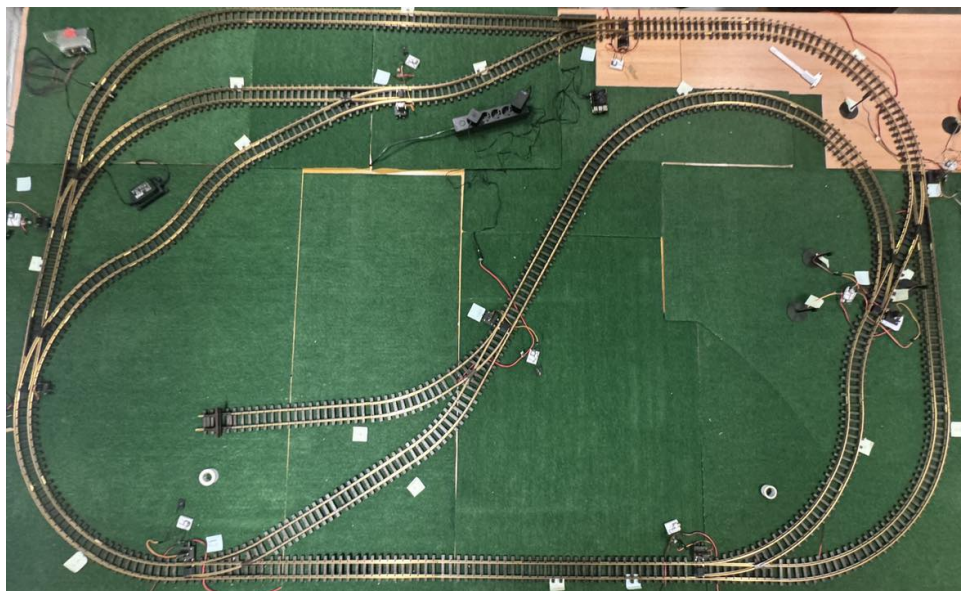
4

Resultat

I detta avsnitt presenteras resultatet för utvecklingen av testbädden, smarta växlar samt navigering och positionering för roboten.

4.1 Rälsnätverket

Det resulterande rälsnätverket kan ses i Figur 4.1. Det har en ökad komplexitet med nio styrbara växlar och flera längre spårsegment.



Figur 4.1: Bild på den slutgiltiga testbädden.

De modifierade växlarne fungerar pålitligt vid drift, och samtliga nio enheter kan styras korrekt från det centrala systemet. Fjädermekanismen från originalväxeln bevaras i konstruktionen, vilket tillåter tåg att köra medväxel även om växeltungan inte är korrekt inställd. Detta minskar risken för urspårning och säkerställer en robust funktion.

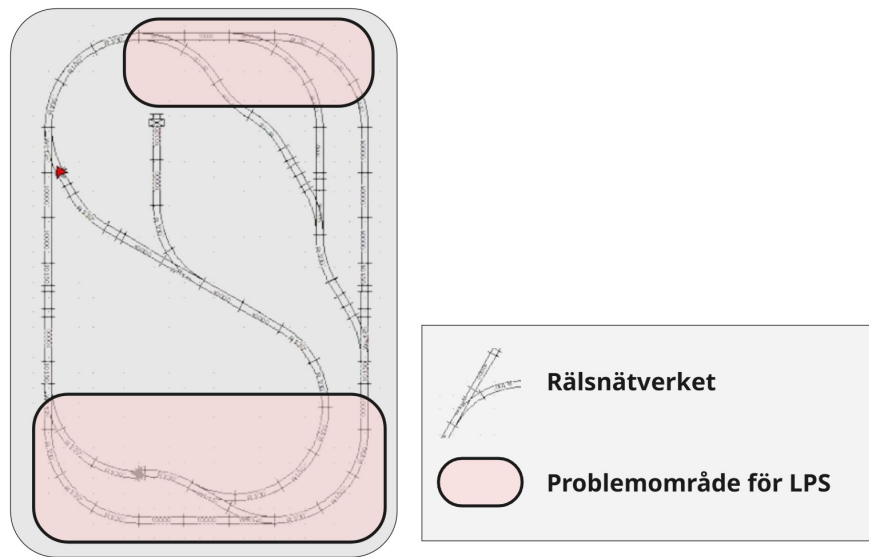
Den servobaserade rörelsen efterliknar det ursprungliga länkaget tillräckligt väl för att uppnå full växling, och den skyddande lådan ger tillräckligt mekaniskt skydd under drift. Vissa växlar uppvisar dock initialt trög rotation på grund av små mekaniska toleranser, vilket löses genom manuell bearbetning med en fil. För att minska variationen mellan olika enheter ställs servon in i mittposition innan den undre armen monteras. Trots dessa mindre justeringar är det möjligt att reproducera samtliga nio växlar med samma modell, vilket bekräftar att lösningen är både repeterbar och tillräckligt robust för ett utbyggt nätverk.

Trafikljusen konstrueras och kopplas för att visa tillåten färdriktning vid varje växel med hjälp av röda och gröna lysdioder. Lösningen möjliggör styrning av tre ljus per växel med endast tre GPIO-utgångar genom en enkel kopplingsstrategi, se Figur 3.7. Kopplingen förhindrar dessutom att den röda och gröna lysdioden lyser samtidigt, vilket ökar robustheten. Funktionaliteten testades framgångsrikt under utvecklingsfasen, men integrerades inte i det slutgiltiga systemet på grund av tidsprioriteringar. Trafikljusen är dock färdigkonstruerade och kan enkelt monteras vid ett senare tillfälle. Det saknas i nuläget logik i programvaran för att växla trafikljusen automatiskt under robotens körning. Kod för manuell styrning finns kompilerad på ESP32-enheterna, och inkopplade trafikljus kan ändras via manuella MQTT-meddelanden.

Som en del av testbädden finns även en Raspberry Pi, på denna körs de ROS-noder som styr växelservorna och trafikljusen. Dessa noder inväntar en förfrågan från de ROS-noder som ansvarar för logiken kring när växling ska ske. När en förfrågan kommer in innehåller den information om vilken växel och till vilket läge den ska ställas. Samma gäller förfrågan om ändring av trafikljusen. Ifall en växel eller en uppsättning ljus redan står i det läge som begärts så sköter programvaran på ESP32-enheterna att de inte ställs om utan att de står kvar.

4.2 Positioneringssystemet

Det lokala positioneringssystemet kan hitta roboten och med hög precision bestämma dess position i rummet. I första hand försöker QTM identifiera positionen på den kropp som följs – i detta fall roboten – och sedan strömma informationen över ROS. I de situationer där QTM inte identifierade kroppen var det möjligt att läsa in de individuella markörerna på roboten och utifrån det beräkna en approximativ position. Vid försök där roboten navigerade runt rälsnätverket i fem minuter lyckades QTM korrekt positionera kroppen 90% av tiden. Dock var det möjligt att identifiera problemområden där systemet hade svårigheter att korrekt bestämma alla markörers position, se Figur 4.2. I dessa problemområde lyckades QTM identifiera en del av markörerna endast 60% av tiden.



Figur 4.2: I figuren är områden där QTM har försämrat prestanda markerade. I de markerade områden är täckningen en markör 60 %, vilket är lägre än snittet för hela rälsnätverket.

För att kunna positionera roboten där varken kroppen eller tillräckligt många markörer identifierades påbörjades en utveckling av ett program som skulle använda robotens takometer för att beräkna dess position. Genom att räkna antal varv robotens hjul roterat samt dess tidigare koordinater är det möjligt att bestämma dess nuvarande position med så kallad *dödräkning*. Detta program har inte implementerats färdigt och dessa luckor i positioneringen där QTM tappar kontakt kvarstår. De slutgiltiga koordinaterna skickas ut i ROS-nätverket för att plockas upp av en ROS-nod som använder robotens position.

Trots att systemet för dödräkning aldrig integrerades i robotens positionering var lokaliseringen tillfredsställande för projektets ändamål. Roboten lyckades genomföra längre körningar på mellan 3 minuter och 5 minuter där den aldrig förlorade positionen till sådan grad att det begränsade dess navigering. När körningar misslyckades var det nästan uteslutande till följd av mekaniska fel i växlar och inte på grund av brister i positioneringen.

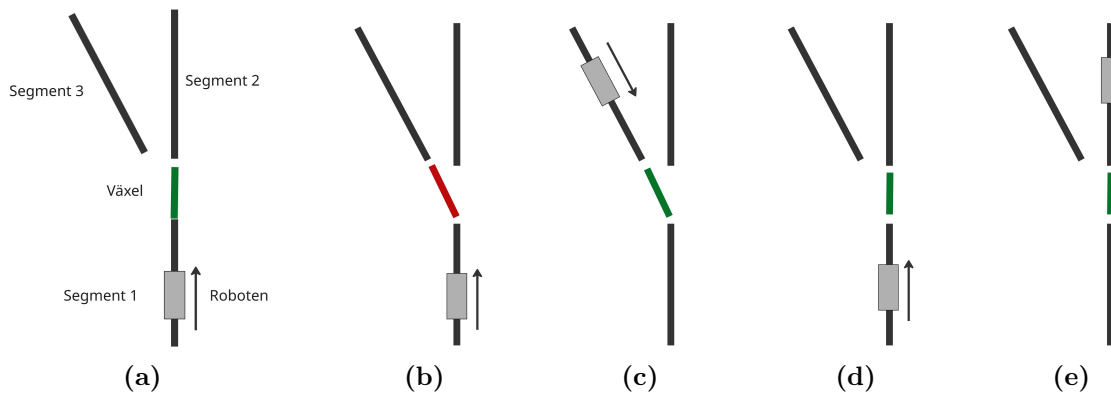
4.3 Realtidsnavigering i rälsnätverket

Roboten klarar av att planera en rutt och korrekt identifiera vilka växlar som behöver ställas om. Genom att begära växling till en trafikledningsserver kan den dessutom ställa om de växlar som identifieras på ett korrekt sätt. Den lyckas även publicera körkommandon till motorstyrningen för att starta, stoppa och byta riktning på motorn.

Genom att kunna identifiera i vilken riktning en växel passeras klarar den av att ställa om växlar som passeras medväxel, se Figur 3.3 i avsnitt 3.4 för förtydligande kring riktning på växlar. Denna typ av kontroll är särskilt viktig för rutter där roboten behöver byta riktning efter en växel. Där är det kritiskt att roboten har hunnit passera förbi medväxeln helt innan den ska begära att växeln ställs om. Vid försök lyckades roboten utföra denna manöver utan några svårigheter och den skickade förfrågan vid korrekt position. För att säkerställa att roboten skickar förfrågningar vid rätt position begränsas den till att

endast få skicka signaler då den är i närheten av mittpunkten av ett givet segment. Med denna begränsning måste roboten helt befinna sig på det nya rälssegmentet innan nya instruktioner kan skickas ut.

För att undersöka robotens förmåga att korrigera felnavigeringar studerades även scenarier där växlar manuellt ställdes om. Detta genomfördes genom att manuellt ändra växlens position efter att robotens förfrågan gått igenom. Detta resulterade i att roboten hamnade utanför sin planerade rutt och den klarade av att självständigt identifiera denna felnavigering och sedan beräkna en ny rutt till målet. För en skiss av förloppet se Figur 4.3.



Figur 4.3: Roboten ska färdas från segment 1 till segment 2 över växeln i **a**. På grund av att växeln ställs fel – se **b** – navigerar roboten in på segment 3, då den upprepade gånger befinner sig på segment 3 identifierar den att det rör sig om en felnavigering. Den avbryter sin tidigare rutt och planerar en ny mellan segment 3 och segment 2. Växlarna ställs om efter den nya ruten i **c** respektive **d** och roboten ankommer på segment 2 i **e**.

Denna typ av lösning där en helt ny rutt planeras är möjlig som en följd av den korta beräkningstiden av en rutt. För rälsnätverket av komplexiteten som används i testbädden – med i storleksordningen 40 rälssegment – klarar algoritmen av att beräkna en rutt på bråkdelen av tiden det tar för roboten att färdas längs rälsnätverket. Det är således inte en begränsning att utföra dessa beräkningar på nytt vid felnavigeringar. Vid större rälsnätverk med fler parametrar hade det varit aktuellt att vidareutveckla algoritmen för att minimera tiden som roboten utför beräkningar ute på rälsen. Vid en fullskalig modell är det dock värt att poängtera att allting sker mycket långsammare, avståndet mellan olika segment kan vara tiotals eller hundratals tåglängder till skillnad från de två som användes i detta projekt. En fullskalig implementation har dock en stor mängd andra komplikationer och det är viktigt att en algoritm tänkt för sådana tillämpningar tar hänsyn till de problemspecifika förutsättningarna.

5

Diskussion

Arbetet har lyckats genomföra de mål som sattes upp för projektet. Den ursprungliga testbädden har utvecklats vidare och har nu nio växlar som kan fjärrstyras med hjälp av ROS och MQTT. Det befintliga kamerasystemet har även integrerats i testbädden genom ett program för att strömma positionsdata över det lokala nätverket. Utöver att utveckla själva testbädden har även den småskaliga roboten vidareutvecklats för att kunna navigera självständigt i rälsnätverket. Detta görs genom att den utför en tidsberoende ruttplanering samt kan begära växlingar.

5.1 Utvecklingen av testbädden och ruttplanering

En stor del av arbetet fokuserade på att utveckla testbädden som i dagsläget innefattar en mer komplex struktur samt ökad funktionalitet. Utöver längre räls och fler växlar, jämfört med den tidigare uppställningen, innehåller det nya rälsnätverket även intressanta specialfall som loopar, återvändsgränder och segment som är kortare än ett normalt tåg. Med denna ökade komplexitet kommer testbädden att ge en god grund för fortsatta studier av rälsgående robotar där man kan studera mer avancerade beteenden.

De växlar som är implementerade i rälsnätverket går att styra autonomt. Genom att dessa är implementerade med hjälp av en tredje part som agerar likt en trafikledare blir systemet både mer modulärt och verklighetstroget då det kapslar in kommunikationen med växlarna i en växlingsserver. En användare kan sedan implementera sin egen logik för att simulera det önskade beteendet hos växlarna. I detta projekt användes en server som hanterade förfrågningar från roboten. Likt en verklig tillämpning skulle en robot inte kunna ställa om växlar själv utan skulle behöva be operatören om växelomställning och sedan invänta svar. Det introducerar även en modularitet som förstärks av möjligheten att köra växlingsrelaterad kod på den dedikerade Raspberry Pi:n som är en del av testbädden. Sammantaget är kommunikationen med växlarna både tillförlitlig och snabb, vilket tillåter roboten att utföra komplexa manövrar där flera växlar behöver ställas om i följd.

Växelmekaniken fungerar pålitligt i det aktuella systemet och har visat sig vara en robust lösning för styrning av ett mindre rälsnätverk. Genom att bevara fjädermekanismen från originalväxeln bibehålls den viktiga funktionen att tåg kan köra medväxel även om växeltungan inte är korrekt ställd. Detta ökar driftsäkerheten och minskar risken för urspårning utan behov av ytterligare återkopplingssystem.

Trots detta uppstod vissa variationer i den mekaniska rörelsen mellan olika växlar. Detta tros bero på små skillnader i tillverkning av delarna, vilket resulterade i för hög friktion mellan den undre armen och lådans botten. I dessa fall kunde inte servon genomföra en

fullständig rörelse, vilket innebar att växeltungan antingen inte bytte läge som avsett eller föll tillbaka när roboten passerade. Problemet kunde lösas genom manuell överarbetning med en fil samt genom att kalibrera servons mittläge innan den undre armen monterades.

Den funktionalitet som utvecklades till roboten syftade framförallt till att komplettera de befintliga system som fanns sedan tidigare. All den kod som är tänkt att köras utöver den kod som funnits på roboten sedan tidigare. Alla ruttberäkningar och inhämtning av position sker i ett separat program som sedan kommunicerar styrkommandon till roboten via ROS. Detta upplägg – likt det för växlarna – ger en stor flexibilitet i hur man väljer att använda systemet. Det gör det exempelvis möjligt att styra en annan robot än den som använts i projektet genom att endast säkerställa att de kommunicerar med samma styrkommandon. Detta är möjligt som en följd av att arbetet använder ROS för sin arkitektur. Detta utnyttjas även för positionsinläsningen där roboten endast behöver prenumerera på positionsdatan utan att bero på hur den datan tagits fram. Med denna struktur kan samma mjukvara användas på MPIR utan modifikation, endast vilken positionsdata som publiceras behöver ändras från LPS till GNSS.

Den del av positioneringen som roboten sköter är att översätta mellan en position i rummet och en position på rälsen. Då detta steg sker separat från själva positioneringen hade det varit möjligt att utföra detta på en separat nod. Den valda strukturen bedömdes dock lämplig för projektets omfattning. Då detta projekt använder en relativt simpel metod för att projicera en position till ett rälssegment, som bygger mycket på att utgå från den rutt som roboten är tänkt att följa, fungerade det väl att detta gjordes direkt i robotens navigeringsnod.

Även om positioneringen fungerade väl för projektets förutsättningar finns det fall som roboten inte klarar av. Då dödräknaren ej är integrerad klarar systemet ej av att positionera roboten då den helt tappar roboten ur sikte. Detta gör att roboten inte skulle kunna navigera i uppställningar som innehåller tunnlar eller döda vinklar på ett tillförlitligt sätt. Det faktum att positioneringen hade varierande precision, se Figur 4.2, kan förklaras med att kameratäckningen varierade över rälsnätverket. Flera olika variationer av kameravinklar testades och den som slutligen användes var den som maximerade precisionen över hela rälsnätverket. Det är möjligt att det med fler kameror, alternativt andra monteringspositioner, går att erhålla bättre täckning. De identifierade områden sammanfaller även med de mer komplexa områdena som innehåller många växlar. Det är möjligt att den ökade tätheten av olika rälssegment i dessa områden skapar ytterligare reflexer som gör det svårare för QTM att korrekt hitta och positionera robotens markörer. Det är även möjligt att flera av markörerna blir skymda av roboten i vissa vinklar och att de därför inte kan ses av nog många kameror för att positioneras.

5.2 Framtida utvecklingsmöjligheter inom projektet

Inom projektet finns det fler saker som skulle kunna vidareutvecklas. Dessa utvecklingar har inte implementerats inom projektet på grund av diverse anledningar, som begränsad tid eller att de ligger utanför ramen för arbetet. Vid är fortsatt utveckling av projektet skulle det fortfarande vara relevanta för en förbättrad funktionalitet och effektivitet hos testbädden.

För att göra testbädden mer verklighetstrogen skulle man kunna införa trafikljus med fem LED-lampor istället för de nuvarande två. Detta skulle bättre efterlikna hur signal-systemet är konstruerat på svenska järnvägar. Där används signalerna för att indikera den högsta tillåtna hastigheten för ett visst spåravsnitt samt om ett tåg får passera en viss punkt. Tillsammans med denna utveckling skulle man även kunna införa hastighetsreglering för att introducera mer komplexa körmönster. Då skulle man kunna gå ifrån att roboten alltid kör i samma förutbestämda hastighet, till att den bestäms utifrån förutsättningar som trafikljusens läge, vägförhållanden och annan trafik.

Framtida iterationer av växlarna kan förbättras genom att öka toleranserna i CAD-modellerna mellan de ytor där friktionen är ett problem. Detta för att slippa den manuella bearbetningen. Ytterligare en förbättringsmöjlighet är att implementera ett återkopplingssystem för att kunna verifiera en växels faktiska läge. Detta skulle kunna göras med en gränslägesbrytare, Halleffektsensor eller annan typ av sensor monterad vid servon eller växeltungan. Med återkoppling från växeln kan systemet i realtid upptäcka felinställningar, vilket är särskilt relevant vid utbyggnad till mer komplexa och automatiserade nätverk där manuell översyn inte är möjlig.

En ytterligare förbättring vore att införa ett mer användarvänligt sätt att manuellt styra växlarna. I dagsläget krävs det att användaren skickar MQTT-meddelanden via en terminal, vilket hämmar användarvänligheten då det kräver att man har upprättat en anslutning till MQTT-servern. Genom att implementera fysiska knappar vid varje växel eller ett enkelt grafiskt gränssnitt skulle växlarna kunna styras med ett knapptryck. Detta skulle inte bara förenkla manuell kontroll och felsökning, utan även möjliggöra avsiktlig felställning av växlar för att testa robotens förmåga att upptäcka och hantera sådana scenarier. På detta sätt kan systemet användas för att utveckla och verifiera mer avancerade funktioner för autonom felsökning i framtida versioner.

Inom projektet är robotens lokalisering en central del, vilket har genomförts med hjälp av ett LPS baserat på Qualisys ProReflex kameror. Systemet fungerar väl under förutsättning att fri sikt mellan kameror och robot upprätthålls. Ett återkommande problem uppstår dock när roboten rör sig genom tunnlar eller andra miljöer där sikten blockeras – något som försvårar eller omöjliggör positioneringen. Detta problem skulle kunna lösas genom att inkludera en dödräknare som räknar antal varv hjulen har roterat och utifrån det approximerar robotens position. En start på utveckling av dödräknaren gjordes i projektet men den skulle behöva utvecklas vidare för att fullt integreras tillsammans med LPS och navigeringsnoden. Ytterligare, skulle man kunna vidareutveckla rälsens komplexitet för att på så sätt utmana algoritmen. Detta skulle kunna göras genom att inkludera trevägskorsningar, x-korsningar eller öka antal växlar. Man kan också inkludera uppför- och nerförsbackar som hade påverkat robotens hastighet och därmed tiden de olika segmenten tar att köra igenom.

Det finns också flera saker som skulle kunna implementeras för att utveckla algoritmens funktionalitet och vilka tester som kan genomföras med den. Dessa skulle kunna inkludera att man kan ha flera robotar som kör tillsammans och navigerar simultant utan att krocka. Algoritmen skulle även kunna ta emot flera punkter och planera den snabbaste rutten som besöker dem alla. Sedan för att utveckla systemet så att de liknar det som är planerat för MPIR skulle man kunna implementera en separat flygande inspektionsrobot. Roboten skulle då endast behöva köra till ett närliggande segment i närheten av den punkt som

ska inspekteras, varpå den flygande enheten kan skickas ut för att undersöka rälsen efter eventuella fel.

6

Slutsatser

Projektet har visat att det är möjligt att implementera en testbädd som tillåter testning av autonom navigering som bygger på ett lokalt positioneringssystem. Hela testbädden grundar sig på kommersiellt tillgänglig räls och ett komplett kamerasystem med tillhörande mjukvara. Dessa system har sammanfogats med ny mjukvara och kompletterats med såväl egenutvecklade växlar som ruttplaneringsalgoritmer, allt för att skapa en mångsidig testbädd.

Att alla dessa olika system och komponenter lyckades föras samman för att skapa en enhetlig testbädd är en av projektets största framgångar. Arbetet har krävt flera iterationer och en stor mängd testning där olika arkitekturer har prövats. Det finns många utvecklingsområden för nya projekt, såväl nya funktionaliteter som förbättring av de befintliga. Det finns möjlighet att öka tillförlitligheten genom att vidareutveckla växlarna, robotens beslutsfattning och ruttplaneringen. Samtidigt är det viktigt att poängtera att den nuvarande implementeringen fungerar tillfredsställande i alla dessa avseenden.

Med den resulterande testbädden är det förhoppningen att det blir lättare att testa och utvärdera den skalmodell som utvecklas på Chalmers. Med en mer komplex och verklighetstrogen testbädd minimeras antalet tester som behöver göras på den fullskaliga modellen i Västerås och det gör att framtida projekt kan spara både tid och resurser. Med den inbyggda modulariteten är även förhoppningen att de olika systemen som implementeras ska kunna användas i många olika applikationer och vara en språngbräda för framtida projekt.

Litteratur

- [1] *Europe's Rail Flagship Project 3 – Holistic and Integrated Asset Management for Europe's RAIL System (FP3-IAM4RAIL)*. Grant Agreement ID: 101101966. Europe's Rail Joint Undertaking, dec. 2022. DOI: 10.3030/101101966. URL: <https://projects.rail-research.europa.eu/eurail-fp3/structure/>.
- [2] Chalmers. *Självkörande järnvägsrobot banar väg för stora samhällsvinster*. 2022. URL: <https://www.chalmers.se/aktuellt/nyheter/m2-sjalvkorande-jarnvagsrobot-banar-vag-for-stora-samhallsvinster/> (hämtad 2025-02-12).
- [3] Johan Berglund. “Analys av olika positioneringssystem för Trafikförvaltningens järnvägsbanor”. Examensarb. KTH, Health Informatics och Logistics, 2019, s. 42. URL: <https://www.diva-portal.org/smash/get/diva2:1353874/FULLTEXT01.pdf>.
- [4] Trafikverket. “Tekniken bakom Tågkartan”. I: (febr. 2024). URL: <https://www.trafikverket.se/e-tjanster/tagkartan--ta-reda-pa-var-taget-ar/>.
- [5] Trafikverket. “ATC – tågskyddssystem”. I: (dec. 2023). URL: <https://bransch.trafikverket.se/for-dig-i-branschen/teknik/anlaggningsteknik/signalteknik/atc--tagsskyddssystem/>.
- [6] Trafikverket. “Så fungerar det nya signalsystemet”. I: (dec. 2023). URL: <https://bransch.trafikverket.se/for-dig-i-branschen/teknik/ett-nytt-digitalt-signalsystem-for-jarnvagen/sa-fungerar-det-nya-signalsystemet/>.
- [7] D Huisman G Budai och R Dekker. “Scheduling preventive railway maintenance activities”. I: *Journal of the Operational Research Society* 57.9 (2006), s. 1035–1044. DOI: 10.1057/palgrave.jors.2602085. eprint: <https://doi.org/10.1057/palgrave.jors.2602085>. URL: <https://doi.org/10.1057/palgrave.jors.2602085>.
- [8] Randika K. W. Vithanage, Colin S. Harrison och Anjali K. M. DeSilva. “Importance and Applications of Robotic and Autonomous Systems (RAS) in Railway Maintenance Sector: A Review”. I: *Computers* 8.3 (2019). ISSN: 2073-431X. DOI: 10.3390/computers8030056. URL: <https://www.mdpi.com/2073-431X/8/3/56>.
- [9] Sophia Achieng m. fl. “Replacing Humans with Machines: Threats and Opportunities”. I: *East African Journal of Business and Economics* 1.2 (sept. 2019), s. 54–68. URL: <https://www.journals.eanso.org/index.php/eajbe/article/view/80>.
- [10] Alexander Hevelke och Julian Nida-Rümelin. “Responsibility for Crashes of Autonomous Vehicles: An Ethical Analysis”. I: *Science and Engineering Ethics* 21.3 (2015), s. 619–630. ISSN: 1471-5546. DOI: 10.1007/s11948-014-9565-5. URL: <https://doi.org/10.1007/s11948-014-9565-5>.

- [11] Raspberry Pi Foundation. *Raspberry Pi for Home Use*. 2025. URL: <https://www.raspberrypi.com/for-home/> (hämtad 2025-04-22).
- [12] Vad är en mikrokontroller, drift och typer. 2022. URL: <https://www.arduino.cc/en/Guide/Introduction> (hämtad 2025-05-06).
- [13] Open Source Robotics Foundation. *Why ROS?* 2021. URL: <https://www.ros.org/blog/why-ros/> (hämtad 2025-04-02).
- [14] MQTT. *MQTT - The Standard for IoT Messaging*. 2024. URL: <https://mqtt.org/> (hämtad 2025-04-02).
- [15] Gernot Veit Batz och Peter Sanders. "Time-Dependent Route Planning with Generalized Objective Functions". I: *Algorithms – ESA 2012*. Utg. av Leah Epstein och Paolo Ferragina. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, s. 169–180. ISBN: 978-3-642-33090-2.
- [16] Yajun Yang m. fl. "Finding the cost-optimal path with time constraint over time-dependent graphs". I: *Proc. VLDB Endow.* 7.9 (maj 2014), s. 673–684. ISSN: 2150-8097. DOI: 10.14778/2732939.2732941. URL: <https://doi.org/10.14778/2732939.2732941>.
- [17] Amit Patel. *A* Search Algorithm: A Comparison of A* and Other Pathfinding Algorithms*. 1997. URL: <https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html> (hämtad 2025-02-13).
- [18] Qualisys. *Qualisys Corporate Booklet 2021*. 2021. URL: https://cdn-content.qualisys.com/2021/04/Qualisys_Corporate_Booklet-2021.pdf (hämtad 2025-01-29).
- [19] Joakim Odengård. "Positioning systems for scale model cars - A survey of available systems and an application with Qualisys Track Manager". I: (2019). URL: <https://research.ebsco.com/linkprocessor/plink?id=857bbb68-a43e-3859-8c88-6e8f4b4bee45>.
- [20] Qualisys. *Qualisys SDK for Python*. URL: https://github.com/qualisys/qualisys_python_sdk.



CHALMERS