CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Utveckling samt evaluering av lokalisering och kooperativa styrsystem

Kandidatarbete vid institutionen för data- och informationsteknik

Tobias Malmentun
Emil Nylander
Tobias Reinerson
Elias Samuelsson
Elias Svensson
John Viberud

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige juni 2025

Kandidatarbete 2025

Utveckling samt evaluering av lokalisering och kooperativa styrsystem

Tobias Malmentun
Emil Nylander
Tobias Reinerson
Elias Samuelsson
Elias Svensson
John Viberud

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för data- och informationsteknik
Chalmers tekniska högskola
Göteborgs universitet
Göteborg, Sverige 2025

Utveckling samt evaluering av lokalisering och kooperativa styrsystem
Tobias Malmentun Emil Nylander Tobias Reinerson Elias Samuelsson Elias Svensson
John Viberud

Handledare: Elad Schiller, institutionen för data- och informationsteknik
Medrättande lärare: Roger Johansson, institutionen för data- och informationsteknik
Examinator: Arne Linde, institutionen för data- och informationsteknik

Kandidatarbete 2025
Institutionen för data- och informationsteknik
Chalmers tekniska högskola
Göteborgs Universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: Bild av laboratoriets testbädd.

Typeset in LaTeX
Göteborg, Sverige 2025

Utveckling samt evaluering av lokalisering och kooperativa styrsystem
Tobias Malmentun, Emil Nylander, Tobias Reinerson, Elias Samuelsson, Elias Svensson, John Viberud

Institutionen för data- och informationsteknik
Chalmers tekniska högskola och Göteborgs universitet

Abstract

This report covers a bachelor's thesis at Chalmers University of Technology. The purpose of this project was to further develop systems for localization and driving of autonomous vehicles inside of a lab environment building upon previous projects. Improvements were made in three distinct areas. The first being the indoor localization system, called GulliView, which consists of four ceiling-mounted cameras using Apriltags as calibration and detection of vehicles. GulliView was improved by implementing an efficient undistortion algorithm to counteract the camera's distortion while maintaining accuracy. As well as creating a unified global coordination system using the world position in meters. Secondly, additional general maneuvers were implemented to autonomous robots using GulliView for positioning. The added maneuvers handle common traffic situations such as intersections, highway merging and roundabouts. Lastly, an advancement was made on implementing Sensor Fusion between a vehicle, which has a previously developed internal positioning system, and GulliView. The vehicle integrates the internal position and the external position given from GulliView.
GulliView attained a median delay decrease of 86%, going from 102 ms to 14.0 ms in time per execution cycle. Meanwhile, the median frequency increased from 10.5 Hz to 16.6 Hz. Improvements on GulliView's positioning accuracy were also observed, going from discrepancies of 8-18% to 1-2%. For the autonomous vehicles, the added maneuvers added only an average of 24.32% increased waiting time in traffic scenarios while maintaing safety. Finally, the fusion of internal and external values resulted in a positioning discrepancy of 5%. These results prove promising and may greatly help further development of all three systems in future projects.

Keywords: autonomous scaled vehicles, indoor localization, cooperative maneuvers, sensorfusion, gulliview, wifibot, turtlebot

Utveckling samt evaluering av lokalisering och kooperativa styrsystem
Tobias Malmentun, Emil Nylander, Tobias Reinerson, Elias Samuelsson, Elias Svensson, John Viberud

Institutionen för data- och informationsteknik
Chalmers tekniska högskola och Göteborgs universitet

Sammandrag

Denna rapport beskriver ett kandidatarbete vid Chalmers tekniska högskola. Syftet med detta projekt var att vidareutveckla system för lokalisering och manövrering av autonoma fordon i en laboratoriemiljö, med utgångspunkt i tidigare projekt. Förbättringar genomfördes inom tre distinkta områden. Det första området handlar om inomhus lokaliseringssystemet GulliView, som i laboratoriet består av fyra takmonterade kameror, som kan positionera markerade fordon inom dess synfält. Systemet utvecklades genom implementering av en effektiv distorsionskorrigerings algoritm, som motverkar kamerornas förvrängning medans precisionen upprätthålls. Vidare har även ett sammanställt globalt koordineringssystem skapats baserat på den verkliga positionen i meter. Den andra delen av projektet har handlat om utvecklingen av ett standardiserat kooperativt manövreringssystem som implementerats i robotar och utgått från positioneringen från GulliView. Systemet kan hantera olika trafiksituationer så som korsningar, sammanvävning av körfält och rondeller. Slutligen består projektets sista del av en förbättrad positionering genom implementering av sensorfusion mellan ett fordon, som har ett tidigare utvecklat internt positionssystem, och GulliView.
Lokaliseringssystemet GulliView uppnådde en minskning av medianfördröjningen med 86%, där exekveringscykeltiden gick från 102 till 14 ms. Samtidigt ökade medianfrekvensen från 10,5 Hz till 16,6 Hz. Förbättringar i GulliViews positionsnoggrannhet kunde också observeras, där avvikelserna minskade från 8–18% till 1–2%. För de autonoma fordonen innebar det tillagda manövreringssystemet endast en genomsnittlig fördröjning i olika trafiksituationer på 24.32%, samtidigt som säkerheten upprätthölls. Slutligen resulterade sensorfusionen av interna och externa värden i en positionsavvikelse på 5%. Dessa resultaten är lovande och kan i hög grad bidra till en grund för fortsatt vidareutveckling i alla tre områden för framtida projekt.

Keywords: autonoma skalade fordon, lokalisering, kooperativa manövrar, Sensorfusion

Tackord

Vi vill tacka vår handledare Elad Schiller som givit oss värdefulla råd och ett hjälpsamt stöd under projektets gång. Med hans angelägenhet i vårat arbete har han hjälpt oss uppnå en högre potential och givit oss viktig kunskap kring hur arbetsprocessen i denna typ av projektarbete kan struktureras och genomföras på ett bra vis.

Vi vill även tacka Beichen Xie för hans viktiga bidrag till utvecklingen av GulliView. Han har varit mycket hjälpsam både när det gäller att dela med sig av sin kunskap angående systemet samt hitta lösningar. Vi är även tacksamma för Ruohan Li och Yuchuan Dong och deras hjälpsamma förklaring av hur deras GulliView Noetic system fungerar och hur vi kan applicera det.

Slutligen vill vi varmt tacka Maximilian Ygdell för hans ovärderliga stöd med Turtle-Bot 4. Trots att han nu är i arbetslivet ställde han upp och hjälpte oss komma igång, genom att snabbt besvara våra frågor via e-post. Hans insats underlättade projektets start och bidrog till dess framgång.

Tobias Malmentun, Emil Nylander, Tobias Reinerson, Elias Samuelsson, Elias Svensson, John Viberud, Göteborg, juni 2025

Ordlista

AprilTag Tvådimensionell figur skapad för att upptäckas lätt av kameror likt QR-koder.

Create 3 En mobil robotplattform för utbildning och utvecklare skapad av iRobot®
CRI Utvärderingskriterie för säkerhet, se avsnitt 2.11

distorsionskorrigering Metod för att motarbeta en förvrängd bild. Används ofta för att motarbeta en kameras förvrängning på bilden på grund av linsen.

EKF Extended Kalman Filter, är ett icke-linjär filter som genom lokal linjärisering kombinerar modellprediktion och mätningar för att estimera ett systems tillstånd, se avsnitt 2.10

GulliView Ett kamerasystem för positionering av robotarna på testbädden i laboratoriet, se avsnitt 2.3, tillgängligt på BitBucket.
GulliView logs Ett Python script skapat för att visualisera datan från GulliView loggarna, tillgängligt på BitBucket.
GulliView noetic Ett relaterat projekt till GulliView skapat av en grupp studenter i kursen "DAT295/DIT669 Autonomous and Cooperative Vehicular Systems", tillgängligt på BitBucket.

Hermes Ett system för att i realtid kartlägga vägnät. Utvecklades i ett kandidatarbete 2024.

LiDAR Light Detection And Ranging.

OpenCV Open Source Computer Vision Library.

ROS Robot Operating System, se avsnitt 2.4.

SLAM Simultaneous Localization And Mapping.
SSH Secure Shell, används för att koppla upp mot andra datorer på ett lokalt nätverk.

TM Modifierad TTC, se avsnitt 4.2.2
TTC Tid innan kollision inträffar, se avsnitt 2.11
TurtleBot Se avsnitt 2.8.

UDP User Datagram Protocol, nätverksprotokoll för att skicka datagram över ett IP-nätverk.

WifiBot Se avsnitt 2.5.

Innehåll

Figurer **xv**

Tabeller **xvii**

1 Introduktion **1**
1.1 Syfte . . . 1
1.2 Problemdefinition . . . 2
1.3 Testbädden och avgränsningar . . . 2
1.4 Relaterat arbete . . . 3
1.5 Vårt tillvägagångssätt . . . 3
1.6 Vårt bidrag . . . 4
1.7 Validering . . . 4

2 Bakgrund **5**
2.1 Nivåer av automatisering . . . 5
2.2 Laboratoriet . . . 6
2.3 GulliView . . . 7
2.4 ROS . . . 8
2.5 WiFiBot . . . 8
2.6 Tidigare kooperativa system . . . 9
2.7 SLAM . . . 10
2.8 TurtleBot . . . 10
2.9 Sensor fusion . . . 11
2.10 Extended Kalman Filter(EKF) . . . 11
2.11 Evalueringskriteriet CRI . . . 11

3 Metod **13**
3.1 GulliView . . . 13
3.1.1 Effektivare distorsionskorrigering . . . 13
3.1.2 Globalt koordinatsystem . . . 14
3.1.3 Omstrukturering av koden . . . 18
3.1.4 Förbättrad evaluering . . . 18
3.2 Distribuerade kooperativa manövreringssystem . . . 18
3.2.1 Distribuerad manövrering för överlappande trafik . . . 19
3.2.2 Kommunikationsprotokoll för kooperativ manövrering . . . 20
3.2.3 Trafikimplementationer . . . 22

3.3 Integrering av intern- och extern lokalisering . . . 24
3.3.1 Uppdatering och utveckling av mjuk- och hårdvara . . . 24
3.3.2 Implementering av extern lokalisering i äldre GulliView . . . 25
3.3.3 Integration för nyare GulliView-system med globalt koordinatsystem 27
3.3.4 Fusion av sensordata med utökat Kalmanfilter . . . 28

4 Evaluering **31**
4.1 Evaluering av inomhuslokalisering . . . 32
4.2 Evaluering av kooperativ manövrering . . . 32
4.2.1 Trafikflöde . . . 33
4.2.2 Säkerhet . . . 35
4.3 Jämförelse av intern och extern lokalisering mot fysiska referenser . . . 36
4.3.1 Indirekt feluppskattning genom jämförelse med SLAM . . . 36
4.3.2 Jämförelse mot verkliga mått . . . 37

5 Resultat **39**
5.1 GulliViews hastighet . . . 39
5.2 Kooperativ manövrering . . . 41
5.3 Avvikelser mellan intern och extern lokalisering . . . 43

6 Diskussion **47**
6.1 Stabilisering och strukturering av GulliView . . . 47
6.1.1 Slutsats GulliView . . . 48
6.2 Kooperativ manövrering . . . 48
6.2.1 Slutsats kooperativa manövrar . . . 49
6.3 Integrering av intern och extern lokalisering . . . 49
6.3.1 GulliViews potential för extern lokalisering . . . 49
6.3.2 Slutsats intern och extern lokalisering . . . 50
6.4 Teknikens möjligheter och begränsningar i samhället . . . 50
6.5 Framtida arbete . . . 51
6.5.1 GulliView . . . 51
6.5.2 Kooperativ manövrering . . . 51
6.5.3 Integrering av intern- och extern lokalisering . . . 52

A Mätdata från evaluering för intern och extern lokalisering **I**

B Kollisionsförebyggande hastighetsreglering **III**

C Källkod - Github - Integrering av intern- och externa lokaliserings system - TurtleBot4 **VII**

D Källkod - Github - Kooperativa manövrar - WiFiBot **IX**

E Källkod - Bitbucket - Laboratoriets samtliga projekt **XI**

F GulliViews nya kodstruktur **XIII**

Figurer

2.1 Bild av testbädden i laboratoriet. 6
2.2 Två av de fyra Kameror monterade i taket inkopplade till datorn till vänster. 7
2.3 Bild av en WiFiBot. 9
2.4 Bild av TurtleBot 4 med AprilTag. 10
2.5 Trafiksituation där fordon C ska sammanväva mellan fordon A och B. . . 12

3.1 Distorsionskorrigering av hörnen. De röda punkterna indikerar de punkter som Apriltagdetektorn detekterade i den förvrängda bilden och de blåa punkterna indikerar de punkterna efter distorsionskorrigering på verklighetsenlig bild. 14
3.2 Grafisk representation av alla fyra Apriltags som en kamera ser. 15
3.3 Flödesschema av manövreringsalgoritmen för exklusiva zoner. 21
3.4 Fyrvägskorsning med exklusiv zon uppdelad i fyra delområden. 22
3.5 Fyrvägskorsning med exklusiva zoner och robotar som kommer passera varandra. 22
3.6 Fyrvägskorsning med exklusiva zoner och robotar som kommer korsa varandra. 22
3.7 Sammanvävning av körfält med två robotar vars körriktning korsas. . . . 23
3.8 Sammanvävning av körfält med två robotar och ut placerad exklusiv zon. 23
3.9 Cirkulationsplats med två robotar och dess körbana. 23
3.10 Cirkulationsplats med två robotar och exklusiva zoner. 23
3.11 Kartläggning med Hermes: den ljusgrå bakgrundskartan och de svarta konturerna visar den uppbyggda miljön, medan det färgade punktmolnet avspeglar sensorernas realtidsdata. Bilden illustrerar hur TurtleBotens odometri (dödräkning) snabbt börjar avvika från verkligheten. 24

4.1 Numrerade startpositioner och färdvägar genom korsningen. 34

5.1 Låddiagram över fördröjning av en exekveringscykel av Fast Thread från att bilden tags tills meddelande skickas. 39
5.2 En inzoomad variant av låddiagram från data som representeras i figuren 5.1, för ökad detaljrikedom. 39
5.3 Låddaiagram över uppdateringsfrekvens av Fast Thread mätt genom antalet exekveringscykler per sekund. 40
5.4 En inzoomad variant av låddiagrammet som representeras i figur 5.3, för ökad detaljrikedom. 40
5.5 Diagram över beräknade värden av FK för varje evalueringsscenario. . . . 41

5.6 Genomsnittliga värden av CRI för alla scenarion av sammanvävning. . . 42
5.7 Procentuell avvikelse för GulliView från 2023. Segment 1 och 3 motsvarar sträckor på 6.0 m, medan segment 2 och 4 är 2.5 m långa. 43
5.8 Procentuell avvikelse för GulliView från 2025. Segment 1 och 3 motsvarar sträckor på 6.0 m, medan segment 2 och 4 är 2.5 m långa. 44
5.9 Plot som visar vägen TurtleBot tar då SLAM är referense datan och gamla GulliView(GV) är transformerad för att passa SLAM:s data. 45
5.10 Histogram som visar skillnad i distans mellan punkter av SLAM och gamla GulliView, data baserad på figur (5.9) 45
5.11 Plot som visar vägen turtlebot tar då SLAM är referense datan och nya GulliView(GV) är transformerad att passa SLAM:s data. 46
5.12 Histogram som visar skillnad i distans mellan punkter av SLAM och nya GulliView, data baserad på figur (5.11) 46

B.1 Representation av WiFiBotarna med vektorer. III
B.2 Representation av WiFiBotarna med vektorer för kollisions beräkning. . . III
B.3 Representation av WiFiBotarna med vektorer för avstånds beräkning. . . IV

Tabeller

3.1 Kalibreringstaggarnas position i rummet med den vänstra taggen längst in i rummet som referenspunkt. 17

5.1 Jämförelse av fördröjning av Fast Thread från att bilden tas tills koordinater är redo att skickas mellan gamla och nya GulliView. 40
5.2 Jämförelse av uppdateringsfrekvens av Fast Thread mellan gamla och nya GulliView. 40
5.3 Beräknade värden för medelvärdena av FK för de olika trafiksituationerna. 42

1 Mätvärden (överst) och fel (underst) för gamla GulliView (GV) och SLAM per segment (med angiven distans) för tester 1–3 samt deras medelvärden. I
2 Mätvärden (överst) och fel (underst) för nya GulliView (GV) och SLAM per segment (med angiven distans) för tester 4–6 samt deras medelvärden. I

1

Introduktion

Transportsektorn genomgår en revolution i form av utvecklingen av autonom körning som medför stor potential gällande bland annat trafikflöde och säkerhet [1]. I ett laboratorium på Chalmers tekniska högskola har det under flera års tid drivits projekt av studenter som utforskat möjligheten till implementationen av system för lokalisering och styrning av autonoma fordon. I början av detta arbete existerade inget system för att studera autonom körning som möter behoven av hög precision i lokaliseringen vid framförandet av robotarna. Dessutom saknades ett standardiserat system för kooperativa manövrar som kan appliceras i flera trafiksituationer. Det har tidigare genomförts arbeten på att skapa dessa system som inte uppnått tillräckligt god säkerhet i lokaliseringen för detta projektets syfte. Med hjälp av både sensorer ombord på robotplattformarna och ett yttre centralt lokaliseringssystem ämnar detta projekt till att utveckla dessa system för bättre lokalisering och koordinering. Arbetet målsätter sig därmed i att stödja och underlätta framtida projekt i området av autonom körning på Chalmers tekniska högskola. Rapporten presenterar nya metoder som har introducerats till laboratoriet och lyckats uppfylla behoven för framtida utveckling i området av autonom körning med kooperativ manövrering.

1.1 Syfte

Detta kandidatarbete utgör en central del av utbildningen på Chalmers då det ger studenter möjlighet att tillämpa sina kunskaper i en kontext som liknar arbetslivet. Genom grupparbete, problemlösning och leverans av ett konkret resultat skapas förutsättningar för att utveckla både teknisk kompetens och viktiga generella färdigheter såsom samarbete, planering och kommunikation. Dessa förmågor är avgörande i det framtida yrkeslivet och bidrar till studenternas "utbildning och livslångt lärande" [2]. Kandidatarbetet är även ett obligatoriskt moment för att kunna ta ut en kandidatexamen och påbörja studier på masternivå vid Chalmers.

Projektet syftar specifikt till att vidareutveckla befintliga system för autonoma och kooperativa fordon, med fokus på att stödja och underlätta framtida projekt i detta område. Genom att bygga vidare på tidigare kandidatarbeten och projekt som genomförts i laboratoriet är målet att förbättra lokaliserings- och styrsystemet för skalmodeller av fordon i en gemensam testmiljö.

1.2 Problemdefinition

Projektet bemöter de nya behoven i laboratoriet genom att undersöka metoder på hur systemen för lokalisering och styrning av robotarnas kooperativa manövrar kan förbättras. Detta problem kommer att studeras utifrån följande två övergripande frågeställningar:

F-1 *Hur kan lokaliseringen av robotarna förbättras för att uppfylla säkerhetskraven vid autonom körning?*

F-2 *Kan en distribuerad manövreringsalgoritm implementeras på ett standardiserat sätt för att möta säkerhetskraven i olika trafiksituationer?*

Dessa frågeställningar är relevanta för utvecklingen av laboratoriets möjligheter att utforska autonom körning. Genom att förbättra inomhuslokaliseringen kan säkerheten i olika trafiksituationer utvärderas på ett mer realistiskt sätt och i bättre detalj. Det tillåter att tester på implementationen av autonom körning kan utvärderas så att det bidrar till större förståelse av de ingående aspekterna i området, såsom säkerhet och trafikflöde. Vidare ligger intresset i att implementera ett kooperativt manövreringssystem i att bidra med ett verktyg som kan skapa verklighetstrogna trafikscenarion i laboratoriet där säkerheten och trafikflödet kan studeras.

1.3 Testbädden och avgränsningar

Projektet utförs i en välutrustad laboratoriemiljö där autonoma fordon testas under olika förhållanden. En del faktorer som inte berörs är vägunderlag, väderpåverkan, höjdskillnader, fotgängare och andra oförutsägbara trafikelement. Syftet med denna laboratoriemiljö är att fokusera på utvecklingen av fordonens styrning, kommunikation och sensorer utan att komplexiteten ökar genom yttre störningar.

Testningen sker med skalade enheter, i detta fall två WifiBotar och två TurtleBotar. Positionering hanteras genom inomhuslokaliseringssystemet GulliView och SLAM från tidigare års kandidatarbete. Ytterligare sensorer och mer kraftfull hårdvara skulle kunna användas för att förbättra precisionen och hastighet. Fokuset i detta projekt ligger dock på att använda SLAM och lokaliseringssystemet GulliView med befintlig hårdvara för att undvika onödig komplexitet och minimera kostnader. Kommunikationen mellan fordonen hålls till testmiljön, vilket innebär att ingen interaktion med externa system såsom trafikljus eller annan infrastruktur inkluderas.

Dessa avgränsningar säkerställer att projektet hålls genomförbart inom den givna tidsramen och att arbetet kan koncentreras på de tekniska aspekter som är mest relevanta för syftet och problembeskrivningen.

1.4 Relaterat arbete

Arbetet som presenteras i denna rapport bygger vidare på tidigare utvecklade system i laboratoriet. Lokaliseringssystemet utvecklas från ett projekt som syftade till att öka prestandan på GulliView genom parallellprogrammering och implementerade ett fristående system med stöd för global koordinering [3]. Detta system utvecklas med principer i bildanalys med stöd från OpenCVs bibliotek [4]. För att studera kooperativ manövrering utvecklas en algoritm baserat på en konferensartikel av Savic et. al [5], och implementationen i WifiBotarna tar inspiration från ett tidigare projekt som studerade körning vid sammanvävning av körfält [6]. Styrsystemet i WifiBotarna återanvänds från tidigare projekt [7], med små förändringar. Denna implementation av kooperativ manövreringar utvärderas med ett riskkriterium [8], som i tidigare undersökningar i laboratoriet visats ha goda förmågor att studera kollisionsrisk [9]. Arbetet med robotplattformen TurtleBot grundar sig i ett tidigare projekt[10] som introducerade användningen av roboten till laboratoriet. Integrering av extern lokalisering vägleddes inledningsvis av [11]. Dokumentationssidorna till ROS[12] och TurtleBot[13] har också varit värdefulla under projektets inlärnings- och utvecklingsfaser.

Flera aspekter av detta projekt har även koppling till tidigare forskning inom distribuerad styrning och koordinerad rörelseplanering. Forskning kring kollisionundvikning i närvaro av kommunikationsfel, som utöver den tidigare nämnda Savic et al. [5] finns det liknande forskning på metoder i området som Morales-Ponce et al. [14, 15] presenterar. Denna forskning är särskilt relevant för vår kooperativ manövrering med WifiBotar. Vidare är arbetet av Petig et al. [16] om säkerhet i filbyten och trafikkoordination av intresse för rörelseplanering i trånga miljöer. Gulliver-testbädden [17, 18] erbjuder ett systemperspektiv som överlappar med detta arbete i att kombinera lokal sensorinformation med global koordinering. Slutligen tillför KARYON-projektet [19] viktiga principer för säker och förutsägbar samordning, särskilt i system som bygger på samarbete mellan autonoma enheter.

1.5 Vårt tillvägagångssätt

I detta arbete studerar vi laboratoriets helhet utifrån dess olika delsystem. Lokaliseringssystemet GulliView och robotplattformarna WifiBot och TurtleBot bidrar alla med olika funktioner i laboratoriet. GulliView är grunden för systemet och bidrar med möjligheten att ge positionsdata till robotarna. Robotplattformarna WifiBot och TurtleBot bidrar med funktioner som har olika fördelar. TurtleBot är den nyare plattformen och bygger på ROS2 som möjliggör kontrollerad realtids navigering medan den enklare plattformen WifiBot bygger på ROS1 och styrs mer likt ett verkligt fordon. Dessa delsystem bidrar till en helhet, där tillsammans med laboratoriemiljön kan autonom styrning av robotarna undersökas. Vi svarar på de två forskningsfrågorna i avsnitt 1.2 genom att förbättra lokaliseringen i laboratoriet genom GulliView och LiDAR på TurtleBot, samt implementerar vi en ny algoritm för kooperativ manövrering.

1.6 Vårt bidrag

Detta projekt angriper frågor kring att uppfylla säkerhetskrav för fordon där människor kan sättas i fara. För att uppnå detta har fokus varit på att förbättra befintliga system så de är mer robusta med bättre noggrannhet och kortare fördröjning. Arbetet har resulterat i tre huvudsakliga tekniska bidrag som utgör en solid grund för fortsatt utveckling i Chalmers laboratoriemiljö för autonoma fordon.

Inomhuslokaliseringssystemet GulliView behöver vara snabbt och precist. Utveckling i systemet har bidragit med att fördröjningen från då att en bild är tagen tills då att robotarna har möjlighet att ta emot positionsdata, har minskats med 86%. I kombination med ett globalt koordinatsystem som har implementerats gör GulliView väldigt användbart för laboratoriets robotar. En annan del av projektet har utvecklat en ny distribuerad standardiserad manövreringsalgoritm som kan hantera rondeller, korsningar och sammanvävningar av körfält. Detta leder till att robotarna kan hantera en större bredd av trafiksituationer med större flexibilitet och ökat trafikflöde, samtidigt som säkerheten bevarats. En sista del av projektet har arbetat med att kombinera positionsdata från GulliView med SLAM som resulterar i en ökad noggrannhet för positioneringen. Dessa förbättringar bidrar med utveckling av laboratoriet mot ett gemensamt mål, att bidra med system som kan garantera säkerheten för människor i trafiken och tillse fortsatt utveckling av laboratoriets system i framtiden.

1.7 Validering

Systemen undersöks för att säkerställa att projektets syfte uppnåtts genom praktiska experiment i laboratoriet. Stabiliteten av lokaliseringssystemet undersöks genom praktiska experiment där systemets beteende observeras medan positionering utförs för en verklig punkt i laboratoriet. Samtidigt mäts då också tiden det tar att utföra positioneringen. Undersökningen av kooperativa manövreringssystem utförs genom observationer av experiment där en nedskalad variant av situationen studeras med robotar i laboratoriet i olika trafiksituationer. System för kartläggning och informationshantering för positionering undersöks även det genom praktiska experiment, med robotar och befintliga system i laboratoriet. I detta moment studeras positionsdata från sensorer ombord robotarna samt jämförs med det yttre lokaliseringssystemet.

2

Bakgrund

Med de system som finns i laboratoriet kan man undersöka möjligheterna till helt autonoma fordon baserat på många olika metoder och tekniska delsystem. I detta avsnitt presenteras först vad autonom körning innebär i detta arbete för att sedan beskriva det mest relevanta av dessa delsystem.

2.1 Nivåer av automatisering

I detta arbete ligger grunden i att utforska lösningar för ett autonomt manövreringssystem i fordon. För att klargöra vad detta innebär beskriver Society of Automotive Engineers (SAE) nivåer av automatisering på följande vis [20].

- **Nivå 0: ingen förarassistens**
  Denna automationsnivå har ingen automation men kan omfatta säkerhetsassistens såsom varningar och nödbroms.
- **Nivå 1: begränsad förarassistens**
  Till denna nivå räknas funktioner som är till för att hjälpa föraren i vardaglig körning såsom adaptiv farthållare eller körfältscentrering.
- **Nivå 2: partiell automatisering av körning**
  I denna nivå så dras en linje mot föregående nivå då flera assistens funktioner används samtidigt. Exempelvis adaptiv farthållare samtidigt som körfältscentrering, men föraren är fortfarande i kontroll.
- **Nivå 3: betingad automatisering**
  På denna nivå kan fordonet framföra sig självt i ett begränsat antal moment såsom körning i trafikkö eller fickparkering men föraren måste kunna ta över kontroll i alla situationer.
- **Nivå 4: körningsautomatisering**
  I denna nivå så kan fordonet ta sig fram helt självt men vissa oförutsägbara moment kan hindra fordonet. Detta kan exempelvis vara ifall passagerarna vill att fordonet ska färdas över terräng. Föraren behöver ej kunna ta över kontroll av fordonet då det i normala körsituationer ska kunna lösa uppgiften.
- **Nivå 5: full automation**
  I den sista nivån så kan fordonet lösa alla körsituationer i alla omständigheter.

Detta skulle kräva någon form av beslutsfattning i form av en mer komplex artificiell intelligens som inte finns i dagsläget.

Detta arbete fokuserar på lösningar som omfattar problem i nivå 4 för ett system som ska framföra robotarna helt autonomt utan behov av någon förare. Arbetet i detta projekt skiljer sig lite från den generella beskrivningen av automatiserade fordon då det inte är den typiska varianten av automation som undersöks. Med automation i fordon syftas det ofta till att det är fordonet självt som ska uppfatta hela sin omgivning och göra beslut därigenom. I det typiska system som detta projekt behandlar så bedrivs körning i första hand med ett kooperativt system där robotarna styrs genom att samarbeta utifrån positionsdata från ett lokaliseringssystem och kommunikation mellan olika robotar som förmedlar sensordata. Att styra roboten endast med sin egna sensordata sker istället i dessa typer av kooperativa system endast då kommunikationsbrist råder mellan robotarna.

2.2 Laboratoriet

Projektet omfattar praktiska moment som körning av robotar. Av denna anledning genomfördes majoriteten av projektet i ett laboratorium. Denna lokal är ett rum på ca 14 x 5 meter och omfattar en mängd utrustning nödvändig för projektet. Den största delen av laboratoriet består av en testbädd utformad av en svart matta med utritade vägmarkeringar, se figur 2.1. Testbäddens syfte är att simulera en verklig trafiksituation och används för att utför körtest med robotar och kooperativa manövreringar. Inom laboratoriet finns även ett lokalt nätverk vid namn Rostig som möjliggör enkel kommunikation mellan datorer och robotar. Detta gör att information kan delas mellan olika system och möjliggör ett samarbete mellan dem. Över testbädden finns även i taket fyra kameror monterade som ger en överblick över hela testområdet, se figur 2.2. Dessa kameror kan användas för lokalisering av objekt i testområdet. I laboratoriet finns även robotar som kan användas vid utvärdering av olika system. Dessa robotar omfattar bland annat två WifiBotar och två TurtleBotar.

Figur 2.1: Bild av testbädden i laboratoriet.

2.3 GulliView

Lokalisering är ett kritiskt moment för automation av fordon och därför utvecklades GulliView (tillgängligt på BitBucket) som ett lokaliseringssystem för Gulliver projektet [21][22]. Till skillnad från lokalisering utomhus, som ofta förlitar sig på RTK (Real Time Kinematic), sensorer och radar så använder systemet sig utav relativt enkla konsument webbkameror för att lokalisera AprilTags [21]. Det finns fyra Elgato Facecam Pro kameror monterade i taket kopplade till en dator som kan se hela testbädden, se figur 2.2. Dessa kameror är alla kapabla till 4K med en uppdateringsfrekvens på 60Hz [23]. Nuvarande system använder sig endast av Full HD och lyckas inte uppnå maxfrekvensen på kamerorna. [3]. GulliView systemet körs på en Intel NUC som använder sig av en Intel Core i7-1360P för alla beräkningar.

Figur 2.2: Två av de fyra Kameror monterade i taket inkopplade till datorn till vänster.

Ett tidigare projekt har utvecklat ett nytt system som liknar GulliView fast med ett annat tillvägagångssätt som kallas GulliView noetic [3]. Detta nya systemet har en radikalt annorlunda lösning så det är fördelaktigt att behålla gamla systemet för "att stödja och underlätta framtida projekt på Chalmers" (Se avsnitt 1.1). Dock finns det mycket inspiration att hämta från GulliView noetic. Ett exempel på detta är lösningar för ett globalt koordinatsystem där alla kameror använder samma fysiska punkt som origo. Ett globalt koordinatsystem förbättrar systemets användbarhet avsevärt, eftersom mottagaren slipper omvandla data till sitt eget koordinatsystem. Istället kan alla parter använda ett gemensamt system direkt. Tidigare har varje kamera haft ett eget koordinatsystem baserat på pixelkoordinater.

Parallellt med GulliView noetic har också GulliView vidareutvecklats av samma grupp. Resultatet av detta är att koden för programmet nästan fyrdubblats sedan tidigare års kandidatarbete. De stora förbättringar som gjorts är distorsionskorrigering av bilderna och parallellisering av programmet. Distorsionskorrigeringen är viktigt för noggrannhet men tar mycket processorkraft då distorsionskorrigering görs för varje pixel i varje bild. Detta är en av de stora anledningarna till den stora fördröjningen och låga frekvensen. Med parallellisering menas att olika delar av programmet körs på olika processorkärnor istället för att alla skall vänta på sin tur på en. Detta skapar många nya problem med minneshantering och därav krävs en del mer kod. De olika beståndsdelarna som finns är Fast Thread, Nice Thread och Producer Thread som körs i en instans var per kamera.

Producer Thread är tråden som hämtar bilderna från respektive kamera och lägger i en buffert där de andra trådarna som arbetar med samma kamera kan hämta dem. Fast Thread är huvudsakliga tråden som hittar AprilTags, den fungerar genom att den sparar var ett fordon befann sig senaste gången och söker endast igenom närområdet. Denna tråd antar att fordonet bara befinner sig relativt nära positionen där den upptäcktes

senast. Fast Thread kan på så sätt spara mycket tid jämfört med Nice Thread som söker igenom hela bilden för AprilTags som antingen inte upptäckts tidigare eller har rört sig för snabbt så Fast Thread inte riktigt hängt med.

AprilTags är en samling QR-kod liknande koder som skrivs ut på papper och kan enkelt identifieras med hjälp av AprilTags mjukvarubibliotek vilket kan användas i bland annat C++. Biblioteket erbjuder funktioner som gör det möjligt att, genom en AprilTag detektor, detektera AprilTags och ge dess koordinater i bilden [24].

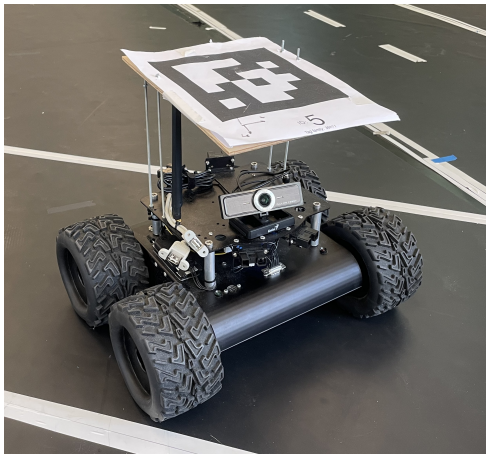
För bearbetning av bilderna används OpenCV, förkortning av Open Source Computer Vision Library, vilket är ett öppet källkods programbibliotek specialiserat i datorseende [4]. Detta gör OpenCV till ett mycket användbart programbibliotek för GulliView där den används för att bearbeta bilderna som skickas från kamerorna. OpenCV stödjer programmeringsspråken C++, Python och MATLAB. Det finns en mängd avancerade funktioner som ingår i OpenCV, varav några av de som kommer att användas i detta projektet är funktioner kring distorsionskorrigering och hantering av bilder. Tack vare att det är öppen källkod är OpenCV under konstant utveckling och därav är dess funktioner väl optimerade.

2.4 ROS

ROS är en uppsättning av mjukvarubibliotek och verktyg för utvecklingen av robotapplikationer. ROS används i allt från drivrutiner till toppmoderna algoritmer, med kraftfulla utvecklarverktyg där allt är öppen källkod [12]. ROS är vad som används för att kontrollera robotarna i detta projekt, mer specifikt används både ROS1 för WifiBot och ROS2 för TurtleBot. Med ROS använder man så kallade noder, som är körbara program skrivna i språk som Python och C++, vars syfte är att utföra en enskild specifik uppgift. Dessa noder kan sedan kommunicera med varandra genom att publicera och prenumerera på så kallade topics. På detta vis kan robotarna styra sina olika delsystem och utföra alla de uppgifter som systemet är utformat för [25].

2.5 WiFiBot

WifiBot är en mobil robotplattform utformad för utbildning, forskning och utveckling [26]. Den är lämplig för studier inom robotik, inbäddade system, artificiell intelligens och djupinlärning. Robotplattformen är utvecklad för att vara anpassningsbar, pålitlig och robust. WifiBotens utformning grundar sig i ett fyrhjulsdrivet chassi med en kamera, en enkortsdator, en WiFi-åtkomstpunkt och en hastighetsmätare. Utöver detta finns det möjlighet att montera många ytterligare sensorer, se figur 2.3.



Figur 2.3: Bild av en WiFiBot.

2.6 Tidigare kooperativa system

Vid tidigare arbeten som omfattat system för manövreringar i laboratoriet har dessa WifiBotar använts. Av denna anledning är många väsentliga funktioner redan implementerade. Robotarna kan kommunicera med en tidigare version av GulliView [6] (äldre än GulliView noetic), via laboratoriets lokala nätverk Rostig. Systemet upptäcker och förmedlar varandras position, hastighet, vinkel och identifieringsnummer (id). ROS noden `gv_socket_server` tar emot denna information och publicerar den på en ROS topic vid namn `gv_positions`. På detta vis kan alla ROS noder på denna robot ta del av information i sitt utövande. Robotarna innehåller även ROS noden `go_to_goal`, vars uppgift är att beräkna styrandet av hastigheten för robotens motorer, i syfte av att navigera längs en förutbestämd bana. Denna bana består av punkter med koordinater baserat på laboratoriets globala koordinatsystem som definierats av GulliView systemet. En konstant hastighet för robotarna kan skickas till denna nod och noden beräknar hur dess motorer behöver anpassa hastigheten för varje motor för att dess vinkel ska rikta sig till nästkommande punkt i den planerade banan. Noden kommer på detta sätt fram till de hastigheter roboten bör hålla för att nå sitt mål. Noden publicerar sedan detta på en ROS topic vid namn `cmd_vel`. En ytterligare ROS nod `wifibot_node` lyssnar på denna topic och justerar motorerna efter dessa hastigheter.

Ett tidigare arbete inom detta projekt [6], implementerade ett system för körning vid sammanvävning av körfält. System baserades på samma infrastruktur som just förklarats men kunde även styra vardera robots totala hastighet. Detta möjliggör en kontroll över ordningen robotarna körde in i sammanvävningsområdet (där vägarna möts) så att kollisioner undveks. Detta gjordes genom att implementera en ytterligare ROS nod på robotarna vid namn `laptop_socket_server`. Denna nod kommunicerar via nätverket Rostig med en extern dator som fattade beslut om robotarnas hastighet. Noden publicerade även på ROS topicen `gv_laptop` som noden `go_to_goal` lyssnade på och justerar sina hastighetsberäkningar efter.

2.7 SLAM

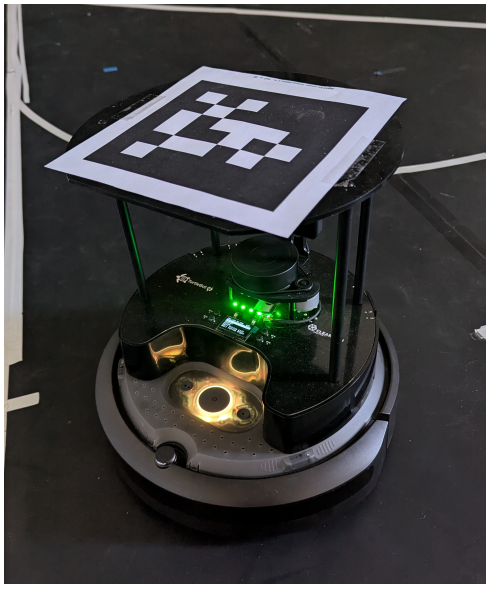
SLAM (Simultaneous Localization and Mapping) är en metod där en robot samtidigt bygger en karta över en okänd miljö och bestämmer sin egen position i kartan [11]. Utmaningen kallas ibland "hönsäggsproblemet": kartan behövs för lokalisering, och lokalisering behövs för kartläggning.

Genom sensorfusion mellan GulliView och SLAM kombineras global positionering från takmonterade kameror med lokal kartläggning från robotens egna sensorer. Detta ger bättre noggrannhet och robusthet, särskilt om något av systemen tillfälligt tappar data.

2.8 TurtleBot

En av robotarna som användes i projektet är TurtleBot 4. Det är en modern och ROS 2-kompatibel mobil robotplattform utvecklad för utbildning och forskning inom robotik[13]. Roboten fungerar som ett tillgängligt men kraftfullt verktyg för att utforska och utveckla autonoma system, särskilt inom områden som SLAM och autonom navigering.

TurtleBot 4 är uppbyggd kring en iRobot Create 3 mobil bas, som förser systemet med bland annat hjuldrift, batteri och odometri [27]. Ovanpå denna bas finns en OAK-D PRO kamera, 2D LiDAR och en Raspberry Pi 4 (RPi4), som fungerar som robotens huvuddator. [13]. Med ROS 2 möjliggörs mer modulär och distribuerad programvara. Några typiska ROS-paket som används inom robotik är `slam_toolbox` för hinderdetektion och kartläggning samt `nav2` för autonom ruttplanering och styrning. Tidigare års projekt utvecklade programvara för att i realtid kartlägga omgivningens körfält och vägmarkeringar, som kallas Hermes [10].



Figur 2.4: Bild av TurtleBot 4 med AprilTag.

2.9 Sensor fusion

Sensorfusion innebär att extrahera information från flera olika sensorer och på så vis generera en uppfattning om ett systems tillstånd [28]. Den resulterande uppfattningen är mer omfattande och berikad än varje sensor individuellt. För att åstadkomma detta kombineras ofta sensorer med olika styrkor, där varje sensor kan kompensera för andras svagheter. Exempelvis används radar, LiDAR och kamera tillsammans inom fordonsindustrin, där kameror kan ge detaljerad visuell information men har begränsad funktion i dimma och regn, medan radar och LiDAR är mer robusta under sådana förhållanden [29].

Det är också vanligt att använda flera likadana sensorer för ökad precision eller större total räckvidd. Brus och störningar kan ibland försämra en sensors pålitlighet och detta kan då reduceras om bruset inte påverkar alla sensorer likadant.

2.10 Extended Kalman Filter(EKF)

Extended Kalman Filter (EKF) är en metod för att kombinera data från flera sensorer i syfte att uppskatta ett systems tillstånd, exempelvis en robots position och orientering. Till skillnad från det linjära Kalmanfiltret hanterar EKF icke-linjära rörelse- och mätmodeller, vilket gör det lämpligt för mobila robotsystem.

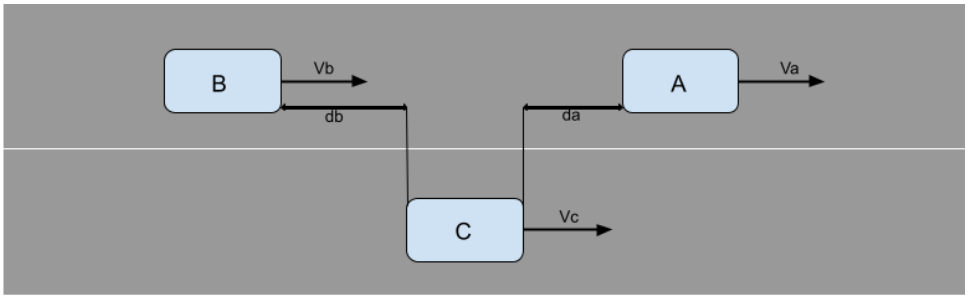
Filtret består av två steg: en prediktion av tillståndet utifrån en rörelsemodell, följt av en uppdatering med ny sensordata. Genom att väga information från olika sensorer enligt deras osäkerhet skapar EKF en mer noggrann och robust uppskattning.

2.11 Evalueringskriteriet CRI

Vid utvärdering av den distribuerade kooperativa manövreringsalgoritmen studeras säkerhet i en trafiksituation där ett fordon ska sammanväva mellan två andra fordon för att ta sig upp på en motorväg från påfarten. Kriteriet för risk vid sammanvävning är Cut-in Risk Indicator (CRI), ett mått som bygger på information om relativa avstånd mellan robotar och deras momentana Time to Collision, (TTC) [8]. En förtydligande bild av trafiksituationen kan ses i figur 2.5.

TTC mellan två fordon C och A definieras enligt nedanstående ekvation 2.1, där d_a är avståndet mellan de två fordonen och v_a, v_c är fordonens respektive hastigheter [8]:

$$\mathrm{TTC}_{ca} = \frac{d_a}{v_c - v_a} \forall v_c > v_a \tag{2.1}$$



Figur 2.5: Trafiksituation där fordon C ska sammanväva mellan fordon A och B.

CRI fås sedan som sambandet mellan fordonet C som ska sammanväva mellan både framförvarande fordon A och bakomliggande fordon B, se ekvation 2.2 [8]:

$$\text{CRI} = \text{CRI}_a + \text{CRI}_b, \begin{cases} \text{CRI}_a = \begin{cases} \exp(-\text{TTC}_{ca} \cdot k_a) \; \forall \; \text{TTC}_{ca} \geq 0 \\ 0 \text{ annars} \end{cases} \\ \text{CRI}_b = \begin{cases} \exp(-\text{TTC}_{bc} \cdot k_b) \; \forall \; \text{TTC}_{bc} \geq 0 \\ 0 \text{ annars} \end{cases} \end{cases} \tag{2.2}$$

Där k_a och k_b är de relativa avståndet enligt ekvation 2.3 [8]:

$$k_a = \frac{d_a}{d_a + d_b}, k_b = \frac{d_b}{d_b + d_a} \; \forall \; d_a, d_b \geq 0 \tag{2.3}$$

CRI antar ett värde mellan 0 och 2 och beskriver den totala kollisionsrisken mellan fordonet C och antingen fordon A eller B. CRI_a och CRI_b antar maximalt värdet 1 vid kollision med fordon A respektive B, därmed ses värdet 1 som hög risk men behöver nödvändigtvis inte innebära kollision. [8].

3

Metod

Arbetet har genomförts på tre delsystem där förbättrar och nya funktioner implementerats för att lösa tidigare problem. Angående GulliView fanns stort förbättringspotential för noggrannhet och fördröjning av systemet. Sedan krävdes nya funktioner för att det kooperativa manövreringssystemet skulle kunna hantera flera olika trafiksituationer. Slutligen undersöks det även om positioneringen skulle kunna gynnas av integrering av både interna och externa lokaliseringsmetoder.

3.1 GulliView

Målet var att implementera flertalet förbättringar vilket skulle leda till en positiv inverkan gällande GulliView systemets noggrannhet, fördröjning samt användarvänlighet. Dessa förbättringar består av att först justera funktionen som sköter distorsionskorrigeringen av bilden i syfte att minska fördröjningen. Därefter implementerades ett globalt koordinatsystem för att öka noggrannheten samt förbättra kommunikationen mellan kameror och fordon baserat på världskoordinater. Utöver dessa förbättringar omstrukturerades även koden till mindre filer för att förenkla framtida arbete på projektet. Slutligen skapades även ett program för att evaluera exekveringstiden av olika delmoment i systemet.

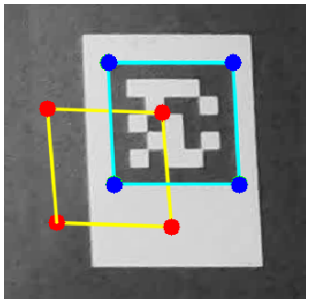
3.1.1 Effektivare distorsionskorrigering

På grund av kamerans lins sker en viss distorsion på bilderna som skickas till GulliView. Den mest framträdande är en sorts radiell distorsion orsakad av kamerans lins. Denna distorsion blir starkare närmare kanter och särskilt hörnen av bilden. Utöver det förekommer även tangentiell distorsion då kameran kan vara lutad vilket gör att vissa punkter uppfattas närmre än andra. För att motverka förvrängningen, alltså distorsionskorrigering av bilden, användes tidigare funktionen `remap` som fås av OpenCV biblioteket[30]. `Remap` gör en distorsionskorrigering av hela bilden genom att kartlägga transformationen på varje individuell pixel baserat på förkalibrerade förvrängningsmatriser för kameran. Resultatet blir en verklighetsriktig bild men samtidigt är detta en process som tar mycket lång tid.

Distorsionskorrigering av bilder bygger på två variabler. Den första är en 3×3 kameramatris, ofta betecknas med K, och den andra är en 5-dimensionell vektor innehållande distorsionskoefficienter, betecknas här d. Se ekvation 3.1. Dessa variabler är olika för varje kamera och måste därmed beräknas och kalibreras individuellt. Detta har genomförts i

tidigare projekt och samma beräknade värden användes i detta projekt [3].

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad d = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \tag{3.1}$$



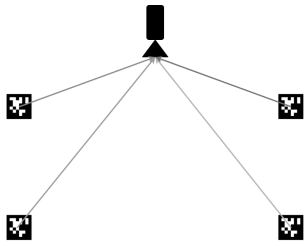
Figur 3.1: Distorsionskorrigering av hörnen. De röda punkterna indikerar de punkter som Apriltagdetektorn detekterade i den förvrängda bilden och de blåa punkterna indikerar de punkterna efter distorsionskorrigering på verklighetsenlig bild.

Istället för att genomföra distorsionskorrigeringen på hela bilden beslutades det att endast distorsionskorrigera de relevanta punkterna för att spara på betydlig prestanda. Därmed implementerades ett system där bilden hämtas, fortfarande förvrängd, och AprilTag detektorn letar efter möjlig AprilTag. Om en AprilTag hittas, sparas detektionen och detektionens koordinater. Det vill säga pixelkoordinaterna för hörnen, vilket sedan distorsionskorrigeras genom användningen av funktionen `undistortPoints`.

`UndistortPoints` är en funktion i biblioteket OpenCV som tar in punkter, en kameramatris och en vektor av distorsionskoefficienter och returnerar de givna punkternas position i den distorsionskorrigerade bilden [30]. Detta kan ses i figur 3.1 där de förvrängda (röd) och distorsionskorrigerade (blå) punkterna ritas upp på den verklighetsenliga bilden. Detta leder till att distorsionskorrigeringen går från att bearbeta en hel bild, i antingen Full HD eller 4K upplösning, till att enbart bearbeta ett fåtal punkter.

3.1.2 Globalt koordinatsystem

Implementeringen av det globala koordinatsystemet genomfördes i två delar. Den första delen bestod utav att, under kalibreringsfasen, beräkna pose mellan kalibreringstaggarna och kameran. Där pose beskriver relationen mellan två objekt med hjälp av en rotationsmatris och en translationsvektor. Den andra delen gick ut på att, under programmets körning, beräkna pose för fordonets tagg och sedan kombinera det med kalibreringsvärdena från första delen. Detta utförs genom flertalet ekvationer och resulterar i en position och rotation i rummet, vilket följer samma standard för alla kameror.



Figur 3.2: Grafisk representation av alla fyra Apriltags som en kamera ser.

För att beräkna kalibreringstaggarnas rotationsmatris och translationsvektor skapades ett program för att ta in en bild ämnad för kalibrering och sedan beräkna pose av varje tagg till kameran. Pose beräknades individuellt för varje tagg och varje kamera. Eftersom varje kamera kan se fyra taggar och det finns fyra kameror blev det totalt sexton rotationsmatriser och translationsvektorer. Först kör programmet en AprilTag detektion för att hitta de fyra kalibreringstaggarna och sedan distorsionskorrigeras varje taggs hörnkoordinater. De fyra taggarna i kamerans synfält kan ses i figur 3.2. Dessa koordinater, `pixel_points`, är i pixel format på kamerans bild. Dessa pixelkoordinater relateras till världskoordinater vilket är hörnen av en AprilTag i världen i meter, förutsatt att taggens mittpunkt har koordinaterna i origo (0,0,0) i rummet. Detta möjliggör att sedan addera världskoordinaterna av en kalibreringstagg för att få positionen. I ekvation 3.2 ses hur världskoordinaterna och pixelkoordinaterna definieras, där a är storleken på en AprilTag dividerat med 2 för att beräkna koordinaterna av hörnen utifrån att origo är i mitten av taggen. Då AprilTags i detta projektet hade storleken 0.16×0.16 meter är $a = 0.08$. P är en vektor med alla hörnen, i pixelkoordinater, som gavs av AprilTag detektorn.

$$\text{world\_points} = \begin{bmatrix} -a & a & 0 \\ a & a & 0 \\ a & -a & 0 \\ -a & -a & 0 \end{bmatrix}, \quad \text{pixel\_points} = \text{undistortPoints}(\begin{bmatrix} p[3] \\ p[2] \\ p[1] \\ p[0] \end{bmatrix}) \tag{3.2}$$

I nästa steg beräknades rotations- och translationsvektor, r och t, genom OpenCV funktionen `solvePnP` [30]. `SolvePnP` står för Solve Perspective n Point och beräknar transformationen mellan de givna tre-dimensionella punkterna i världen i relation till de givna två-dimensionella punkterna i bilden [31]. `SolvePnP` kräver även kamera matrisen, K, och, om pixel koordinaterna är förvrängda, kamerans distorsionskoefficienter. Däremot är punkterna redan distorsionskorrigerade och en ytterligare distorsionskorrigering hade medfört felaktig position och därmed lämnas de nollställda. Den estimerade pose som `solvePnP` ger ut i form av en rotationsvektor, r, och en translationsvektor, t, berättar hur en tre-dimensionell punkt i världen projekteras till en två-dimensionell punkt i kamerans bild.

Rotationsvektorn, lilla r, transformerades med hjälp av OpenCV funktionen `Rodriguez` till en 3×3 matris, R. Detta görs för att möjliggöra enkel rotation genom att matrismultiplicera rotationsmatrisen, R, med punktens koordinater för att få punktens koordinater efter rotationen [30].

Sista steget som genomfördes för att beräkna korrekt pose, var att invertera R och t. Detta gjordes då R och t beskrev projektionen från en punkt i världen till en punkt i bilden. Därmed kräver ett globalt koordinatsystem, som beräknar koordinater i världen baserat på bildens koordinater, det motsatta. Det vill säga projektionen från en punkt i bilden till en punkt i världen. Då R är en rotationsmatris görs en invertering genom en transponering av matrisen. Inversen av translationsvektorn, t, beräknades genom att multiplicera t med negativa R. Dessa två beräkningar kan ses i ekvation 3.3. Värdena på R^{-1} och t^{-1} beräknades för varje kamera och dess fyra kalibreringstaggar för att sedan sparas för användning vid bestämning av globala koordinater.

$$R^{-1} = R^T, \qquad t^{-1} = -R \times t \tag{3.3}$$

När programmet för estimering av pose var klart för varje kamera och tagg började implementationen av funktionen inuti GulliView som skulle beräkna världskoordinaterna. Den nya distorsionskorrigeringen integrerades i både *Fast Thread* och *Nice Thread*. Vilket gjorde att när ett fordon detekterades, beräknades hörnen på fordonets AprilTag i den distorsionskorrigerade bilden. Dessa hörn skickades sedan till den nya världskoordinatfunktionen. Funktionens invärden är kamerans id samt fordonets fyra hörn och funktionen returnerar fordonets estimerade position och rotation i världen.

Det första funktionen gjorde var att beräkna fordonets pose. Detta gjordes på samma sätt som vid estimering av pose för kalibreringstaggarna och följer samma matriser som definieras i ekvationen 3.2. Fast i detta fallet var pixelpunkterna istället de distorsionskorrigerade pixelkoordinaterna för fordonets hörn som detekterades. Medan världspunkterna var de samma, det vill säga att mittpunkten var i origo. Därefter exekverades `solvePnP` för fordonets värden och fordonets rotations- samt translationsvektorer beräknades. Rotationsvektorn transformerades till en 3×3 matris för framtida beräkningar. Sedan hämtas de korrekta rotationsmatriser och translationsvektorer, för kalibreringstaggarna, baserat på kamerans id. I detta sammanhanget var det fyra komponenter, med olika pose, och fordonets pose måste gå igenom flera ekvationer innehållande olika relationer mellan dessa komponenter för att slutligen få fram position och rotation i världen. De fyra komponenterna visas nedan.

obj fordonets tagg
tag kalibreringstagg på golvet
cam kameran
gbl den globala positionen i rummet

För varje kalibreringstagg inuti kamerans synfält exekveras följande. Först hämtas taggens specifika R och t utifrån taggens och kamerans id. Därmed har funktionen tillgång till R och t för både kalibreringstaggen och fordonets tagg. Se ekvation 3.4. Rotation från tagg till global referens berättar hur rotationen av taggen relaterar till rotationen av det globala koordinatsystemet. Anledningen till varför y-axeln är inverterad, det vill säga negativ, är då kalibreringstaggarna pekar uppåt i rummet medan globala koordinatsystemet går nedåt i rummet.

$$R_{gbl}^{tag} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_{tag}^{cam}, \quad t_{tag}^{cam}, \quad R_{cam}^{obj}, \quad t_{cam}^{obj} \tag{3.4}$$

tag 0	$\begin{bmatrix} 0.000 & 0.000 & 0 \end{bmatrix}$
tag 1	$\begin{bmatrix} 4.295 & 0.000 & 0 \end{bmatrix}$
tag 2	$\begin{bmatrix} 0.000 & 2.002 & 0 \end{bmatrix}$
tag 3	$\begin{bmatrix} 4.292 & 1.995 & 0 \end{bmatrix}$
tag 4	$\begin{bmatrix} 0.000 & 4.003 & 0 \end{bmatrix}$
tag 5	$\begin{bmatrix} 4.286 & 3.995 & 0 \end{bmatrix}$
tag 6	$\begin{bmatrix} 0.000 & 5.992 & 0 \end{bmatrix}$
tag 7	$\begin{bmatrix} 4.264 & 5.997 & 0 \end{bmatrix}$
tag 8	$\begin{bmatrix} 0.000 & 7.989 & 0 \end{bmatrix}$
tag 9	$\begin{bmatrix} 4.264 & 7.964 & 0 \end{bmatrix}$

Tabell 3.1: Kalibreringstaggarnas position i rummet med den vänstra taggen längst in i rummet som referenspunkt.

För att beräkna fordonets position i världen beräknades först translationen från fordonet till kalibreringstaggen. Detta gjordes genom att matrismultiplicera rotationsmatrisen från kamera till kalibreringstaggen med translationen från fordonet till kameran och sedan adderades translationen från kamera till kalibreringstagg. Utifrån denna translation kunde positionen beräknas genom matrismultiplikation mellan rotationen från kalibreringstagg till ett globalt koordinatsystem, vilket är definierad i ekvation, och translationen som beräknades. Utifrån det fås endast den globala positionen utifrån origo (0,0,0) och därmed adderades detta med kalibreringstaggens världsposition, vilket ses i tabell 3.1, för att få ut den globala positionen. Se ekvation 3.5.

$$\begin{aligned} t_{tag}^{obj} &= R_{tag}^{cam} t_{cam}^{obj} + t_{tag}^{cam} \\ \text{pos}_{gbl} &= R_{gbl}^{tag} t_{tag}^{obj} + \text{pos}_{tag} \end{aligned} \tag{3.5}$$

Beräkningen av rotationen skedde genom ytterligare matrismultiplikationer. För att få rotationen från fordonet till kalibreringstagg görs en matrismultiplikation mellan rotationen från kamera till tagg med rotationen från fordon till kamera. Slutligen beräknades rotationen i världen genom att matrismultiplicera rotationen från kalibreringstagg till globalt koordinatsystem med rotationen från fordonet till kalibreringstagg. Se ekvation 3.6.

$$\begin{aligned} R_{tag}^{obj} &= R_{tag}^{cam} R_{cam}^{obj} \\ R_{gbl} &= R_{gbl}^{tag} R_{tag}^{obj} \end{aligned} \tag{3.6}$$

Dessa steg repeterades för alla resterande kalibreringstaggar inom kamerans vy. Efter att detta hade gjorts för varje tagg fanns det fyra estimeringar av position och rotation utifrån varje taggs perspektiv. Dessa värden sammanställdes genom att beräkna medelvärde av alla fyra värden för att få ut den slutliga globala positionen och rotationen av fordonet i världen.

3.1.3 Omstrukturering av koden

För att göra GulliView mer användarvänligt under utvecklingen och för framtida arbete valdes det att göra en omstrukturering av koden. Innan var hela GulliView skriven i samma kodfil. Vilket resulterade i mindre effektivitet under arbetet. På grund av detta delades kodfilen upp i flertalet kodfiler med planen att varje kodfil fyller en viss unik funktion. De olika kodfilerna bestämdes genom att kategorisera alla funktioner i GulliView och hitta samband och i så fall kombinera de som hör ihop. Genom funktioner i programmeringsspråket C++, som `#define` och `#include` samt headerfiler `.hpp` kunde dessa kodfiler sedan kopplas samman till varandra för att dela funktioner på ett enkelt och standardiserat sätt. Den nya stukturen av de olika filerna och hur de är sammankopplade kan ses i bilaga F.

3.1.4 Förbättrad evaluering

Ett program i språket Python utvecklades för att förenkla evaluering kring systemets uppdateringsfrekvens och fördröjningen. Detta för att göra det till ett användbart program under utvecklingen av GulliViews prestanda. Först implementerades det tidsmätande funktioner i GulliView vilket kontinuerligt samlade data angående funktioner och mätte dess tid för att spara i evalueringsfiler. Sedan analyserar programmet dessa filer för att få ut och kategorisera värdena. Den data som programmet analyserade var bland annat exekveringstid för olika funktioner, inklusive tiden det tar för en hel cykel från att GulliView får en ny bild från kameran till att meddelandet om detektion skickas till fordonet. För att visualisera datan skapades verktyg för att plotta datan i låddiagram.

3.2 Distribuerade kooperativa manövreringssystem

Den andra delen av projektet har handlat om att att utveckla ett distribuerat system för kooperativa manövreringar. Planen var att utveckla ett system för kooperativ körning i en mängd olika trafiksituationer. Robotarna skulle kunna köra igenom situationer som korsningar, sammanvävning av körfält och cirkulationsplatser utan att stanna eller kollidera. För att göra robotarna mer självständiga och ej beroende av ett yttre kontrollsystem baseras detta på distribuerad databehandling, där datorerna i vardera robot fattar besluten i hanterandet av trafiksituationen. Detta möjliggörs genom en kommunikation mellan systemen där data delas och en gemensam uppfattning över situationen bildas. Då detta system bygger på befintliga program behöver denna nya funktionalitet integreras. Den kod vi utvecklat finns tillgänglig på laboratoriets privata bitbucket samt en privat Github, se bilaga D.

På liknande vis som ett tidigare arbete [6], skall detta projekt implementera sin funktio-

nalitet (se avsnitt 2.6). Den största skillnaden är dock att den hastighetskontrollerande delen skall flyttas över från en extern dator till robotarnas egna datorer. Det nya systemet kommer därav inte använda noden `laptop_socket_server` utan i dess ställe via en ny ROS nod `cooperative_maneuver_control`, publicera på ROS topicen `gv_laptop` för att styra robotarnas hastigheter.

3.2.1 Distribuerad manövrering för överlappande trafik

För att göra systemet applicerbart i flera olika situationer baseras det på att definiera ett visst område i rummet där endast en robot får vistas åt gången. Genom att applicera detta område på ett ställe där två robotars körbana korsas, som i en korsning, förhindras därmed en kollision med hjälp av dessa områden som i fortsättningen betraktas som exklusiva zoner. Detta område kan även vara uppdelat i ett flertal delområden där det totala området får beträdas av flera robotars samtidigt om de vistas i olika delområden.

Via kontakt med GulliView samlar alla robotar ständigt information om varandra och sig själva. Detta omfattar robotens id, dess positionskoordinater samt dess vinkel i förhållande till koordinatsystemet. Med denna information har varje robot vetskap om varandras relation till dessa exklusiva zoner. Robotarna gör därav beräkningar kring trafiksituationen för alla områden. För att spara på onödiga beräkningar etableras ett visst upptäcktsavstånd från den exklusiva zonens mittpunkt. Robotar inom detta avstånd anses vara relevanta för detta område och inkluderas i beräkningar för det, medan utomstående exkluderas. Robotarna har även en kommunikation emellan sig via Rostig för att dela information om sig själva relaterat till ett visst område. För att undvika onödiga meddelanden, skickar bara robotarna informationen till de som är relevanta för den specifika zonen.

Med etablerade exklusiva zoner krävs även vetskap om robotens planerade rutt, för att beräkna hur den kommer köra i förhållande till zonerna. Av denna anledning etableras ROS topicen path, som noden `go_to_goal` publicerar sin planerade körrutt på. Med vetskap om robotens rutt itererar programmet över dessa punkter och noterar alla de exklusiva områden och delområden som roboten kommer köra igenom. Under iterationen beräknas även avstånd och de specifika ingång och utgångs punkter till de olika zonerna. Under robotens färd beräknas avståndet till in- och utgångs punkterna kontinuerligt.

För att bara tillåta en robot att beträda de exklusiva områdena i taget används ett bokningssystem. Med detta kan området eller olika delområden reserveras av en robot och hindra andra från att åka in i det. Alla robotars bokningar sker på alla robotar som är inom upptäcktsavståndet för zonen. Vid situationer där flera robotar vill boka samma område eller delområde samtidigt skapas en prioritetsordning, där robotar med högre prioritet får chansen att boka innan de med lägre. Denna prioritetsordning baseras på robotens beräknade ankomsttid till området samt de eventuella delområden som roboten kommer köra igenom. Om robotarnas ankomsttid är inom en sekund relativt varandra ges däremot prioritet till roboten med lägst id värde, för att undvika oenighet. När en robot fått sin bokning, kört igenom och ut ur området, genomförs avbokningar. Om en robot inte skulle få tillgång till ett område justeras dess hastighet så att roboten anländer vid området den tidpunkt då den fått högst prioritet, enligt ekvation 3.7 med fullständigt flödesschema i figur 3.3.

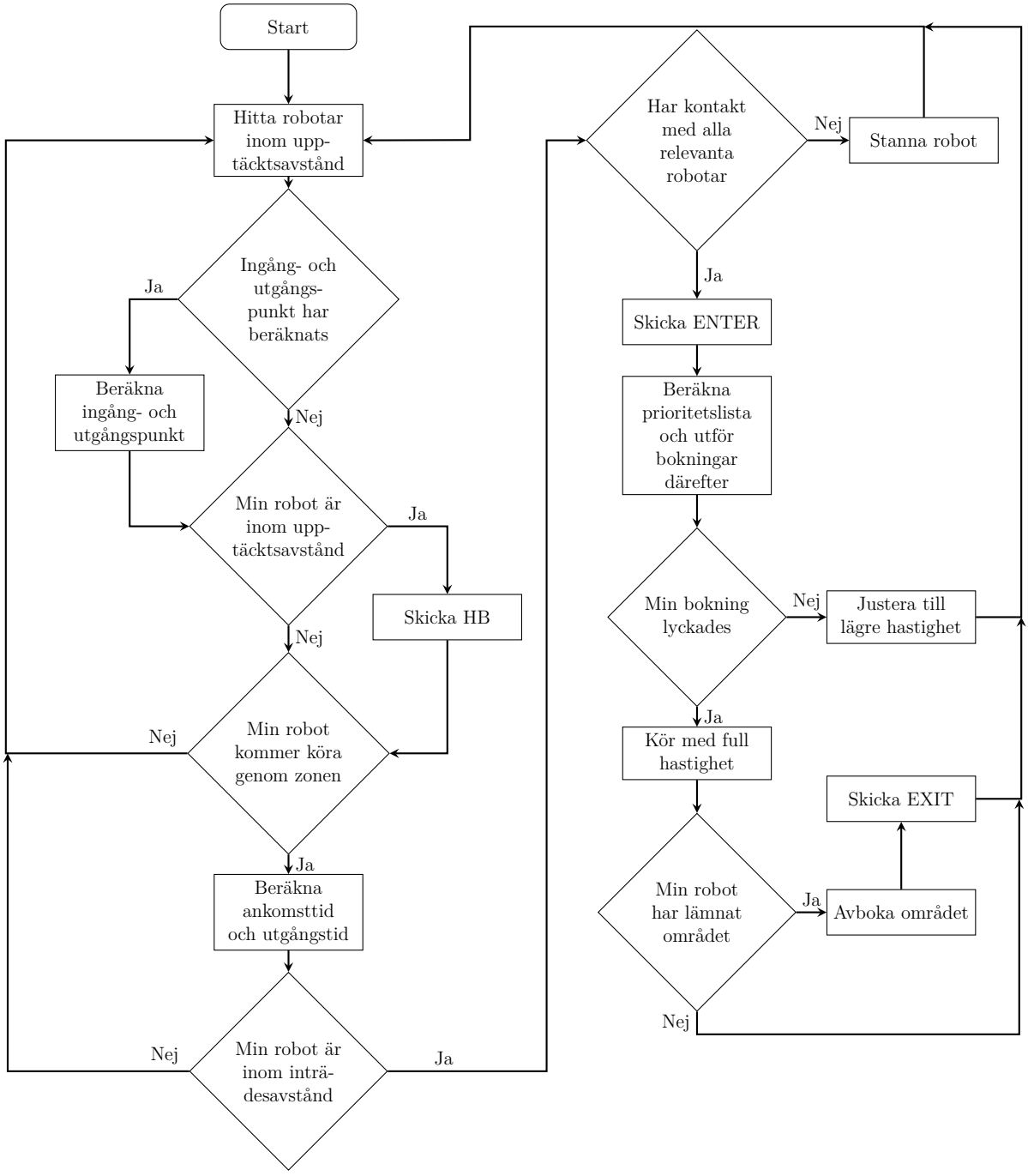
$$\text{Hastighet} = \frac{\text{Avstånd till område}}{\text{Tid till ledigt}} \tag{3.7}$$

3.2.2 Kommunikationsprotokoll för kooperativ manövrering

För att säkerställa en kollisionsfri manövrering genom de exklusiva zonerna upprätthåller robotarna, som är relevanta till varandra, en kontinuerlig kommunikation. När exempelvis två robotar är på väg mot och är inom räckvidd av en exklusiv zon, skickar de statusuppdateringar till varandra. Denna kommunikationsstruktur är baserad på konferensartikeln [5], där en kommunikationsalgoritm för autonoma fordon i en fyrvägskorsning presenteras. Precis som i denna artikel är meddelanden mellan robotarna utformade efter en viss struktur i form av en lista där första elementet är robotens id, andra är meddelandets typ, som därefter är följt av data relevant till meddelandet.

Till skillnad från denna artikel är kommunikationen mellan robotarna i detta arbete relaterad till en specifik exklusiv zon. Av denna anledning inkluderas även zonens namn i meddelandet för att ge den mottagande roboten meddelandets sammanhang. De tre meddelandetyperna som även används i artikeln är HB (Heartbeat), ENTER och EXIT. Ett HB meddelande skickas kontinuerligt och meddelar omgivande robotar om att dess kooperativa manövreringsprogram är aktivt och fungerar. Den andra typen är ENTER meddelanden som skickas då roboten är inom inträdesavståndet till zonen och förmedlar att roboten börjat genomföra bokningar och planera för manövreringar genom zonen. Den sista typen är EXIT meddelanden och skickas då roboten lämnat en zon och förmedlar till omgivande fordon att området är ledigt.

Under kommunikationen av robotarna mäts tiden mellan meddelanden. Överstiger detta värde en viss gräns, antas kommunikationen med roboten vara bruten och färd genom zonen anses ej vara säker. I denna situation stannas roboten tills kommunikationen återupptagits, se figur 3.3. Denna säkerhetsrutin är även den inspirerad av konferensartikeln.

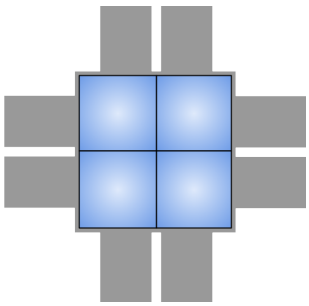


Figur 3.3: Flödesschema av manövreringsalgoritmen för exklusiva zoner.

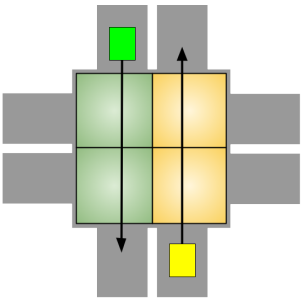
3.2.3 Trafikimplementationer

Eftersom denna metod baseras på att definiera ett visst område där endast ett fordon får vistas åt gången, kan detta system implementeras i ett flertal trafiksituationer. Där flera vägar möts, och trafik korsar eller överlappar varandra, kan en exklusiv zon definieras. Ett första exempel på en implementation är korsningar. En korsning kan implementeras genom att definiera en exklusiv zon i det område där de ingående vägarna möts, se figur 3.4. Då systemet kan hantera delområden i zonen kan även flerfiliga vägar hanteras. I en sådan här situation kommer ett fordon endast reservera de delområden som den kommer beträda och lämna övriga lediga för andra fordon.

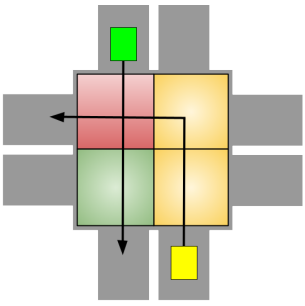
I situationen av en fyrvägskorsning är zonen uppdelad i fyra delområden, se figur 3.4. För att ta sig igenom korsningen behöver en robot lyckas boka alla de delområden som kommer användas. I en situation där två robotar från motsatta håll kör mot korsningen och planerar köra rakt igenom den, kommer fordonen behöva använda olika delområden. Av denna anledning kommer båda robotar lyckas köra igenom korsningen samtidigt utan att förändra sina hastigheter, se figur 3.5. Om dessa robotar skulle behöva använda samma område (se figur 3.6) kommer en av dem behöva justera sin hastighet med det beskrivna kontrollsystemet. Med detta kan robotarna ta sig igenom korsningen på ett säkert vis, utan onödiga fördröjningar.



Figur 3.4: Fyrvägskorsning med exklusiv zon uppdelad i fyra delområden.

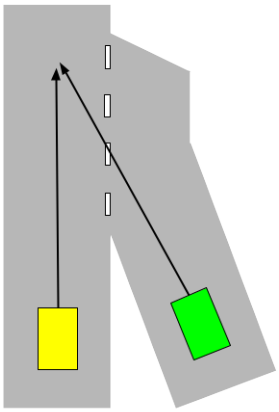


Figur 3.5: Fyrvägskorsning med exklusiva zoner och robotar som kommer passera varandra.

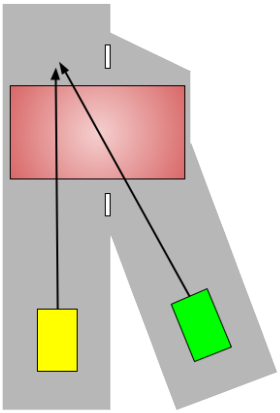


Figur 3.6: Fyrvägskorsning med exklusiva zoner och robotar som kommer korsa varandra.

En annan trafiksituation som detta system kan appliceras på är sammanvävningar av körfält. Likt föregående exempel omfattar denna situation vägar som överlappas, se figur 3.7. Även i denna situation kan ett område markeras ut, där fordonens färd på de båda vägarna korsas och en potentiell kollision kan uppstå, se figur 3.8. Genom att applicera en exklusiv zon över detta område kommer en av robotarna i situationen anpassa sin hastighet för att släppa fram den andra. Systemet kommer då förhindra en kollision i detta område och lämna ett minimum avstånd mellan robotarna som är längden på den exklusiva zonen. När väl robotarna tagit sig förbi zonen kan deras avstånd bibehållas med en farthållare, se bilaga B.

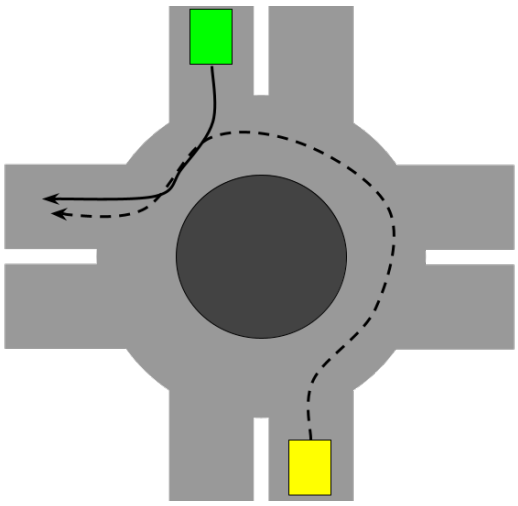


Figur 3.7: Sammanvävning av körfält med två robotar vars körriktning korsas.

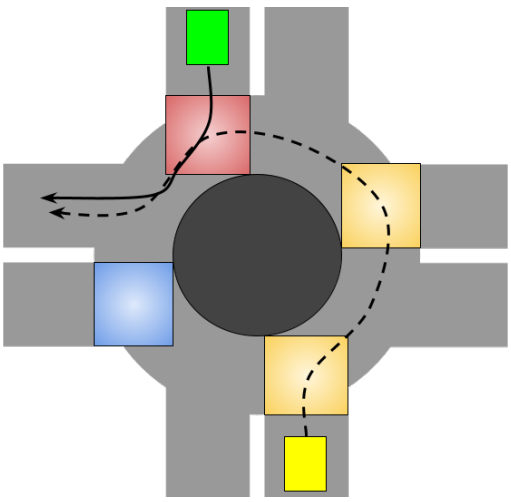


Figur 3.8: Sammanvävning av körfält med två robotar och ut placerad exklusiv zon.

Slutligen skulle detta system även kunnat appliceras i en cirkulationsplats. Även i denna situation överlappar flera körfält och robotar från olika håll riskerar att krocka, se figur 3.9. Inne i en cirkulationsplats rör sig däremot all trafik åt samma håll vilket innebär att trafik mellan fordon inuti och på väg ut ur cirkulationsplatsen kan hanteras med farthållaren, se bilaga B. Det är mellan trafik inuti och på väg in i cirkulationsplatsen som ytterligare planering krävs. Denna situation kan dock liknas med en sammanvävning av körfält då två olika vägar går ihop och fortsätter in i rondellen. Av denna anledning framgår det då även att systemet kan appliceras på samma vis, som tidigare beskrivits. Zonen ska då placeras så den täcker de områden som inkommande fordon och fordon i cirkulationsplatsen kommer använda, se figur 3.10.



Figur 3.9: Cirkulationsplats med två robotar och dess körbana.

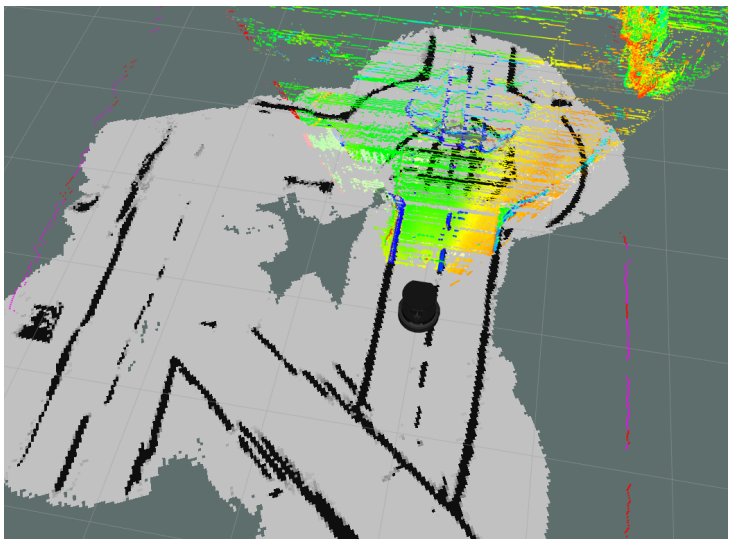


Figur 3.10: Cirkulationsplats med två robotar och exklusiva zoner.

3.3 Integrering av intern- och extern lokalisering

För att möjliggöra ett skalbart och robust system för autonom navigering i laboratoriet genomfördes ett antal hårdvaru- och mjukvaruförbättringar av TurtleBot 4-plattformen och det befintliga Hermes-systemet. Arbetet fokuserade främst på att modernisera kodbasen, möjliggöra samtidig drift av flera robotar, samt förbättra positionsuppskattningen genom integration med externa sensorkällor. Koden vi använt finns tillgänglig på Github, se bilaga C.

En tydlig brist i Hermes-systemet är den uppbyggda driften i odometri/SLAM, som syns i figur 3.11. Detta motiverar en sammanslagning av den interna lokaliseringen (odometri/SLAM) med den externa lokaliseringen från GulliView. Odometri/SLAM ger hög frekvens i positionsuppdatering men leder till ökande fel över tid. GulliView, med sina takmonterade kameror, tillför absoluta referenser med minimal drift men med lägre uppdateringsfrekvens. Genom sensorfusion kombinerar vi dessa mätningar så att SLAM:s lokala skattning regelbundet korrigeras mot GulliViews absoluta position, något som är nödvändigt för att hindra den snabba avvikelse som redan efter korta körsträckor kan observeras.



Figur 3.11: Kartläggning med Hermes: den ljusgrå bakgrundskartan och de svarta konturerna visar den uppbyggda miljön, medan det färgade punktmolnet avspeglar sensorernas realtidsdata. Bilden illustrerar hur TurtleBotens odometri (dödräkning) snabbt börjar avvika från verkligheten.

3.3.1 Uppdatering och utveckling av mjuk- och hårdvara

Vid projektstart visade det sig att den överlämnade kodbasen från föregående projektgrupp inte var fullt kompatibel med den nuvarande tekniska miljön. Trots att strukturen i systemet var intakt, kunde varken Hermes-systemet eller TurtleBot 4 initieras korrekt i kombination med Ubuntu 22.04, ROS 2 Humble och de senaste versionerna av tillhörande beroenden.

En djupgående felsökningsprocess identifierade att problemens huvudsakliga orsak var

förändringar i externa paket, inklusive uppdaterade versioner av `numpy`, `numba`, `cv_bridge` och andra centrala ROS-paket. Dessutom hade ROS 2 Humble infört mer strikt hantering av namespaces, vilket krävde att flera delar av systemet anpassades. Detta omfattade justeringar i lanseringsfiler såsom `start_hermes.launch.py`, samt omstrukturering av nodspecifik kommunikation i filer som `depth_intensity_image_syncer.py` och `depthCam.launch.py`. Även kamerans konfigurationsfil `pc.yaml` krävde uppdatering för att korrekt stödja ROS 2:s nya kommunikationsmodell.

För att systemet ska fungera som tidigare behövs ett kamerastativ. Tidigare projektgrupp hade använt ett fristående kamerastativ som inte längre fanns tillgängligt. Därmed behövdes ett nytt fysiskt fäste utvecklas för TurtleBotens kamera. Ett nytt kameraställ konstruerades i CAD programmet Autodesk Inventor och tillverkades genom additiv tillverkning. Det nya stället är monterbart direkt på robotens chassi och tillåter justering av kamerans vinkel och höjd. Genom att integrera kameran med robotens konstruktion förväntas stabiliteten förbättras i datainsamlingen, vilket i sin tur leder till bättre prestanda för SLAM-baserade kartläggnings- och navigeringsalgoritmer.

För att systemet ska fungera som avsett krävs att flera TurtleBot 4-enheter kan köras parallellt i samma nätverksmiljö. I ROS 2 används ett namespace som ett prefix framför t.ex. nod- eller topic-namn för att gruppera och undvika namnkonflikter. Detta behövde implementeras i hela systemet. Till exempel implementerades stöd för dynamisk namespace-hantering i Hermes-systemet. Genom att varje robot tilldelades ett unikt namespace vid initiering kunde alla noder och topic isoleras från varandra, vilket eliminerade konflikter i kommunikationen. Denna lösning realiserades genom att ersätta statiskt definierade topic-namn med parametriserade strängar baserade på robotens ID. Vilket blev ett mer modulärt och flexibelt system där robotar kan köras parallellt utan att påverka varandras funktionalitet.

3.3.2 Implementering av extern lokalisering i äldre GulliView

I detta och kommande avsnitt kommer metoden för hur positionsdata från både den äldre och den nya GulliView publiseras på en ROS 2 topic.

För den äldre versionen av GulliView implementeras en ROS 2-baserad UDP-server, som tar emot positioneringsdata från GulliView-systemet och vidarebefordrar dessa till ROS 2-nätverket. När servern mottar ett UDP-datapaket avkodas det med funktionen `parse_packet`, som tolkar den binära strömmen till ett strukturerat objekt bestående av en header, med en tidsstämpel, och en lista av detektioner. För varje detekterat objekt extraheras relevanta fält som beskrivs nedan, och översätts till ett ROS 2-meddelande av typen `PoseWithCovarianceStamped`, vilket sedan publiceras. På så vis integreras GulliViews externa lokaliseringsdata direkt i robotsystemets “map”-koordinatsystem. Varje GulliView-detektion i paketet innehåller följande fält som motsvara formatet från GulliViews ursprungliga output:

- $\mathbf{x}$: Objektets x-position i GulliViews plan.
- $\mathbf{y}$: Objektets y-position i planet (samma enhet som x).
- $\boldsymbol{\theta}$: Objektets orienteringsvinkel i planet (radianer, definierad runt vertikal axel).

- **v**: Uppskattad hastighetsparameter (ingår i GulliViews data; används ej i denna nod men finns tillgänglig för framtida bruk).
- `tag_id`: Identifierare för objektets tagg (unikt ID-nummer för att särskilja olika objekt/robotar).
- `camera_id`: ID för den kamera som detekterade objektet (anger från vilken av flera kameror observationen kommer).

GulliView rapporterar objektets (x, y)-position i ett pixelbaserat lokalt plan för respektive kamera. För att omvandla detta till robotens globala kartkoordinater i meter appliceras en skalningsfaktor. Empiriskt har 1298 pixlar per meter fastställts baserat på GulliViews kalibrering. Därför beräknas den globalt tolkade positionen som visas i ekvation 3.8 där (x_{map}, y_{map}) är positionen uttryckt i meter i "map"-koordinatsystemet. Rörelse antas ske i markplanet, så höjdkomponenten sätts till noll (d.v.s. position.z = 0).

$$x_{map} = \frac{x}{1298.0}, \qquad y_{map} = \frac{y}{1298.0}, \tag{3.8}$$

GulliViews vinkel θ definieras så att en ökning sker moturs i planet kring den vertikala axeln. För att anpassa detta till ROS2-konventionen (där positiv yaw också är moturs) inverteras vinkeln enligt ekvation (3.9). Den inverterade vinkeln representeras sedan som en orientering i ROS2 genom en kvaternion, enligt ekvation (3.10).

$$\theta_{\text{map}} = -, \theta_{\text{GV}}. \tag{3.9}$$

$$q_x = 0, \quad q_y = 0, \quad q_z = \sin!\left(\tfrac{\theta_{\text{map}}}{2}\right), \quad q_w = \cos!\left(\tfrac{\theta_{\text{map}}}{2}\right). \tag{3.10}$$

En kvaternion är en matematisk representation som används för att effektivt beskriva rotationer i tre dimensioner utan att drabbas av de problem (t.ex. gimbal lock) som kan uppstå med andra representationsformer som Euler-vinklar. Denna representation beskriver en rotation runt z-axeln i planet och används direkt i ROS2-meddelandet msg.pose.pose.orientation. För att indikera osäkerheten i positionen beräknas en kovariansmatris utifrån avståndet d mellan detektionen och kamerans centrum (x_{cam}, y_{cam}) i pixelplanet enligt ekvation 3.11. Därefter sätts positionsvariansen enligt ekvation 3.12.

$$d = \sqrt{(x - x_{cam})^2 + (y - y_{cam})^2}. \tag{3.11}$$

$$\sigma_{pos}^2 = 0.01 + 10^{-6}\, d, \tag{3.12}$$

Detta ger en basosäkerhet på $0.01\,\text{m}^2$ som ökar marginellt med avståndet. Orienteringsvariansen sätts vanligtvis till en konstant, t.ex. $\sigma_\theta^2 = 0.1\,\text{rad}^2$. Dessa värden placeras på diagonalen i en 6×6 kovariansmatris (position x, y och yaw), medan övriga element är noll. Matrisen tilldelas `msg.pose.covariance` innan publicering. När `PoseWithCovarianceStamped` meddelandet skapats publiceras det på ROS2 topicen

`/namespace/gv_pose`. Noden deklarerar en parameter `tag_id` som filtrerar ut detektioner med avvikande taggar och endast objekt vars tag_id matchar parametern publiceras. Varje meddelande får en ROS2-tidstämpel (hämtad ur paketets millisekund-fält konverterat till sekunder) och fältet `frame_id` sätts till "map" för att indikera global referens.

$$d = \sqrt{(x - x_{cam})^2 + (y - y_{cam})^2}. \tag{3.13}$$

$$\sigma_{pos}^2 = 0.01 + 10^{-6}\, d, \tag{3.14}$$

Därefter sätts positionsvariansen till enligt ekvation (3.14) `gv_socket_server.py` implementeras som en ROS2-nod kallad `GulliViewServerNode`. Vid uppstart deklarerar noden följande parametrar som kan anpassas:

- `host` (sträng, standard "0.0.0.0"): IP-adressen som UDP-servern binder till. `"0.0.0.0"` innebär att lyssna på alla nätverksgränssnitt.
- `port` (heltal, standard 2121): UDP-port som noden lyssnar på för inkommande GulliView-paket. Detta måste matcha porten som GulliView-systemet sänder data på.
- `tag_id` (sträng, standard "all"): ID-filter för detektioner. Värdet "all" innebär ingen filtrering (publicera alla taggar), medan ett numeriskt ID anger att endast detektioner med motsvarande tag_id publiceras.

När noden startas loggas en startbekräftelse och därefter initieras UDP-servern på host:port. Servern hanterar inkommande paket i en separat tråd (via Python funktionen socketserver.UDPServer) och anropar `GulliViewPacketHandler` för varje paket. GulliViewPacketHandler är definierad som en inre klass som tar emot paket, utför tolkning, filtrering och meddelandebyggnad, och använder en referens till nodens ROS2-publisher för att publicera PoseWithCovarianceStamped-meddelandet. Noden kör kontinuerligt genom `rclpy.spin` tills avstängning, varefter UDP-servern stängs ned ordnat.

3.3.3 Integration för nyare GulliView-system med globalt koordinatsystem

Till skilljand från föregående avsnitt beskrivs här en implementation utvecklad för att stödja en nyare version av GulliView, som har utvecklas under detta projektets gång. I denna version levererar GulliViews egen programvara objektpositioner direkt uttryckta i ett *globalt* koordinatsystem. Flera kameror har kalibrerats mot en gemensam referens, vilket innebär att den rapporterade (x, y)-positionen för ett objekt representerar dess position relativt en global karta, snarare än i lokala pixelkoordinater.

`gv_client_2025.py` använder samma grundläggande ROS2-nodstruktur med en UDP-server som lyssnar via `parse_packet2025` för det uppdaterade dataprotokollet. Varje paket innehåller en lista detektioner, där varje detektion också inkluderar ett tidsfält `time_msec`. Nodalternativen host, port och tag_id deklareras på samma sätt, men notera att standardporten för den nya versionen kan vara t.ex. 2020. Istället för att samla alla taggar på ett gemensamt topic skapar noden dynamiskt en separat publisher per unikt tag_id. Exempelvis publiceras robot med tagg 5 på `/robot_5/pose`, tagg 7 på

/robot_7/pose och så vidare. Detta möjliggör modulär prenumeration, där varje komponent kan lyssna på en specifik robots topic.

Eftersom den nya GulliView nu tillhandahåller globala koordinater behövs ingen pixelskalanpassning. Däremot kan axelorienteringen skilja sig. I koden byts därför x och y för att matcha ROS-konventionen:

$$x_{\text{map}} = y_{\text{GV}}, \quad y_{\text{map}} = x_{\text{GV}}. \tag{3.15}$$

Eftersom denna nod genererar ett separat ROS2-topic per tagg-ID så snart en detektion registreras, skapas en publisher för just den taggens topic. Ett `PoseWithCovarianceStamped`-meddelande publiceras sedan omedelbart där. Alla meddelanden tidsstämplas med denna detektions egna tidsstämpel, time_msec konverterat till ROS2-tid, och anges i `frame_id=map"`. Filtret tag_id kan fortfarande användas. Om nodens parameter tag_id sätts till något annat än *all*, ignoreras detektioner som inte matchar det ID:t. Denna funktionalitet är gemensam med gv_socket_server.py och möjliggör att köra flera instanser av noden för olika robotar vid behov.

3.3.4 Fusion av sensordata med utökat Kalmanfilter

För att öka tillförlitligheten i positionsuppskattningen av TurtleBot 4 användes sensorfusion baserad på ett Extended Kalman Filter (EKF), implementerat genom ROS 2-paketet `robot_localization` [32]. Målet var att kombinera intern odometri med extern positionsdata från GulliView-systemet och därigenom reducera den totala positionella osäkerheten över tid. I vår implementation konfigurerades ekf_node för att ta emot två huvudsakliga sensorkällor:

- Odometri genererad av TurtleBotens interna sensorer, som publiceras som `nav_msgs/Odometry` på topicen `/odom`.
- Externa positionsuppskattningar från GulliView, konverterade till `geometry_msgs/PoseWithCovarianceStamped` och publicerade på `/gv_pose`.

För att möjliggöra samtidig drift av flera robotar konfigurerades en launchfil, `ekf.launch.py`, med stöd för namespaces. Detta innebar att varje robot kunde köras isolerat med unika topics som till exempel `/robot1/odom` och `/robot2/odom`, vilket förhindrar krockar i ROS2:s kommunikationsstruktur. Parametrarna för EKF specificerades i `ekf.yaml`. Med hjälp av konfigurationsvektorerna `odom0_config` och `pose0_config` kunde vi definiera exakt vilka delar av tillståndsvektorn som skulle tas med i fusionen från respektive sensor.

$$\mathbf{x}_t = \begin{bmatrix} x \\ y \\ \theta \\ v_x \\ v_y \\ \omega_z \end{bmatrix} \tag{3.16}$$

Odometri tillförde information om hastigheter och orientering, medan GulliView gav absoluta positionsuppskattningar utan hastighetskomponent, vilket kompenserar för odometridriftsproblem över tid. För att säkerställa att endast relevanta dimensioner användes aktiverades `two_d_mode: true`, vilket gör att z-led samt roll och pitch ignoreras. Detta är en rimlig förenkling då roboten rör sig på en plan yta. EKF är en iterativ uppskattningsmetod som bygger på en prediktions- och en uppdateringsfas. I vår tillämpning användes följande modeller:

$$\hat{\mathbf{x}}_{t|t-1} = f\Big(\mathbf{x}_{t-1}, \mathbf{u}_t\Big) + \mathbf{w}_t \tag{3.17}$$

$$\mathbf{z}_t = h\Big(\hat{\mathbf{x}}_{t|t-1}\Big) + \mathbf{v}_t \tag{3.18}$$

- $\mathbf{x}_t$ är tillståndsvektorn enligt ekvation 3.16,
- $\mathbf{u}_t$ är styrsignalen,
- $\mathbf{w}_t$ och $\mathbf{v}_t$ är process- respektive mätbrus,
- $f(\cdot)$ är rörelsemodellen och $h(\cdot)$ mätningsmodellen, båda linjäriserade med Jacobianer i varje steg.

Kovariansmatriserna $\mathbf{Q}$ (processbrus) och $\mathbf{R}$ (mätbrus) styr hur stor vikt varje informationstyp ges. I vårt fall tilldelades GulliView-data relativt hög mätkovarians, särskilt vid flerdetektering av en tagg, vilket hanterades genom att parametriskt variera $\mathbf{R}$ beroende på taggposition och kameravinkel. Den filtrerade positionen publiceras som `nav_msgs/Odometry` på topicen `/odometry/filtered`, och en tillhörande `/tf`-transform mellan `odom`- och `base_link`-ramar görs tillgänglig för SLAM- och planeringsmoduler.

En central anledning till att EKF valdes i detta projekt är dess förmåga att hantera icke-linjära samband i både rörelse- och mätmodeller, vilket gör metoden särskilt väl lämpad för mobila robotar. Eftersom både odometridata och externa positioner från GulliView är behäftade med osäkerheter som varierar över tid, erbjuder EKF ett effektivt sätt att kontinuerligt väga samman dessa mätningar till en samlad och filtrerad uppskattning av robotens tillstånd. Till skillnad från enklare metoder som ett klassiskt Kalmanfilter eller medelvärdesfusion klarar EKF av att anpassa sig till varierande mätkvalitet och sensorbrus.

4

Evaluering

För att utvärdera huruvida projektets syfte har uppnåtts med den valda metoden definieras här evalueringsaspekter till detta ändamål. Kapitlet är uppdelat i tre underavsnitt *Evaluering av inomhuslokalisering* , *Evaluering av kooperativ manövrering* och *Jämförelse av intern och extern lokalisering mot fysiska referenser* som var och en innehåller motsvarande utvärderingsaspekter. Dessa aspekter omfattar evalueringskriterium, miljö och plan, som tillsammans ger en helhetsbild över hur arbetet har utvärderats. Här redovisas hur arbetet har bemött frågeställningarna i avsnitt 1.2.

F-1 *Hur har fördröjning och noggrannhet i inomhuslokaliseringssystemet GulliView, i kombination med SLAM genom sensorfusion, förbättrats för att uppfylla realtidskraven vid autonom körning?*

F-1.1 *Hur har GulliViews fördröjning och i sin tur frekvens förändrats?*

F-1.2 *Hur tillförlitlig är positionsdatan från det externa lokaliseringssystemet GulliView?*

F-2 *Hur kan en standardiserad kooperativ manövreringsalgoritm i samband med lokaliseringssystemet GulliView tillsammans med ett distribuerat kontrollsystem bidra till ökat trafikflöde samtidigt som säkerheten bevaras?*

F-2.1 *Vilken inverkan har den standardiserade kooperativa manövreringsalgoritmen på körtiden i olika trafiksituationer?*

F-2.2 *Hur påverkas kollisionsrisken med den standardiserade kooperativa manövreringsalgoritmen vid sammanvävning på en motorvägspåfart?*

F-2.3 *Hur har färdtid och säkerhet utvecklats i samband med införandet av nya kontrollsystem för WifiBotarna, i jämförelse med tidigare implementerade system?*

4.1 Evaluering av inomhuslokalisering

De måttet på prestanda för GulliView som vart i fokus under projektet är fördröjning och i sin tur uppdateringsfrekvens. Det fanns redan metoder att samla in data exempelvis hur lång tid det tog att utföra vissa delar av koden. När GulliView körs så sparas tidsstämpeln en viss del startas och i slutet jämförs detta med nuvarande tidsstämpeln och skrivs till en fil. Problemet är att dessa filer är väldigt långa, ett exempel på detta är en fil från en testkörning på en minut som skapar tio tusentals rader per kamera. För att kunna tolka datan på ett bra sätt har vi utvecklat ett Python skript som finns tillgängligt på BitBucket. Programmet är skapat för Windows och körs på en dator på samma nätverk som GulliView datorn. Skriptet kan med hjälp av SSH koppla upp sig mot GulliView datorn och kopiera loggarna från en specificerad mapp till datorn som kör GulliView logs programmet. Dessa arkiveras sedan i datorn som kör skriptet för senare referens då loggarna i GulliView datorn skrivs över varje körning. Den huvudsakliga funktionen är att kunna tolka denna data till låddiagram så man kan få en bild av hur prestandan är mellan olika versioner av GulliView och lätt kan identifiera vilka funktioner i programmet som behöver förbättras mest. När koden delades upp i olika filer kunde GulliView testköras med en förinspelad film och loggarna jämföras så inga kraftiga förändringar i prestandan skett. Det finns också möjlighet skapa en tidsgraf vilket kan hjälpa till att identifiera förflyttning mellan kameror och hur de påverkar fördröjningen.

Låddiagram (se figurer 5.1 och 5.2) är ett väldigt bra sätt att identifiera spridningen av data. Datan delas in i fyra delar där gränsvärdena är minimivärde, första kvartilen ($Q1$), median (också kallad andra kvartilen eller $Q2$), tredje kvartilen ($Q3$) och maximivärde. Dessa kvartiler kan visuellt läsas av för att tolka data. I låddiagrammen används $1.5 \cdot IQR$-regeln för att identifiera avvikande värden. IQR står för interkvartilintervall och beräknas som skillnaden mellan tredje kvartilen och första kvartilen, det vill säga spridningen för de mittersta 50% av observationerna. Regeln innebär att man definierar ett undre gränsvärde ($Q1 - 1.5 \cdot IQR$) och ett övre gränsvärde ($Q3 - 1.5 \cdot IQR$. Värden som ligger utanför dessa gränser betraktas som avvikare och markeras med punkter. Anledningen till att just faktorn 1.5 används är att det är en vedertagen tumregel som ger en bra balans: den fångar upp värden som tydligt skiljer sig från resten av datamängden utan att överdrivet många observationer klassas som avvikande.

För att utvärdera F-1.1 är fördröjningen från bilden är tagen till att en robot mottar sin position från GulliView intressant. För att evaluera systemets prestanda och isolera det från nätverksstörningar mättes tiden från bilden är tagen tills att koordinaterna är redo att skickas. Detta utfördes genom att spara tidsstämpeln på varje bild och när koordinaterna är redo att skickas kan denna tidsstämpeln jämföras med den nuvarande tiden. Detta loggas och visualiseras sedan med GulliView logs skriptet som låddiagram.

4.2 Evaluering av kooperativ manövrering

För utvärderingen av kooperativa manövrar studeras särskilt två olika aspekter enligt F-2.1 och F-2.2. Dessa två egenskaper studeras eftersom de ger en överblick av systemets förmåga att samarbeta för att lösa olika trafiksituationer, och kan jämföras med andra

projekt som genomförts tidigare enligt F-2.3.

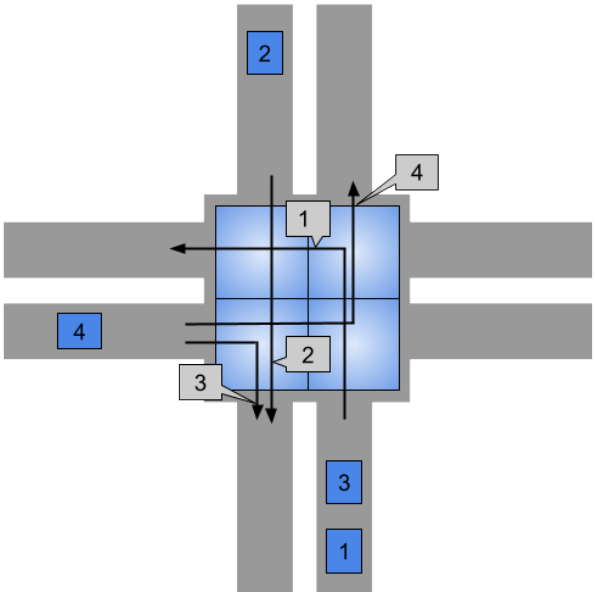
4.2.1 Trafikflöde

Projektet syftar till att studera hur implementationen av en kooperativ manövreringsalgoritm påverkar trafikflödet i olika trafiksituationer. Detta avsnitt av evalueringen avser att behandla frågeställningen F-2.1.

Först så definieras en trafiksituation som ett moment i trafiken som kräver samarbete såsom en korsning eller en rondell. Medan ett trafikscenario syftar på sätt som en trafiksituation kan utfalla, såsom att en bil tar en högersväng eller vänstersväng genom en korsning. I brist på tillgång till fler än två WifiBotar avgränsas projektet till att utföra tester på trafiksituationer med två fordon, även om algoritmen är skapad för att tillåta ett godtyckligt antal. Detta bidrar till att det behövs ett sätt att uppskatta hur systemet förhåller sig till en trafiksituation som involverar flera fordon. Trafikflödet kommer därmed studeras som en kvot där förhållandet ges av två olika tidtagningar. Det första är tiden det tar en robot att köra genom ett trafikscenario med kooperativ manövrering då roboten saktar ner för en annan robot. Detta benämns i fortsättningen kontrollerad fördröjning KF. KF jämförs med sedan med tiden det tar att köra rakt igenom trafikscenariot i samma bana, utan att behöva vänta på något annat fordon, som vi benämner friflödestiden FFT. Detta ger en fördröjningskvot FK, se ekvation 4.1. Varje scenario upprepas tio gånger och denna kvot ges då av det högst uppmätta värdena för KF och FFT. Vidare kan trafikflödet genom en trafiksituation uppskattas för ett godtyckligt antal fordon genom att studera medelvärdet av FK för olika trafikscenarion i samma trafiksituation.

$$\mathrm{FK} = \frac{\max(\mathrm{KF})}{\max(\mathrm{FFT})}, \overline{\mathrm{FK}} = \frac{\sum \mathrm{FK}}{\text{Antal scenarion}} \tag{4.1}$$

Evalueringstesterna för FK utförs på följande vis. Tre olika trafiksituationer studeras. Korsning, sammanvävning vid motorvägspåfart och körning i rondell. För samarbete i en korsning ställs fem scenarion upp. I dessa scenarion kör de två WifiBotarna i 0.3 m/s genom korsningen på planerade rutter både med och utan manövreringsalgoritmen. När robotarna rör sig mellan två definierade start respektive slutpunkter i det globala positioneringssystemet mäts tiden för körningen. Tiden det tar att köra sträckan med algoritmen blir den kontrollerade fördröjningen KF, för roboten som saktar ned. Medan tiden det tar att köra sträckan utan manövreringsalgoritmen blir friflödestiden FFT. Startpositionen för mätningen är bestämd så att roboten färdas i hastigheten 0,3 m/s när mätning påbörjas. Dessa tester upprepas totalt 10 gånger för varje trafikscenario. FK fås sedan genom den största uppmätta FFT och KF för varje scenario. Avslutningsvis blir FK för hela trafiksituationen medelvärdet av summan av alla trafikscenariers fördröjningskvot. De fem olika scenariona för en korsning beskrivs i punktlistan nedan med hjälp av figur 4.1.



Figur 4.1: Numrerade startpositioner och färdvägar genom korsningen.

1. Robotarna startar vid position 1 och 2, och kör väg 1 respektive 2. Roboten som startar vid position 1 behöver vänta tills roboten som startar vid position 2 har passerat korsningen.
2. Robotarna startar vid position 3 och 2, och kör väg 1 respektive 2. Roboten som startar vid position 2 behöver vänta tills roboten som startar vid position 3 har passerat korsningen.
3. Robotarna startar vid position 4 och 2, och kör väg 2 respektive 3. Roboten som startar vid position 2 behöver vänta tills roboten som startar vid position 4 har passerat korsningen.
4. Robotarna startar vid position 3 och 4, och kör väg 1 respektive 4. Roboten som startar vid position 4 behöver vänta tills roboten som startar vid position 3 har passerat korsningen.
5. Roboten som startar vid position 1 och 4, och kör väg 1 respektive 4. Roboten som startar vid position 1 behöver vänta tills roboten som startar vid position 4 har passerat korsningen.

På samma sätt utförs testerna på körning i en rondell. Robotarna framförs i 0,3 m/s men här studeras endast ett scenario på grund av symmetri. Se förtydligande figur 3.10. Roboten i rondellen har förtur och en annan robot närmar sig korsningen och behöver lämna företräde. Scenariot upprepas 10 gånger.

Som sist studerar vi även FK för sammanvävning vid motorvägspåfart där roboten på påfarten har 3 startpositioner som motsvarar 3 olika scenarion. I scenario 1 startar roboten på påfarten 4 meter ifrån den exklusiva zonen där sammanvävning sker, medan i scenario 2 och 3 startar roboten på påfarten på ett avstånd av 4,5 respektive 5 meter ifrån zonen. Roboten på motorvägen startar vid samma position i de tre olika fallen. I samtliga scenarion väntar roboten på påfarten på att den andra roboten ska passera den

exklusiva zonen. Robotarna kör i dessa tester i hastigheten 0,5 m/s och de tre scenariona upprepas 10 gånger. För förtydligande av körsituation se figur 3.8.

4.2.2 Säkerhet

Projektet undersöker även kollisionsrisk vid sammanvävning vid motorvägspåfart. Detta utforskas eftersom det har genomförts tidigare i projektet "Kooperativa manövrar med nedskalade fordon" [6], och är ett sätt att mäta hur systemet presterar säkerhetsmässigt.

Som beskrivit i 2.11 behövs tre fordon för att beräkna CRI men i projektet har det använts två. Lösningen för detta är att simulera ett av de två fordonen på vägen som sammanvävningen sker mellan. Med ett konstant avstånd och samma hastighet som framförvarande fordon simuleras en trafiksituation där bakomvarande fordon på motorvägen håller ett konstant avstånd enligt tresekundersregeln. Detta sätt att simulera ett tredje fordon för att utvärdera CRI genomfördes också tidigare i [6]. Ett problem uppstår med denna metod och det är att det simulerade bakomvarande fordonet ej kan sakta ner då det definieras utifrån hur det riktiga framförvarande fordonet agerar. Detta innebär att vid kollisionsrisk vid sammanvävning på motorvägspåfart studerar denna rapport endast CRI_a. Alltså en potentiell krock med framförvarande fordon, medan avståndet till det simulerade bakomvarande fordonet d_b används för att bestämma det relativa avståndet k_a som är nödvändigt i beräkningen.

Författarna till [6] valde även att utesluta TTC i deras beräkning av CRI_a som då endast baserades på relativa avstånd. I detta projekt inkluderas TTC i beräkningen men genom en modifikation av definitionen. Algoritmen, som beskrivit i metoden 3.2.1, bestämmer hur mycket roboten som anländer senare till den exklusiva zonen saktar ner enligt tiden det tar den robot som anländer först att slutföra sin körning genom den exklusiva zonen. Detta medför ett adaptivt säkerhetsavstånd som är beroende av hastigheten på roboten som anländer först och bestämmer hur mycket roboten som anländer senare behöver sakta ner för att det ska finnas en viss tidsmarginal till kollision. Intresset ligger i att utvärdera CRI vid tillfället då roboten på påfarten genomför sitt filbyte, och eftersom CRI utvärderas med insamlad data efter körningen kan då denna tidsmarginal tas ut som skillnaden i ankomsttid till den exklusiva zonen där det är bestämt att filbytet kommer ske. Denna skillnad kommer då att utgöra tidsmarginalen mellan fordonen och ersätter TTC i ekvation 2.2. För att skilja på dessa uttryck hänvisas i fortsättningen detta som tidsmarginalen TM.

Testet för kollisionsrisken utvärderas som följande. Två WifiBotar ställs upp på vägen som representerar motorvägen respektive påfarten. Robotarna kör sedan i hastigheten 0,5 meter per sekund mot en exklusiv zon där de riskerar att kollidera vid respektive ingångspunkt, se figur 3.8. På detta vis blir skillnaden i ankomsttid mellan de två robotarna tidsmarginalen TM. CRI tas ut efter körningen med position- och hastighetsdata som samlats in under körningen. Det relativa avståndet k_b fås med ett simulerat bakomvarande fordon B som håller ett konstant avstånd på 1,5 meter från fordonet A. Av intresse är CRI värdet vid ingångspunkten för roboten på påfarten, CRI_a. Skillnaden i CRI_a för fordonet C innan manövreringen till vad det är efter manövreringen, vid ingångspunkten, visar då på hur mycket manövreringen har påverkat kollisionsrisken. Tre testfall ställs upp där robotens startposition på påfarten varieras. Ett fall där kollision ej förväntas med ett

avstånd på 5 meter från den exklusiva. Ett fall där kollision kommer att inträffa, med ett avstånd från den exklusiva zonen på 4,5 meter. Och sist ett fall där roboten på påfarten förväntas anlända först men roboten på motorvägen har förtur på grund av prioritet. I detta sista fall startar roboten på påfarten på ett avstånd av 4 meter från den exklusiva zonen.

4.3 Jämförelse av intern och extern lokalisering mot fysiska referenser

För att utveckla ett robust lokaliseringssystem som integrerar både intern och extern positionsdata krävs en god förståelse för datakällornas tillförlitlighet. I ett sensorfusionssystem vägs information från flera oberoende källor samman, där varje enskild mätning förses med en kovariansmatris som beskriver osäkerheten i positionen. Denna osäkerhetsmodellering är central för att algoritmen ska kunna göra en statistiskt välinformerad uppskattning av robotens faktiska läge. Det är därför avgörande att varje datakällas kovariansmatris återspeglar den verkliga felmarginalen i mätningarna. Mot denna bakgrund uppstår följande nyckelfråga:

> *F-1.2 Hur tillförlitlig är positionsdatan från det externa lokaliseringssystemet GulliView?*

Under projektets gång användes två olika versioner av GulliView: en äldre version från 2023 och en nyare version som i projektets slut nådde tillräcklig funktionell nivå för praktisk användning. Den äldre versionen valdes initialt för sin stabilitet och förmåga att skicka positionsdata via UDP, men den visade sig ha flera begränsningar. Bland annat noterades att dess koordinatsystem inte var metrisk i skala – en förflyttning på en meter i verkligheten motsvarade ungefär 1300 koordinatenheter i GulliViews system. Detta tyder på att koordinaterna kan representera pixelvärden snarare än fysiska avstånd i millimeter.

Ytterligare tester avslöjade en systematisk avvikelse på 200–300 koordinatenheter mellan olika kameror när roboten samtidigt var synlig för flera kameror. Dessa fel pekar mot bristfällig kamerakalibrering och mot att systemet sannolikt är optimerat för objekt nära golvnivå, medan robotens markör (AprilTag) i detta fall är placerad cirka 35 cm ovanför golvet.

4.3.1 Indirekt feluppskattning genom jämförelse med SLAM

Eftersom det är praktiskt svårt att exakt mäta robotens absoluta position i rummet med hög precision, valdes en indirekt metod för att uppskatta GulliViews felmarginal. GulliViews positionsdata jämfördes med robotens interna uppskattning baserad på SLAM. Denna metod möjliggör insamling av ett stort antal datapunkter över hela testområdet och möjliggör en robust statistisk jämförelse mellan de två positionskällorna.

För att sätta GulliViews koordinater i relation till SLAM-systemets användes en linjär transformationsmodell, se ekvation 4.2.

$$\begin{aligned} x_{\text{SLAM}} &= a_x x_{\text{GV}} + b_x \\ y_{\text{SLAM}} &= a_y y_{\text{GV}} + b_y \end{aligned} \tag{4.2}$$

Parametrarna a_x, a_y och b_x, b_y skattades med hjälp av linjär regression på insamlade datapunkter. Modellen användes sedan för att visualisera robotens körbana enligt både SLAM och GulliView. Vidare användes a_x, a_y för att översätta koordinaterna från det gamla GulliView-systemet till den bästa linjära approximationen i metrisk skala.

Samma testprocedur tillämpades även på den nya versionen av GulliView för att möjliggöra en direkt jämförelse mellan de två systemen.

4.3.2 Jämförelse mot verkliga mått

För att validera både GulliView- och SLAM-data mot verkliga rörelser utfördes även tester med fysiska mätningar. Roboten kördes mellan fyra markerade hörn som tillsammans bildade en rektangel med måtten $6{,}0 \times 2{,}5\,\text{m}$. De uppmätta sträckorna mellan punkterna jämfördes med motsvarande positionsförändringar enligt respektive system. På så sätt utvärderades hur väl SLAM och GulliView lyckades estimera förflyttningsavstånd i praktiken. Resultaten presenteras i stapeldiagram som visar de procentuella mätfelen (se avsnitt 5.3). För mer detaljerade värden och sammanställningar, se appendix A.

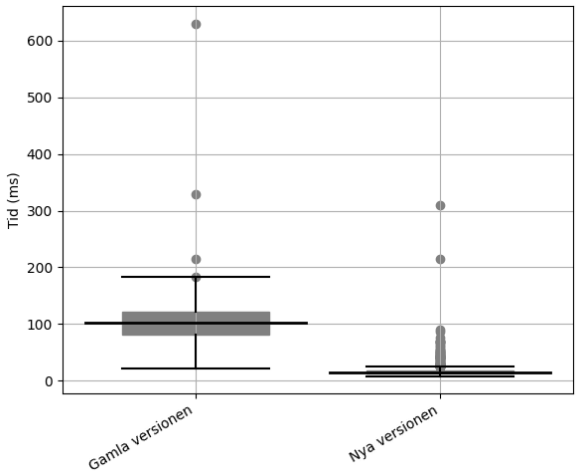
5

Resultat

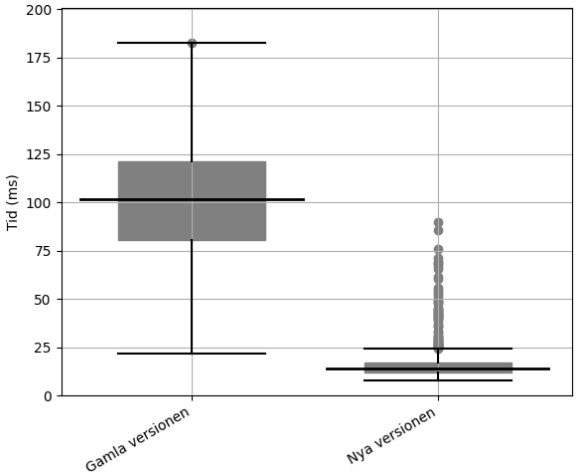
Resultaten är organiserade enligt område och arbetsgrupper men överlappar ibland mycket. Målet är att besvara forskningsfrågorna som presenteras i kapitel 1.2 och vidare utvecklas i kapitel 4. För att uppnå dessa resultat har arbetet utförts enligt kapitel 3.

5.1 GulliViews hastighet

Resultatet för GulliView presenteras för att besvara forskningsfrågan F-1.1 angående fördröjning och prestanda av systemet. Fördröjningen av exekveringscykel definieras som tiden det tar från att tråden får nästa bild till att alla funktioner har körts och meddelandet om detektioner är redo att skickas. I figur 5.1, och dess inzoomade variant figur 5.2, visas den beräknade fördröjningen för den äldre versionen och den nya versionen av GulliView med uppdaterad distorsionskorrigering och global lokalisering. Några avvikande värden förekommer, de flesta av dessa är ett resultat av uppstarten av programmet. Den nya versionen har betydligt fler avvikande värden inom 75 ms och under. I tabell 5.1 kan de exakta värdena ses. Där visas det minsta, största samt median, undre kvartil och övre kvartil. Medianen för fördröjningen har gått från 102 ms till 14,0 ms. Vilket är en minskning på 86% i jämförelse med den äldre versionen.



Figur 5.1: Låddiagram över fördröjning av en exekveringscykel av Fast Thread från att bilden tags tills meddelande skickas.

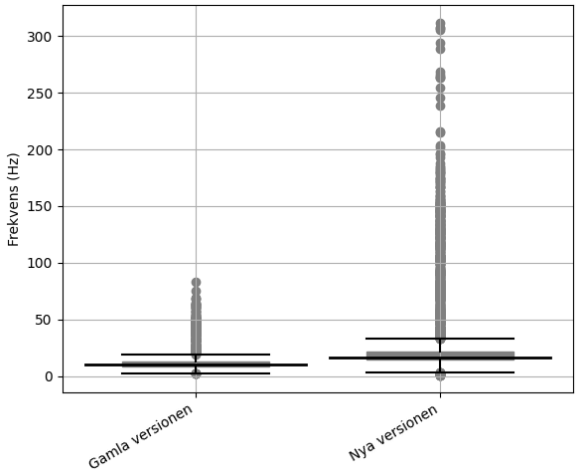


Figur 5.2: En inzoomad variant av låddiagram från data som representeras i figuren 5.1, för ökad detaljrikedom.

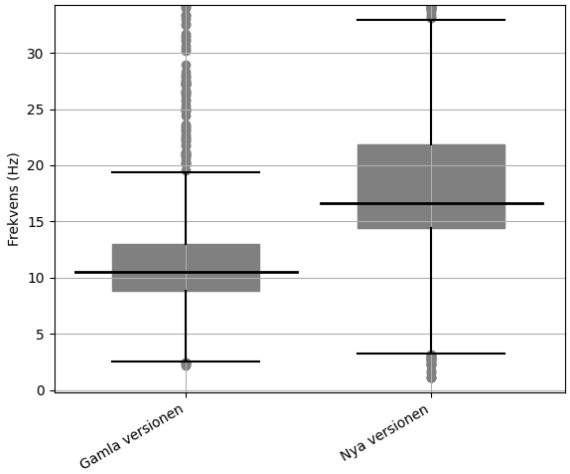
Värde (ms)	Minsta (exkl. avvikare)	Undre kvartil	Median	Övre kvartil	Högsta (exkl avvikare)
Gamla versionen	21,6	80,5	102	121	182
Nya versionen	7,90	12,1	14,0	17,0	24,4
Förbättring	63%	85%	86%	86%	87%

Tabell 5.1: Jämförelse av fördröjning av Fast Thread från att bilden tas tills koordinater är redo att skickas mellan gamla och nya GulliView.

Den uppmätta uppdateringsfrekvensen av den äldre och nya versionen kan ses i figur 5.3, och dess inzoomade variant figur 5.4. De avvikande värdena från nya systemet är betydligt större än från den äldre versionen och sträcker sig upp till över 300 Hz i vissa fall medan den äldre ger maximalt ungefär 80 Hz. De exakta värdena kan ses i tabell 5.2. Enligt tabellen är medianen för det äldre systemet 10,5 Hz och för nya systemet 16,6 Hz. Vilket motsvarar en ökning med 58%. Det minsta värdet skiljer sig inte så mycket åt, 30%, medan de högsta värdet har största förbättringen från 19,3 Hz till 33,0 Hz, 71%.



Figur 5.3: Låddaiagram över uppdateringsfrekvens av Fast Thread mätt genom antalet exekveringscykler per sekund.



Figur 5.4: En inzoomad variant av låddiagrammet som representeras i figur 5.3, för ökad detaljrikedom.

Värde (Hz)	Minsta (exkl. avvikare)	Undre kvartil	Median	Övre kvartil	Högsta (exkl avvikare)
Gamla versionen	2.51	8.82	10.5	13.0	19.3
Nya versionen	3.26	14.4	16.6	21.8	33.0
Förbättring	30%	63%	58%	68%	71%

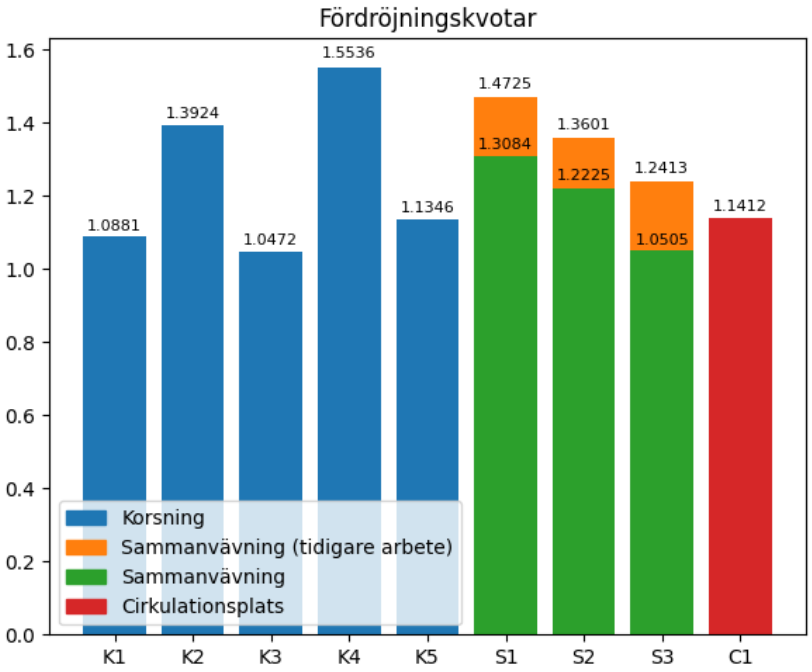
Tabell 5.2: Jämförelse av uppdateringsfrekvens av Fast Thread mellan gamla och nya GulliView.

5.2 Kooperativ manövrering

För att besvara forskningsfråga F-2.1 och F-2.3 beskrivs i figur 5.5 fördröjningskvoten (FK) för alla evalueringsexperiment. Figuren innehåller ett stapeldiagram där vardera stapel representerar FK för varje trafikscenario. Staplarna för evalueringsresultatet för korsning är i figuren markerat med K följt med körscenariots numrering. På samma vis är trafiksituationerna med sammanvävning av körfält och körning genom en cirkulationsplats markerat med S respektive C. För sammanvävning av körfält (S i figuren) har även resultatet för evalueringen av de tidigare arbetet [6] noterats. För varje scenario har maxvärdet av friflödestiden (FFT) och den kontrollerade fördröjning (KF) för de tio körningarna använts i beräkningarna av FK enligt ekvation 4.1. I figur 5.5 har varje värde för FK angetts ovanför respektive stapel.

För de fem korsnings scenarierna visar resultatet hur en ökning av körtiden har uppstått för alla fall. Där den procentuella tids ökningen relativt FFT för körning genom en korsning med denna algoritm sträcker sig mellan $4.72\% - 55.36\%$ med ett genomsnitt $\left(\overline{\text{FK}}\right)$ på 24.32%, se tabell 5.3. För evalueringen av systemet vid sammanvävningen beräknades den procentuella fördröjningen till ett spann mellan $5.05\% - 30.84\%$ med ett medelvärde på 19.38%. I jämförelse med den tidigare implementationen har denna fördröjning minskats med det nya systemet.

Enligt dessa beräkningar (se figur 5.5) leder den äldre implementationen till en ökad fördröjningar upp till 18.16% vid kröning i situationen. Den genomsnittliga ökningen kan beräknas med värdera i tabell 5.3 till 13.75%. För implementeringen av cirkulationsplatsen har den procentuella fördröjningen beräknats till 14.12%, se figur 5.5 och tabell 5.3.



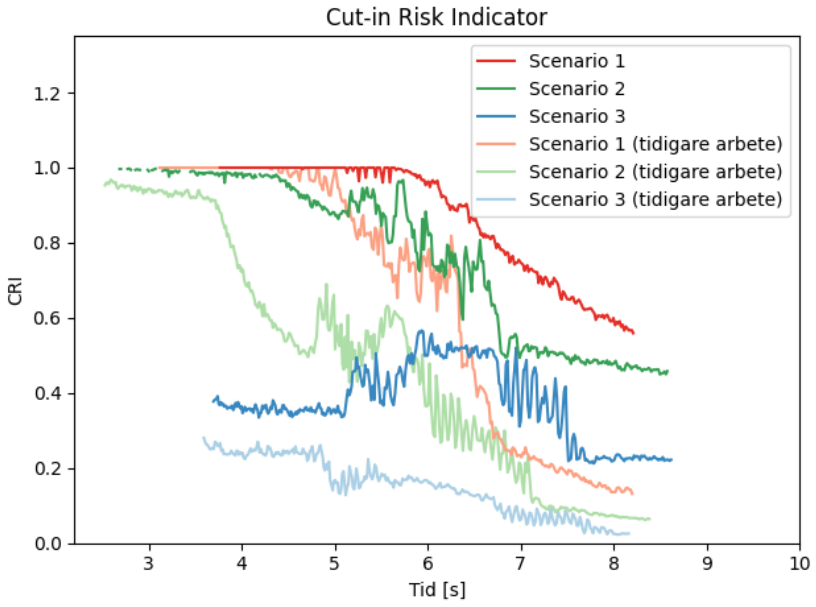
Figur 5.5: Diagram över beräknade värden av FK för varje evalueringsscenario.

Trafiksituation	Korsning	Sammanvävning	Sammanvävning (tidigare arbete)	Cirkulationsplats
$\overline{\mathbf{FK}}$	1.2432	1.1938	1.3580	1.1412

Tabell 5.3: Beräknade värden för medelvärdena av FK för de olika trafiksituationerna.

För att besvara forskningsfråga F-2.2 och F-2.3 presenteras i figur 5.6 hur värden på CRI förändras under körningarnas förlopp. Figuren innehåller beräknade värden för varje evalueringsscenario med sammanvävning av körfält. Tre av dessa kurvor beskriver evalueringen av detta projektets implementering och de resterande tre beskriver för samma scenarion det tidigare arbetes implementering [6]. För att ge en förenklad bild över förloppet beskrivs i figuren medelvärdet av CRI för varje scenario, över tid. Då beräkningen av CRI innefattar TTC (se ekvation 2.2) som är beräknat utifrån avståndet till den exklusiva zonen för båda robotarna, är CRI endast beräknat för ett visst intervall. CRI beräknats först då båda robotarna är inom upptäcksavstånd för den exklusiva zonen, och fram till att en av dem bestigit området. Efter denna tidpunkt är sammanvävningen fullbordad och de båda robotarna på samma väg.

Figuren visar dock tydligt för alla scenarion att värdet på CRI sjunker då robotarna närmar sig den exklusiva zonen. Detta betyder att denna uppskattade risken för kollision sjunker då manövreringsalgoritmen kontrollerar situation och att säkerheten upprätthålls. För scenariona från 1 till 3 ökar det relativa avståndet mellan robotarna då en av robotarnas startpositions förflyttas längre ifrån den exklusiva zonen. Detta medför ett lägre värde för CRI (se figur 5.6) då kollisionsrisken sjunker med större avstånd mellan robotarna, se ekvation 2.2. Den äldre algoritmen skapar ett större avstånd mellan robotarna vid sammanvävningen, vilket medför ett lägre värde på CRI, se figur 5.6.

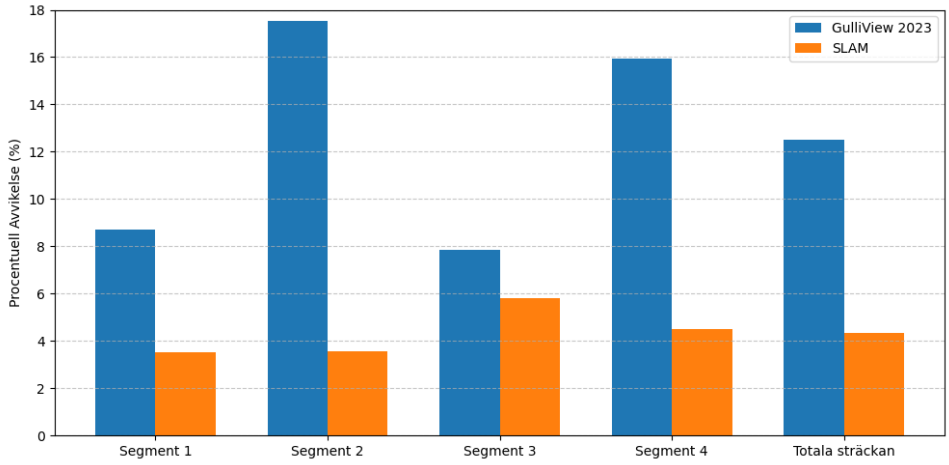


Figur 5.6: Genomsnittliga värden av CRI för alla scenarion av sammanvävning.

5.3 Avvikelser mellan intern och extern lokalisering

En jämförande evaluering har genomförts mellan robotens interna SLAM-lokalisering, den äldre versionen av det externa kamerasystemet GulliView (GV) samt den nya förbättrade GulliView-versionen. Syftet är att kvantitativt bedöma deras positionsnoggrannhet, stabilitet och eventuella systematiska förskjutningar i positioneringsdata. Testerna utfördes på TurtleBot 4-plattformen, som fick färdas en rektangulär bana med kända segmentlängder (6,0 m respektive 2,5 m). Det gjordes även tester där roboten fick köra fram och tillbaka längs testbädden. Robotens bana enligt SLAM användes som referensram, och GulliViews data, både den äldre och den nya versionen, transformerades linjärt för att passa samma referens innan jämförelser gjordes. En SVD-transformering genomfördes på insamlad data för att anpassa skalan, eftersom GulliView (speciellt den äldre versionen) använde ett icke-metriskt koordinatsystem. SVD (Singular Value Decomposition) är en matrisfaktorisering som används för att identifiera och rangordna de viktigaste riktningarna i data. En verklig meter motsvarade cirka 1 300 längdenheter i den äldre GulliView-koordinaten, och dessutom upptäcktes en konstant förskjutning på 200–300 längdenheter mellan olika kameror, vilket indikerar bristfällig kalibrering och systematiska förskjutningar i positionerna.

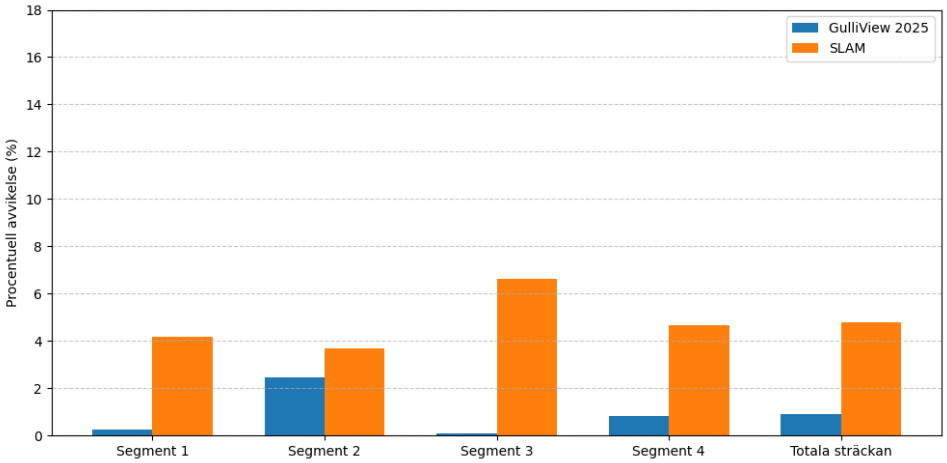
Resultaten visar tydliga skillnader i mätnoggrannhet mellan systemen. Den äldre GulliView-versionen uppvisar genomgående större positioneringsfel jämfört med SLAM. Som framgår i tabellen 1, i bilagorna, hade gamla GV i genomsnitt cirka 0,4–0,5 m absolutfel per segment (t.ex. $\sim 0,52$ m fel över en 6,0 m-sträcka), medan SLAM:s motsvarande fel låg kring 0,1–0,3 m. Detta motsvarar ungefär 8–18 % relativt segmentens längd för GV, mot 2–5 % för SLAM, se figure (5.7). GV:s fel var dessutom systematiskt positiva, det vill säga distanserna överskattades konsekvent (GV rapporterade längre sträcka än den faktiska), vilket tyder på en skalförskjutning i systemet.



Figur 5.7: Procentuell avvikelse för GulliView från 2023. Segment 1 och 3 motsvarar sträckor på 6.0 m, medan segment 2 och 4 är 2.5 m långa.

Den nya GulliView-versionen har däremot kraftigt förbättrad noggrannhet. Enligt tabell 2, i bilagorna, var medelfelet för nya GV på 6,0 m-segment endast 0,01–0,02 m, att

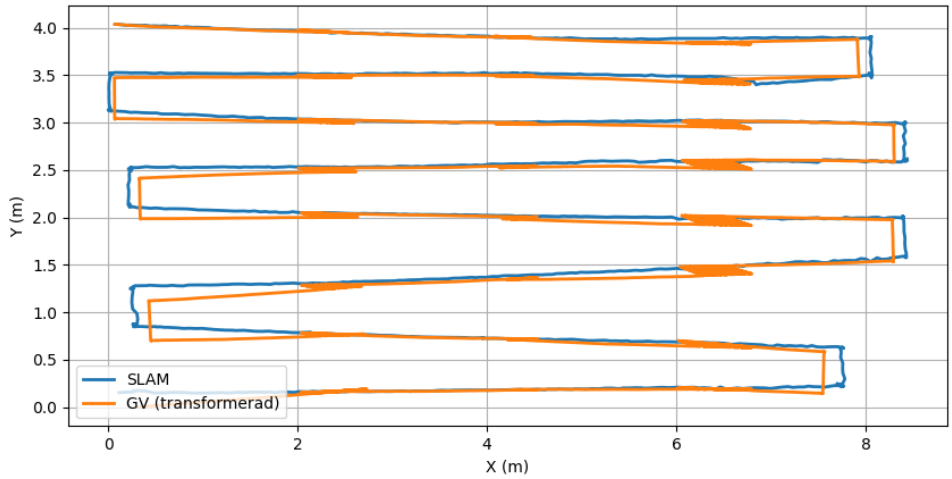
jämföra med SLAM:s 0,25–0,40 m på samma sträckor. Även på kortare 2,5 m-segment låg GV:s fel i storleksordningen 0,02–0,06 m (cirka 1–2 % av sträckan), mot SLAM:s 0,09–0,12 m (cirka 4 % av sträckan), se figure (5.8). Detta innebär att extern lokalisering med nya GulliView är mer precis än den interna SLAM, med avvikelser på endast några få centimeter. Till skillnad från den äldre versionen saknar nya GulliView de tidigare skalfelen och systematiska förskjutningarna.



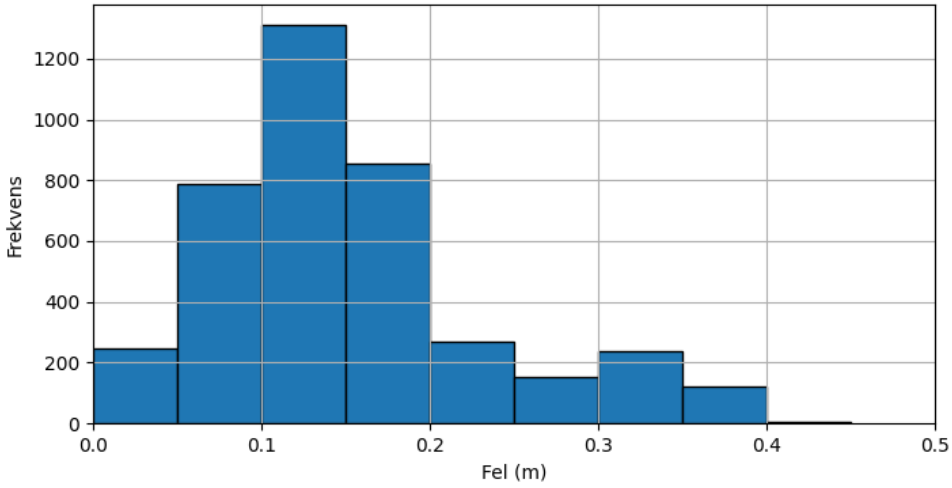
Figur 5.8: Procentuell avvikelse för GulliView från 2025. Segment 1 och 3 motsvarar sträckor på 6.0 m, medan segment 2 och 4 är 2.5 m långa.

Förutom bättre noggrannhet uppvisar nya GulliView även högre stabilitet och konsekvens i positionsdatan. I experiment med TurtleBot 4 observerades att positionerna från den äldre GulliView fluktuerade mer och kunde visa tydliga språng när roboten bytte mellan kamerornas synfält, på grund av tidigare nämnda kalibreringsproblem. Detta syns i figur 5.11. Histogrammet i figur 5.10 (äldre GV vs. SLAM) indikerar en relativt bred felfördelning, där skillnaden i rapporterad position ibland nådde 0,4–0,5 m. Motsvarande histogram för nya GulliView, figur 5.12 är betydligt smalare, med huvuddelen av felen koncentrerade inom $\pm 0,1$–0,2 m och inga extrema avvikelser över $\sim 0,3$ m. Dessutom var mätningarna mycket reproducerbara mellan upprepade testkörningar, den nya GulliView-versionen gav nära på identiska resultat i tre efterföljande tester (skillnader på bara millimeternivå mellan testerna), medan den äldre versionen återkom med samma systematiska fel.

Sammanfattningsvis är det tydligt att nya GulliViewär mer tillförlitligt än både gamla GulliView och TurtleBotens egna interna lokaliseringssystem(SLAM), vilket svarar på frågan F-1.2, Hur tillförlitlig är positionsdatan från det externa lokaliseringssystemet GulliView?

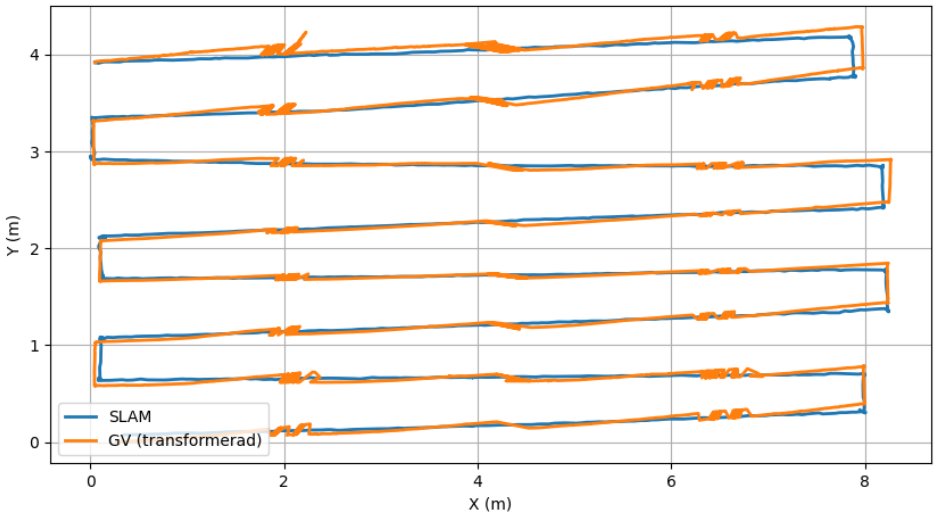


Figur 5.9: Plot som visar vägen TurtleBot tar då SLAM är referense datan och gamla GulliView(GV) är transformerad för att passa SLAM:s data.

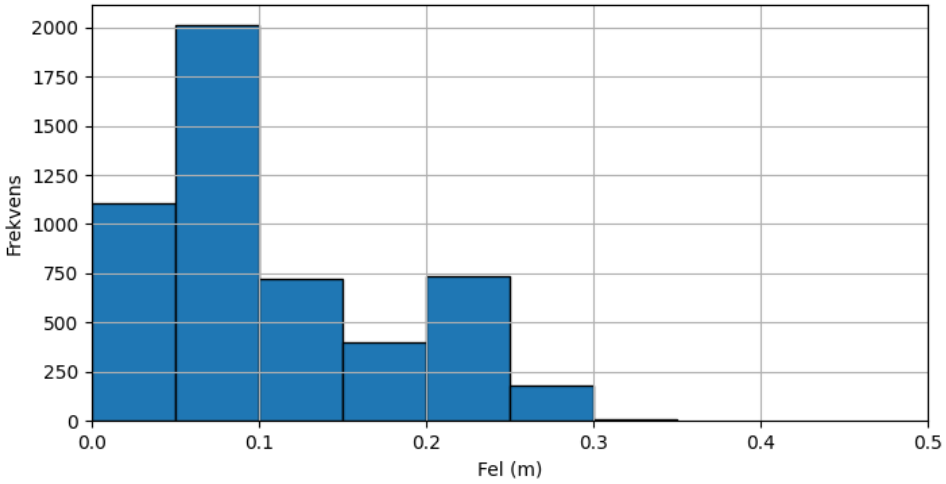


Figur 5.10: Histogram som visar skillnad i distans mellan punkter av SLAM och gamla GulliView, data baserad på figur (5.9)

.



Figur 5.11: Plot som visar vägen turtlebot tar då SLAM är referense datan och nya GulliView(GV) är transformerad att passa SLAM:s data.



Figur 5.12: Histogram som visar skillnad i distans mellan punkter av SLAM och nya GulliView, data baserad på figur (5.11)

.

6

Diskussion

I detta avsnitt diskuteras resultaten i relation till projektets syfte och frågeställningar, med fokus på hur väl de uppnådda resultaten överensstämmer med förväntningarna. Avsnittet belyser styrkor och begränsningar i det genomförda arbetet, samt ger en tolkning av resultatens betydelse. Avslutningsvis presenteras de huvudsakliga slutsatserna och möjliga riktningar för framtida utveckling eller forskning.

6.1 Stabilisering och strukturering av GulliView

GulliView är ett långsiktigt projekt som har utvecklats under mer än tio år vid Chalmers tekniska högskola [33]. Under hösten 2024 genomgick systemet omfattande förändringar inom ramen för ett projekt [3], där kodbasens omfattning ökade från cirka 1000 rader till omkring 3700 rader. Denna expansion ledde initialt till en komplex och svårhanterlig struktur. Projektets mål inkluderade därför att förbättra kodens struktur och läsbarhet. Vid projektets början var systemet instabilt och svårt att analysera, med kod utspridd i en enda stor fil. Avsevärd tid investerades i att modulärt strukturera koden i flera filer, vilket resulterade i att studenter och forskare numera enklare kan navigera och förstå systemets funktioner, såsom beräkningen av objektens hastigheter. Även om ytterligare förbättringar av struktur och läsbarhet är möjliga, utgör detta arbete en stabil grund för framtida utveckling.

Under projektets gång användes inspelat videomaterial i kombination med GulliView logs för att effektivt evaluera ändringar. Real-tidsdata från kamerorna testades däremot i begränsad omfattning. När realtidsinspelningar väl användes, upptäcktes en instabilitet där koordinatdata inte skickades korrekt över nätverket, vilket medförde problem vid utvärderingen av sensorfusion mellan GulliView och SLAM. Ursprungligen var ambitionen att mäta både fördröjning och noggrannhet, men på grund av identifierade stabilitetsproblem prioriterades felsökning och åtgärder. Efter preliminära tester konstaterades att stabiliteten avsevärt förbättrades genom att reducera upplösning och bildfrekvens från 4K vid 60Hz till Full HD vid 30Hz. Dessutom implementerades en temporär lösning för att säkerställa överföring av AprilTag 4-7 till robotarna, vilket möjliggjorde evaluering av sensorfusion vid lägre upplösning och frekvens.

På grund av ovan nämnda prioriteringar kunde en planerad noggrannhetsutvärdering ej utföras. Metoden för denna evaluering baseras på en punktmatta och ett skript utvecklat under projektet hösten 2024 [3]. Då avståndet mellan punkterna på mattan endast

är 44mm, är metoden begränsad i sin förmåga att identifiera gradvisa systematiska fel som ackumuleras över längre avstånd. Utöver detta utförs noggrannhetstesterna med ett separat skript, vilket endast evaluerar transformationsmatriserna för att omvandla bildkoordinater till världskoordinater.

6.1.1 Slutsats GulliView

Sammanfattningsvis representerar den nuvarande versionen av GulliView det mest avancerade stadiet hittills, med avsevärda förbättringar i prestanda, framför allt avseende fördröjning och uppdateringsfrekvens enligt resultaten i avsnitt 5.1. Trots detta finns ytterligare utrymme för framtida optimeringar och förbättringar av systemets prestanda. Målet att utvärdera systemets noggrannhet kunde däremot ej fullföljas på grund av de tidigare nämnda tekniska utmaningarna med stabilitet.

6.2 Kooperativ manövrering

Resultatet från de tre trafiksituationerna som kan ses i figur 5.5 och tabell 5.3 visar på att den kooperativa manövreringen ökar tiden det tar att genomföra manövreringen genom scenarierna i de olika trafiksituationerna med allt ifrån 4,7 till 55,6%, med en genomsnittligt ökning beroende på trafiksituation på 14,1 till 24,3%. Dessa fördröjningar är beroende på hur lång tid roboten behöver vänta på att roboten som anländer först till den exklusiva zonen att manövrera genom den. Att genomföra många fler tester hade varit intressant för att få ett bättre uppskattat genomsnittligt värde för trafiksituationerna. I dessa testfall medförde manövreringen också att inga kollisioner mellan fordonen inträffade och således kan fördröjningen ses som en bekostnad för att upprätthålla säkerhet i trafikscenarierna.

Resultatet på utvärderingen av säkerhet vid sammanvävning på motorvägspåfart, se figur 5.6, visar på att systemet minskar risken för kollision. Från insamlad data syns det att det kvarstår en risk även efter manövrering vid ingångspunkten. Detta är beror på användandet av tidsmarginal, TM som ersättning för TTC, se avsnitt 4.2.2. Ett högre värde på TM ger ett lägre CRI värde på liknande vis som TTC men på grund av hur TTC är definierat så innebär vårt scenario, där hastigheten för fordon C är lägre eller samma som för fordon A, att TTC inte är definierat. Detta medförde att utvärdera med TM istället för TTC ansågs mer lämpligt. Anledning till varför utvärdering med TTC inte var möjligt är på grund av hur tidsbokningen är implementerad i systemet, uppdateringsfrekvensen från lokaliseringssystemet samt projektets avgränsning till två WifiBotar som innebar att det endast fanns möjlighet att studera en av de två ingående delarna i CRI, nämligen CRI_a.

Man kan se i figur 5.6 att algoritmen minskar CRI värdet i alla körfall och vid upprepade tester kolliderade fordonen aldrig. Detta resultat kan därmed ses som förväntat. Men för att uppskatta hur väl systemet upprätthåller säkerheten i dessa fall kan vi jämföra med tidigare arbeten som är genomförda i laboratoriet. Genom att använda det program vi har skapat men låta styrningen hanteras av implementeringen i [6] fås data för samma trafikscenario längs samma körbana. Skillnaden i resultatet visar att tidigare implemen-

tation lyckades minska CRI mer än vad vi har lyckats med, men detta baseras nästan enbart på att i [6] användes ett dubbelt så stort avstånd, d_a och d_b. Detta bidrar till både en större TM, och ett stabilare värde på det relativa avståndet k_a i beräkningen av CRI. Samtidigt kan vi se i figur 5.5 att den nya algoritmen inte bidrar till lika stor fördröjning som tidigare arbete då den äldre metoden saktar ner fordonet mycket mer.

I figur 5.6 finns en del större steg och svängningar, som exempelvis mellan tidpunkterna 5.2 s och 6.8 s för scenario 1, eller 5.0 s och 7.5 s för scenario 3 eller 4.8 s och 6.5 s för scenario 2 (tidigare arbete). Detta fenomen beror på att detta system har utvärderats med en äldre version av GulliView vars synfält överlappar något och positionering kan variera för samma punkt. Då roboten befinner sig inom synfältet av två kameror skickar GulliView robotens beräknade position emellanåt utifrån vardera kamera. Detta får robotens uppfattning om sitt avstånd till den exklusiva zonen att variera vilket medför de större svängningarna i figuren. Dessutom om de två robotarna befinner sig i olika kameraområden medför detta också ett konstant fel i avståndsberäkningen av d_a. Detta resulterar i de större stegen figuren 2.2. Hänsyn till detta togs då intresset ligger i skillnaden i CRI_a värdet innan jämfört med efter manövreringen, och i både början och slutet av grafen så befinner sig robotarna i samma kameraområde.

6.2.1 Slutsats kooperativa manövrar

Vi har lyckats skapa en kooperativ manövreringsalgoritm som är implementerat distribuerat över WifiBot plattformarna och som är standardiserad så att den kan appliceras i många olika trafiksituationer. Algoritmen beter sig liknande till tidigare projekt i laboratoriet i avseende på säkerhet och har samtidigt god förmåga att styra robotarna i dessa trafiksituationer utan att bidra till stora fördröjningar. Algoritmen innebär att fortsatt arbete i laboratoriet kommer att ha tillgodo ett verktyg som kan användas för styrning av robotarna där de delar yta och som tillser att de ej kolliderar men samtidigt tar sig igenom trafiksituationen på ett effektivt sätt.

6.3 Integrering av intern och extern lokalisering

Arbetets ursprungliga målsättning var att integrera TurtleBotens interna positionsuppskattning med det externa lokaliseringssystemet GulliView, i syfte att uppnå en mer robust och tillförlitlig realtidslokalisering. Detta bedömdes vara centralt för att möjliggöra stabil autonom körning och förbättra kartläggningen av miljön i Hermes-plattformen. Under projektets gång identifierades dock flera praktiska och tekniska utmaningar, vilket ledde till ett skifte i fokus. Istället för att färdigställa ett komplett sensorfusionssystem, kom arbetet att lägga grunden för fortsatt utveckling genom att utvärdera tillförlitligheten i GulliViews data och etablera en arkitektur som möjliggör framtida flerrobot-system.

6.3.1 GulliViews potential för extern lokalisering

I utvärderingen (se avsnitt 4.3 och 5.3) framkom tydliga skillnader mellan den äldre och den nyare versionen av GulliView. Den äldre versionen från år 2023 saknade metriskt koordinatsystem och visade systematiska fel, särskilt i överlappande kamerazoner. Trots

detta erbjöd den ett stabilt flöde av positionsdata över UDP, vilket underlättade integrationen.

Den nya versionen, med stöd för metriska enheter och förbättrad kalibrering, visade i figur 5.8 betydligt lägre felvärden än den äldre versionen, figur 5.7. Vid körningar mellan kända fysiska punkter kunde systemet uppvisa hög överensstämmelse med verkliga avståndsmätningar. Detta tyder på att GulliView, med rätt kalibrering och konfiguration, har stor potential att höja noggrannhet och precision i robotens lokaliseringsförmåga. En av GulliViews främsta styrkor är att dess koordinater är absoluta och refererar till en specifik fysisk position i rummet. Det innebär att systemet inte är känsligt för ackumulerad drift över tid, vilket annars är en känd begränsning hos interna lokaliseringsmetoder som SLAM. Även om GulliViews mätvärden inte alltid är helt exakta, erbjuder de en stabil referenspunkt som har stor inverkan på det integrerade systemets robusthet och kan effektivt kompensera för SLAM:s gradvisa avvikelse från robotens verkliga position.

Den linjära transformation som användes för att översätta de gamla GulliView-koordinaterna till metrisk skala (se figur 5.7) användes också inom mottagaren för GulliViews datapaket. Innan informationen publiceras inom ROS-nätverket är det viktigt att den följer ROS-standarder. Genom upprepade tester kunde denna linjära skalning bestämmas och implementeras.

6.3.2 Slutsats intern och extern lokalisering

Trots att projektet inte omfattade en fullständig implementation av ett avancerat sensorfusionssystem med Extended Kalman Filter (EKF), lades en viktig grund för vidare utveckling. Den grundläggande konfigurationen av `robot_localization` gav både praktisk insikt i hur olika komponenter samverkar och tydliggjorde vilka faktorer som påverkar systemets kvalitet. Det arbete som utförts kan därmed fungera som både startpunkt och referens för framtida utveckling och förbättringar.

Sammanfattningsvis har detta projekt bidragit med viktiga insikter kring hur ett externt lokaliseringssystem kan samverka med robotens interna sensorer. Trots att den fullständiga sensorfusionen inte färdigställdes, har arbetet belyst centrala tekniska krav och lagt en solid grund för framtida förbättringar av realtidslokalisering i inomhusmiljöer.

6.4 Teknikens möjligheter och begränsningar i samhället

Systemen uppnår tillräckligt hög noggrannhet, hastighet och säkerhet för att kunna appliceras i ett flertal områden med behov för inomhuslokalisering och autonom körning. Detta kan bidra till effektivisering av fabriker och lagervaruhus vilket därmed kan stärka utvecklingen och ekonomin. De autonoma fordonen har även potential att ersätta monotona och slitande arbetsuppgifter vilket lämnar behagligare uppgifter åt arbetare för en mer trivsam arbetsmiljö. Denna teknologins enda påverkan på miljön är material- samt energiframställningen för de tekniska komponenter som krävs. Då fordonen är små och batteridrivna samt kamerasystemet är effektivt och använder få komponenter, leder detta

till en material- och energieffektiv implementering av inomhuslokaliseringssystem.

Det finns även möjlighet för att applicera och anpassa systemet för autonoma fordon inom transportsektorn. Däremot förekommer en del sociala risker i överlämning av kontroll till datorn. Dessa system, vilket endast testats inomhus och inom isolerade fall, riskerar att göra opålitliga val vid unika trafiksituationer. Mer utveckling krävs innan all kontroll kan överlåtas utan möjlighet för mänskligt ingripande.

6.5 Framtida arbete

I detta avsnitt föreslås konceptuella förbättringar i tre centrala områden för systemets fortsatta utveckling och utvärdering. Eftersom de enskilda delsystemen nu nått en mogen utvecklingsnivå, föreslås även en sammanslagning av dessa för att etablera en sammanhållen och synergistisk helhetslösning.

6.5.1 GulliView

Med avseende på resultatet i avsnitt 5.1 har GulliView förbättrats under projektets gång, men det finns fortfarande mer potential. Till att börja med är systemet instabilt, detta är högst prio i ett framtida arbete. Utöver detta skulle kodstrukturen kunna se ytterligare omstrukturering. Några funktioner är oerhört långa och kan delas upp i mindre funktioner för att lättare kunna utveckla systemet i enlighet med syftet i avsnitt 1.1. Allmänna förbättringar på GulliView som är värda att undersöka är att frånkoppla AprilTag sökningen från kamerorna. För varje kamera körs en sökningstråd, därför klarar GulliView endast en AprilTag per kamera. Hade det istället varit en sökningstråd för varje robot hade systemet klarat av fler robotar samtidigt och det skulle möjliggöra smidigare övergångar mellan kamerorna. I nuläget måste varje kamera köra en noggrann genomsökning för att hitta en AprilTag innan snabba sökningen kan köras. Det nya implementerade globala koordinatsystemet kräver en fullständig evaluering för att bedöma hur noggrant systemet är.

6.5.2 Kooperativ manövrering

Den nuvarande algoritmen är fortfarande en relativt simpel implementering som reglerar hastigheten beroende på tillståndet av andra robotar. Denna modell har varit tillräcklig för att utforska hur väl implementationen kan fungera för laboratoriet men kan utvecklas för mer noggrann kontroll över fordonet. Särskilt att styra robotarna med avseende på att reglera för acceleration hade medfört ytterligare kontroll. Vidare bör implementationen av denna algoritm kopplas till den nya versionen av GulliView som detta arbete har implementerat och använda manövreringsalgoritmen med den. Den uppdaterade versionen av GulliView har möjligheten att med dess högre uppdateringsfrekvens och säkrare positionsdata, särskilt mellan kameraområden, ge tillräckligt mycket data för att estimera hastigheten och accelerationen på fordonen. Vidare hade denna implementation inneburit att algoritmen hade kunnat använda sig av en mindre säkerhetsmarginal, som beskrivit i avsnitt 3.2.1, vilket bidrar till ökat trafikflöde.

6.5.3 Integrering av intern- och extern lokalisering

Även om filterbaserad sensorfusion erbjuder en mer robust lösning än enskilda sensorkällor, kvarstår vissa utmaningar i den nuvarande implementationen:

- Detektioner från flera kameror kan ge upphov till lokala konflikter i GulliView-data, vilka kan komma att störa EKF algoritmen.
- Eftersom GulliView inte tillhandahåller hastighetsdata, förlitar sig filterprediktionen helt på odometri mellan externa uppdateringar.
- Kovariansmatriserna är statiska, vilket begränsar systemets förmåga att anpassa sig till varierande förhållanden. En dynamisk viktning baserad på tillit, antal synliga kameror eller deras vinkel mot roboten skulle kunna förbättra noggrannheten.

Dessa begränsningar utgör värdefulla utgångspunkter för framtida utveckling. Exempelvis kan en viktmodul införas som analyserar mätvariationer och justerar GulliView:s kovarianser i realtid.

Vidare kan ett mer avancerat förbehandlingssteg bidra till att minska brus och systematiska fel i externa mätningar innan de matas in i filteralgoritmen. Detta är särskilt viktigt vid användning av EKF, som antar att brus är normalfördelat—en förutsättning som ofta inte är helt uppfylld i praktiken.

Det fortsatta arbetet med integrering av extern lokalisering bör även inkludera en jämförande utvärdering av olika filteralgoritmer, såsom EKF och Particle Filter. Det är också relevant att undersöka mer integrerade SLAM-lösningar, som exempelvis Cartographer, vilket erbjuder sammansatt optimering av odometri, LiDAR, punktmoln och externa fixpunkter inom ett och samma ramverk. Sådana system innehåller ofta inbyggda loop closure-mekanismer som ytterligare kan minska ackumulerad drift över tid.

Slutligen är noggrann kalibrering mellan det externa systemets koordinatsystem och robotens interna karta avgörande för att uppnå högkvalitativ sensorfusion. I kombination med korrekt tidsstämpling och latenskompensation, kan detta lägga grunden för ett mer tillförlitligt och responsivt lokaliseringssystem.

Användandet av AI

AI verktyg såsom ChatGPT och andra modeller har använts under arbetets gång som ett verktyg för snabb återkoppling, hjälp med mindre kodstycken och för stödhjälp i skrivandet av denna rapport. Medvetenhet om missvisande och inkorrekt information har iakttagits under projektets utförande och aldrig har dessa AI verktygs svar använts utan en kritisk analys.

Litteraturförteckning

[1] Othman, "Exploring the implications of autonomous vehicles: a comprehensive review." 2022. [Online]. Tillgänglig: https://doi.org/10.1007/s41062-022-00763-6

[2] C. T. högskola, "Föreskrift för genomförande och examination av kandidatarbete på civilingenjörs- och arkitektprogram," 2024, hämtad 2-Maj-2025. [Online]. Tillgänglig: https://webbpublicering360.portal.chalmers.se/Intern/Home/Download?recordnor=998033%262024_09%261238940_1_1.PDF%26pl

[3] J. P. Vilaseca, R. Li, B. Xie, Y. Dong, och H. Tan, "Ml and cv for precision indoor localization of scaled vehicular systems," 2024, rapport kurs DAT295/DIT669 Autonoma and samverkande fordonssystem.

[4] OpenCV, "About," hämtad 9-Apr-2025. [Online]. Tillgänglig: https://opencv.org/about/

[5] V. Savic, E. M. Schiller, och M. Papatriantafilou, "Distributed algorithm for collision avoidance at road intersections in the presence of communication failures," i *Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, s. 1005–1012. [Online]. Tillgänglig: https://ieeexplore.ieee.org/document/7995846

[6] K. Habibi, J. Johansson, K. Rydén, A. Shirzad, J. Sjöberg, och A. Soltani, "Kooperativa manövrar med nedskalade självkörande fordon," 2022, kandidatarbete vid Institutionen för data och informationsteknik.

[7] A. A. et. al, "Styrning av nedskalade fordon," 2020, kandidatarbete vid Institutionen för data och informationsteknik.

[8] M. Aramrattana, T. Larsson, C. Englund, J. Jansson, och A. Nåbo, "A novel risk indicator for cut-in situations," i *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, s. 1–6. [Online]. Tillgänglig: https://doi.org/10.1109/ITSC45102.2020.9294315

[9] E. Andreotti, Selpi, och M. Aramrattana, "Cooperative merging strategy between connected autonomous vehicles in mixed traffic," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, s. 825–837, 2022. [Online]. Tillgänglig: https://doi.org/10.1109/OJITS.2022.3179125

[10] E. Blohm, E. Diatchkov, E. Stenmark Tullberg, K. Nguyen, O. Wir, och M. Ygdell, "Globala koordinatsystem och autonom navigering med slam och turtlebot," Institutionen för Data- och Informationsteknik, Chalmers Tekniska Högskola, Göteborgs

Universitet, Kandidatarbete, 2025, handledare: Elad Michael Schiller; Examinatorer: Arne Linde, Patrik Jansson.

[11] H. D.-W. och T. Bailey. (2006) Simultaneous localization and mapping: Part i," ieee robotics & automation magazine. [Online]. Tillgänglig: https://www.wevolver.com/article/sensor-fusion

[12] ROS.org, "Ros: Robot operating system," hämtad 8-Apr-2025. [Online]. Tillgänglig: https://www.ros.org/

[13] C. Robotics, "Turtlebot 4," hämtad 8-Apr-2025. [Online]. Tillgänglig: https://clearpathrobotics.com/turtlebot-4/

[14] O. Morales-Ponce, E. M. Schiller, och P. Falcone, "Cooperation with disagreement correction in the presence of communication failures," i *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2014, s. 1105–1110. [Online]. Tillgänglig: https://doi.org/10.1109/ITSC.2014.6957835

[15] ——, "How to stop disagreeing and start cooperating in the presence of asymmetric packet loss," *Sensors*, vol. 18, nr 4, s. 1287, 2018. [Online]. Tillgänglig: https://doi.org/10.3390/s18041287

[16] T. Petig, E. M. Schiller, och J. Suomela, "Changing lanes on a highway," i *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, ser. OASIcs, vol. 65, 2018, s. 9:1–9:15. [Online]. Tillgänglig: https://doi.org/10.4230/OASIcs.ATMOS.2018.9

[17] C. Berger, O. Morales-Ponce, T. Petig, och E. M. Schiller, "Driving with confidence: Local dynamic maps that provide los for the gulliver test-bed," i *Computer Safety, Reliability, and Security - SAFECOMP 2014 Workshops*, ser. LNCS, vol. 8696. Springer, 2014, s. 36–45.

[18] A. Casimiro, O. Morales-Ponce, T. Petig, och E. M. Schiller, "Vehicular coordination via a safety kernel in the gulliver test-bed," i *34th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2014, s. 167–176. [Online]. Tillgänglig: https://doi.org/10.1109/ICDCSW.2014.25

[19] A. Casimiro, J. Kaiser, E. Schiller, P. Costa, J. Parizi, R. Johansson, och R. Librino, "The karyon project: Predictable and safe coordination in cooperative vehicular systems," i *IEEE/IFIP DSN Workshops*, 2013, s. 1–12. [Online]. Tillgänglig: https://doi.org/10.1109/DSNW.2013.6615530

[20] SAE international, "SURFACE VEHICLE RECOMMENDED PRACTICE," 2021, hämtad: 08-Apr-2025. [Online]. Tillgänglig: https://wiki.unece.org/download/attachments/128418539/SAE%20J3016_202104.pdf?api=v2

[21] Z. Fu, A. Axelzon, J. Zhang, och Y. Huang, "Ems2: Image processing in vehicular systems," rapport i kursen DAT295/DIT669 Autonoma och samverkande fordonssystem.

[22] E. Ahlberg, J. Danielsson, M. Isaksson, S. Ivarsson, och A. Johnsson, "Gulliver - vidareutveckling av ett system för testning av autonoma bilar,"

Chalmers University of Technology, Tekn. rapp., 2014. [Online]. Tillgänglig: https://hdl.handle.net/20.500.12380/203218

[23] Elgato, "Facecam pro," hämtad 7-Feb-2025. [Online]. Tillgänglig: https://www.elgato.com/se/sv/p/facecam-pro

[24] A. R. Laboratory, "Apriltags visual fiducial system," hämtad 19-Maj-2025. [Online]. Tillgänglig: https://april.eecs.umich.edu/software/apriltag

[25] ROS.org, "Understanding ros nodes," hämtad 8-Apr-2025. [Online]. Tillgänglig: https://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

[26] WiFiBot, "Wifibot lab: An open platform for r&d and education," hämtad 4-Feb-2025. [Online]. Tillgänglig: https://www.wifibot.com/

[27] iRobot Education, "Create 3 hardware overview," hämtad 8-Apr-2025. [Online]. Tillgänglig: https://iroboteducation.github.io/create3_docs/hw/overview/

[28] B. Parida. (2023) Sensor fusion: The ultimate guide to combining data for enhanced perception and decision-making. [Online]. Tillgänglig: https://www.wevolver.com/article/sensor-fusion

[29] Aptiv. (2020) What is sensor fusion? Hämtad: 10-Apr-2025. [Online]. Tillgänglig: https://www.aptiv.com/en/insights/article/what-is-sensor-fusion

[30] OpenCV, "Opencv library modules," hämtad 19-Maj-2025. [Online]. Tillgänglig: https://docs.opencv.org/4.x/index.html

[31] ——, "Perspective-n-point (pnp) pose computation," hämtad 19-Maj-2025. [Online]. Tillgänglig: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

[32] T. Moore, "robot_localization: State estimation for ros robots," hämtad 8-Apr-2025. [Online]. Tillgänglig: https://docs.ros.org/en/rolling/p/robot_localization/

[33] S. Hangal, E. Nylander, E. Olson, T. Petig, A. Soderberg-Rivkin, E. Svensson, och M. Zucker, "Gulliview," 2025.

A

Mätdata från evaluering för intern och extern lokalisering

Värden som har tagits under tester av jämförelse mellan intern- och externlokalisering.

	Segment 1 (6 m)		Segment 2 (2.5 m)		Segment 3 (6 m)		Segment 4 (2.5 m)	
Test	GV (m)	SLAM (m)	GV (m)	SLAM (m)	GV (m)	SLAM (m)	GV (m)	SLAM (m)
Test 1	6.526	6.128	2.935	2.594	6.471	6.260	2.905	2.611
Test 2	6.520	6.284	2.937	2.581	6.476	6.444	2.896	2.626
Test 3	6.525	6.223	2.943	2.593	6.463	6.339	2.893	2.601
Medel	6.524	6.212	2.938	2.589	6.470	6.348	2.898	2.613
Test	GV Err (m)	SLAM Err (m)	GV Err (m)	SLAM Err (m)	GV Err (m)	SLAM Err (m)	GV Err (m)	SLAM Err (m)
Test 1	0.526	0.128	0.435	0.094	0.471	0.260	0.405	0.111
Test 2	0.520	0.284	0.437	0.081	0.476	0.444	0.396	0.126
Test 3	0.525	0.223	0.443	0.093	0.463	0.339	0.393	0.101
Medel	0.524	0.212	0.438	0.089	0.470	0.348	0.398	0.113

Tabell 1: Mätvärden (överst) och fel (underst) för gamla GulliView (GV) och SLAM per segment (med angiven distans) för tester 1–3 samt deras medelvärden.

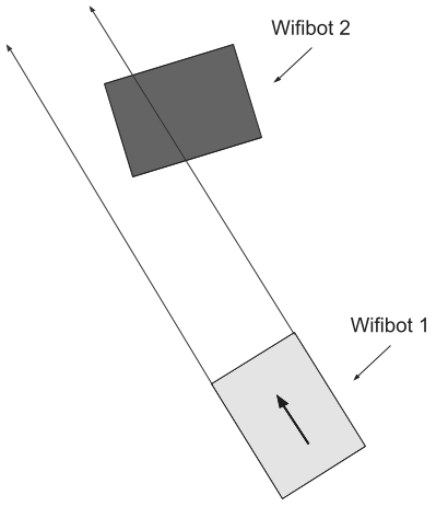
	Segment 1 (6 m)		Segment 2 (2.5 m)		Segment 3 (6 m)		Segment 4 (2.5 m)	
Test	Nya GV (m)	SLAM (m)	Nya GV (m)	SLAM (m)	Nya GV (m)	SLAM (m)	Nya GV (m)	SLAM (m)
Test 4	5.983	6.254	2.436	2.596	6.003	6.374	2.480	2.609
Test 5	5.982	6.247	2.436	2.595	6.007	6.438	2.481	2.629
Test 6	5.989	6.248	2.444	2.586	6.008	6.379	2.478	2.611
Medel	5.985	6.250	2.439	2.592	6.006	6.397	2.480	2.616
Test	Nya GV Err (m)	SLAM Err (m)	Nya GV Err (m)	SLAM Err (m)	Nya GV Err (m)	SLAM Err (m)	Nya GV Err (m)	SLAM Err (m)
Test 4	0.017	0.254	0.064	0.096	0.003	0.374	0.020	0.109
Test 5	0.018	0.247	0.064	0.095	0.007	0.438	0.019	0.129
Test 6	0.011	0.248	0.056	0.086	0.008	0.379	0.022	0.111
Medel	0.015	0.250	0.061	0.092	0.006	0.397	0.020	0.116

Tabell 2: Mätvärden (överst) och fel (underst) för nya GulliView (GV) och SLAM per segment (med angiven distans) för tester 4–6 samt deras medelvärden.

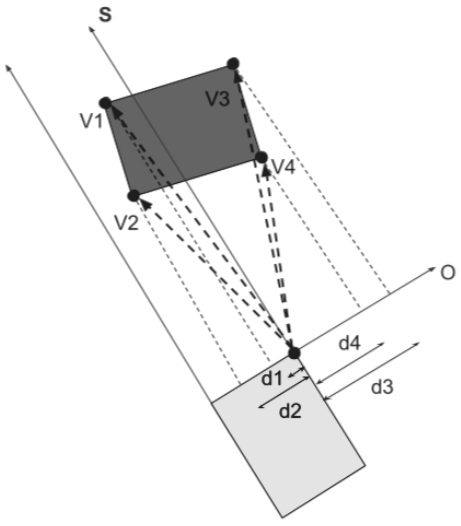
B

Kollisionsförebyggande hastighetsreglering

I projektets utvecklades även en farthållare som beräknar avståndet till framförvarande fordon och justerar hastighetsbegränsningen för roboten så att ett visst avstånd åtminstone hålls till fordonet. Avståndsberäkningen baseras på positionsdata från GulliView som robotarna ständigt samlar in. I programmet representeras en robot med en position (dess centerpunkt), en hastighetsvektor, samt fyra hörnpunkter. För att beräkna avståndet till fordon som är i vägen för en robot, behöver programmet hitta vilka fordon som är i vägen för roboten. För att göra detta definieras två sidovektorer, i samma riktning som robotens hastighetsvektor, utifrån de yttersta punkterna av roboten i sidled, se figur B.1. För att avgöra om WifiBot 2 i figur B.1 är i vägen för WifiBot 1 i samma figur, beräknas på vilken sida av ettans sidovektorerna som tvåans hörnpunkter hamnar. Detta görs med vektor projektion för en ortogonal vektor till sidovektorerna och en vektor mellan sidovektorernas startpunkt och hörnpunkten på tvåan som ska undersökas, se figur B.2.



Figur B.1: Representation av WiFiBotarna med vektorer.

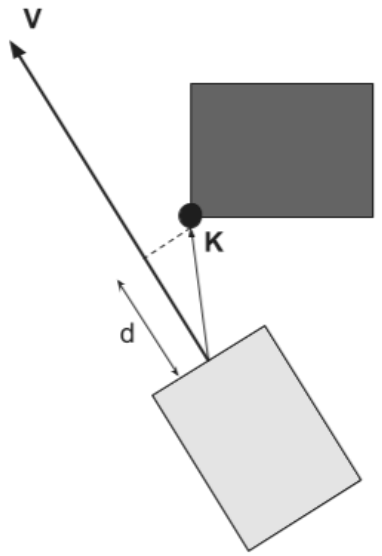


Figur B.2: Representation av WiFiBotarna med vektorer för kollisions beräkning.

Genom att projektera vektorerna V_1, V_2, V_3 och V_4 på vektorn O, fås en vektor parallell med O vars riktning kan avgöra vilken sida av sidovektorerna punkten befinner sig, se ekvation B.1. I denna ekvation är $\vec{e_O}$ en enhetsvektor för O och d_1, d_2, d_3 samt d_4 är reella koefficienter som beskriver projektionsvektorns längd och riktning. Då längdkoefficienten är negativ är punkten på vänster sida om sidovektorn, om den är positiv är den på höger sida. Om ett fordon har någon av sina hörnpunkter på olika sidor om dessa två sidovektorer är den i vägen för roboten. Om fordonets hörnpunkter är på samma sida om båda sidovektorerna är den inte i vägen. Skulle ett fordon vara i vägen kan avståndet till det beräknas genom en vektorprojektion av en vektor mellan roboten och den kolliderande punkten hos det andra fordonet, samt robotens hastighetsvektor, se figur B.3 och ekvation B.2.

$$proj_{V_i}(O) = d_i \cdot \vec{e_O} = \frac{V_i \cdot O}{||O||^2} O \quad i \in \{1, 2, 3, 4\} \tag{B.1}$$

$$proj_K(V) = d \cdot \vec{e_V} = \frac{K \cdot V}{||V||^2} V \Rightarrow d = \frac{K \cdot V}{||V||} \tag{B.2}$$



Figur B.3: Representation av WiFiBotarna med vektorer för avstånds beräkning.

I denna ekvation (B.2) beskriver d avståndet till kollision med fordonet, K är en vektor som pekar från roboten till kollisionspunkten, V är fordonets hastighetsvektor och $\vec{e_V}$ är enhetsvektorn av V. Med ett avstånd d till framförvarande fordon beräknas en hastighetsbegränsning för fordonet som hindrar den bakre roboten från att komma närmre de främre än en viss distans d_{ref}. Detta görs genom att kontinuerligt uppdatera fordonets hastighetsbegränsning genom att addera termen $(d - d_{ref}) \cdot 0.01$. Hastighetsbegränsningen är i sig även begränsad till att vara större än noll och mindre än ett maximivärde. Detta värde hindrar roboten från att åta en allt för hög hastighet som exempelvis överstiger begränsningen på vägen den kör på. Med en hastighetsbegränsning kontrolleras den efterfrågade hastigheten från kontrollsystemet av de exklusiva zonerna. Är hastigheten inom

gränserna skickas den till go_to_goal noden, som utför förändringen. Är värdet utanför gränserna skickas värdet för gränsen som den efterfrågade hastigheten överträdde.

C

Källkod - Github - Integrering av intern- och externa lokaliserings system - TurtleBot4

Denna Github rymmer både källkod och dokumentation för integrering av intern- och externa lokaliserings system, projekt(TurtleBot4), baserat på för årets projekt[10].

Länk till Github: turtlebot4-scaled-vehicles-2025

D

Källkod - Github - Kooperativa manövrar - WiFiBot

Detta repo innehåller de olika python filer vi utvecklat samt ytterligare filer från tidigare arbeten som har sett små förändringar för att fungera för oss.

Länk till Github: Cooperative-driving-2025

Övriga filer som systemet är beroende av finns i laboratoriets privata bitbucket.

E

Källkod - Bitbucket - Laboratoriets samtliga projekt

I laboratoriets privata bitbucket ryms tidigare utförda projekt samt vad detta projekt har utvecklat.

Länk till bitbucket: automationArticulatedVehicles

F

GulliViews nya kodstruktur