



# Kooperativa manövrar med nedskalade självkörande fordon

Kandidatarbete vid institutionen för Data- och informationsteknik

Keivan Habibi  
Johannes Johansson  
Kristofer Rydén  
Ali Shirzad  
Jakob Sjöberg  
Ali Soltani



## KANDIDATARBETE 2024

# Kooperativa manövrar med nedskalade självkörande fordon

Keivan Habibi

Johannes Johansson

Kristofer Rydén

Ali Shirzad

Jakob Sjöberg

Ali Soltani



UNIVERSITY OF  
GOTHENBURG



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och informationsteknik  
CHALMERS TEKNISKA HÖGSKOLA  
GÖTEBORGS UNIVERSITET  
Göteborg, Sverige 2024

Kooperativa manövrar med nedskalade självkörande fordon  
Keivan Habibi Johannes Johansson Kristofer Rydén Ali Shirzad Jakob Sjöberg  
Ali Soltani

© Keivan Habibi, Johannes Johansson, Kristofer Rydén, Ali Shirzad, Jakob Sjöberg,  
Ali Soltani, 2024.

Handledare: Elad Michael Shiller, Institutionen för Data- och informationsteknik  
Examinatorer: Arne Linde och Patrik Jansson, Institutionen för Data- och informationsteknik

Rättande lärare: Magnus Almgren, Institutionen för Data- och informationsteknik

Kandidatarbete 2024  
Institutionen för Data- och informationsteknik  
Chalmers tekniska högskola och Göteborgs universitet  
SE-412 96 Göteborg  
Telefon +46 31 772 1000

Typsatt i L<sup>A</sup>T<sub>E</sub>X  
Göteborg, Sverige 2024

## **Abstract Jakob Kristofer**

The transportation industry is undergoing a significant transformation as autonomous driving systems become more prevalent. Safety and reliability are two great concerns with this new technology. To ensure that these concerns are properly handled a safe test environment is a necessity. This project extends the development and utilization of a laboratory environment for scaled autonomous vehicles at Chalmers University of Technology, building upon previous work done by multiple groups of bachelor students. The objectives of this project were to enhance software for vehicle detection and develop algorithms for managing critical situations, such as intersections and merges, where multiple vehicles require coordination.

The software improves the handling of such situations by enabling faster vehicle detection through image processing and enhanced communication via the indoor positioning system. To achieve this, a multi-threading approach was employed to increase the processing speed of captured images, resulting in a notable performance enhancement of 267% and 315% for Full HD and 4K resolutions, respectively.

Communication capabilities of the system were enhanced through the implementation of a new architecture for the membership service, utilizing Inter-Process Communication (IPC) methodology and transitioning from a decentralized to a centralized sender model. This transition led to a significant improvement, achieving nearly a 500% increase in communication speed.

Algorithms for different driving situations were implemented using a server that communicates with vehicles, instructing them to adjust speeds based on the positions and velocities of other vehicles. Remaining challenges with the lab environment are to decrease detection time by only searching certain parts of images, also to integrate the membership service with the maneuvers to make better decisions and handle more complex situations.

This work contributes new technology to the lab that improves the learning experience for future students at Chalmers University of Technology making it possible for more advanced projects. This in turn furthers the development of a safe environment for testing autonomous driving systems.

Keywords: Wifibot, GulliView, autonomous vehicles, AprilTag, indoor localization, computer vision, IPC

## Sammandrag Jakob Kristofer

I takt med att autonoma körsystem blir allt vanligare genomgår transportsektorn en betydande omvandling. Säkerheten och tillförlitligheten är två aspekter som har stor prioritet i arbetet kring denna nya teknik. För att försäkra sig om att dessa aspekter hanteras är en säker testmiljö essentiell. Detta projekt har fortsatt utvecklingen och användningen av en labbmiljö för nedskalade självkörande fordon på Chalmers tekniska högskola, som bygger vidare på tidigare kandidatarbeten utförda av studenter där. Projektets mål var att förbättra mjukvaran för fordonsdetektering och att utveckla algoritmer för att hantera kritiska situationer såsom korsningar och sammanvävningar där koordination mellan flera fordon krävs.

För att förbättra hanteringen av sådana situationer möjliggjorde den uppdaterade mjukvaran snabbare detektering av fordon genom bildbehandling och förbättrad kommunikation via inomhuspositioneringssystemet. Multitrådnings användes för att öka bearbetningshastigheten för tagna bilder, vilket resulterade i en noterbar prestandaförbättring på 267% och 315% för Full HD- respektive 4K-upplösningar.

Systemets kommunikationskapacitet förbättrades genom implementeringen av en ny arkitektur för medlemstjänsten, med användning av metoder inom Inter-Process Communication (IPC) och övergång från en decentraliserad till en centraliserad avsändarmodell. Denna övergång ledde till en betydande förbättring och uppnådde nästan 500% ökning av kommunikationshastigheten.

Algoritmer för olika trafiksituitioner implementerades med hjälp av en server som kommunicerar med fordon och instruerar dem att justera hastigheter baserat på positioner och hastigheter hos andra fordon. Kvarstående utmaningar i projektet är bland annat att minska detektionstiden för GulliView genom att bara söka i vissa delar av bilden. Att integrera medlemstjänsten tillsammans med manöveralgoritmerna skulle leda till bättre beslutsfattande och därmed kunna hantera mer komplexa situationer.

Detta arbete bidrar med ny teknik till labbet som förbättrar lärandeupplevelsen för framtida studenter vid Chalmers tekniska högskola och gör det möjligt för mer avancerade projekt. Detta i sin tur främjar utvecklingen av en säker miljö för testning av autonoma körsystem.

Nyckelord: Wifibot, GulliView, självkörande fordon, AprilTag, inomhuslokalisering, datorseende, IPC



## Förord

Vi vill uttrycka vår djupa tacksamhet gentemot vår handledare Elad Michael Schiller för hans ovärderliga expertis inom ämnet, vilket har varit avgörande under alla delar av projektet. Vi vill även rikta ett varmt tack till grupp 16 för deras exemplariska samarbete och delande av labbmiljön under hela projektets gång. Slutligen vill vi tacka Matei Schiopu för hans tidigare erfarenhet av labbmiljön, vilket har varit till stor hjälp med vägledning och stöd.

Keivan Habibi, Johannes Johansson, Kristofer Rydén, Ali Shirzad, Jakob Sjöberg,  
Ali Soltani  
Göteborg, september 2024



# Innehåll

<b>Figurer</b>	<b>xi</b>
<b>Tabeller</b>	<b>xiii</b>
<b>List of Algorithms</b>	<b>xiv</b>
<b>1 Introduktion Kristofer Jakob</b>	<b>1</b>
1.1 Syfte . . . . .	1
1.2 Relaterade arbeten . . . . .	1
1.3 Vårt bidrag Jakob Kristofer . . . . .	2
1.4 Samhälleliga och etiska aspekter Soltani Jakob . . . . .	3
<b>2 Bakgrund</b>	<b>4</b>
2.1 GulliView Keivan Shirzad . . . . .	4
2.1.1 Versioner . . . . .	4
2.1.2 Mjukvaran . . . . .	5
2.2 Wifibot Jakob Johannes . . . . .	5
2.2.1 Robot Operating System (ROS) . . . . .	6
2.2.2 Navigering . . . . .	6
2.3 Medlemstjänst Soltani Jakob . . . . .	7
2.3.1 Tidigare arkitektur . . . . .	7
2.3.2 Grupp Tur och Returtiden (GTR) . . . . .	9
2.3.3 Interprocesskommunikation . . . . .	9
<b>3 Metod</b>	<b>10</b>
3.1 GulliView Johannes Kristofer . . . . .	10
3.1.1 Multitrådning . . . . .	10
3.1.2 Bildupplösning och bildfrekvens . . . . .	12
3.1.3 AprilTags . . . . .	12
3.1.4 Bildbehandling . . . . .	13
3.1.5 Kamerasammansättning . . . . .	13
3.2 Manövrar Kristofer Soltani . . . . .	13
3.2.1 Sammanvävningsalgoritmen . . . . .	13
3.2.2 Korsningar . . . . .	15
3.3 Medlemstjänsten Shirzad Keivan . . . . .	16
3.3.1 Uppdaterad arkitektur Shirzad Keivan . . . . .	16

3.3.2	Sammanhangande meddelande via en centralisera <sup>n</sup> d avsändare <i>Shirzad Keivan</i>	17
3.3.3	Övergång från nätverkssändning till delat minne <i>Soltani Keivan</i>	17
<b>4</b>	<b>Utvärdering</b>	<b>19</b>
4.1	GulliView <i>Johannes Soltani</i>	19
4.1.1	Tidsprofilering	19
4.1.2	Prestanda	19
4.1.3	Maximal hastighet	20
4.2	Manövrar <i>Jakob Shirzad</i>	21
4.3	Medlemstjänst <i>Shirzad Jakob</i>	22
4.3.1	K-värdeseffekt på GTR <i>Soltani Jakob</i>	22
4.3.2	Hastighetens påverkan på GTR <i>Shirzad Jakob</i>	22
<b>5</b>	<b>Resultat</b>	<b>23</b>
5.1	GulliView <i>Keivan Jakob</i>	23
5.1.1	Bildtransformering	23
5.1.2	Kameraläsning	25
5.1.3	Taggsökning	26
5.1.4	Prestanda	28
5.2	Manövrar <i>Jakob Shirzad</i>	30
5.2.1	Sammanvävning	30
5.2.2	Korsning	31
5.3	Medlemstjänst <i>Shirzad Jakob</i>	32
5.3.1	K-värdeseffekt på GTR <i>Soltani Jakob</i>	32
5.3.2	Hastighetens påverkan på GTR <i>Shirzad Jakob</i>	33
<b>6</b>	<b>Diskussion</b>	<b>35</b>
6.1	GulliView <i>Keivan Soltani</i>	35
6.1.1	Bildtransformation	35
6.1.2	Kameraläsning	35
6.1.3	Taggsökning	36
6.1.4	Prestanda och Upplösning	37
6.2	Manövrar <i>Kristofer Johannes</i>	37
6.2.1	CRI	37
6.2.2	Tidsskillnad mellan robotarna	38
6.2.3	Korsningar	39
6.3	Medlemstjänst <i>Soltani Keivan</i>	39
<b>7</b>	<b>Slutsats <i>Keivan Johannes</i></b>	<b>41</b>
7.1	Framtida arbeten <i>Keivan Kristofer</i>	41
7.1.1	GulliView <i>Keivan Kristofer</i>	41
7.1.2	Manövrar <i>Jakob Kristofer</i>	41
7.1.3	Medlemstjänst <i>Shirzad Kristofer</i>	42
7.2	Sammanfattning	42

# Figurer

2.1	Wifibot med AprilTag . . . . .	5
2.2	Sammanvävningens utseende i labbmiljön. Roboten till vänster får designationen main, medan den till höger får ramp. . . . .	7
2.3	Kommunikationsarkitekturen för den tidigare medlemstjänsten [1]. Återgiven med tillstånd. . . . .	8
2.4	Flödesdiagrammet illustrerar processen för att beräkna (GTR) . . . . .	9
3.1	Bild över korsningen. När en robot upptäcks inom den gula cirkeln läggs de till i en kö. Först i kön får prioritet och kör vidare, senare ankomster stannar. . . . .	15
3.2	Den reviderade arkitekturen för medlemstjänsten utnyttjar (IPC) till-sammans med en universell sändare. . . . .	16
4.1	Oändlighetens symbol som förtydligar en omfattande testkörning. . . . .	22
5.1	Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer. . . . .	23
5.2	Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer. . . . .	24
5.3	Grupperade stapeldiagram som jämför den genomsnittliga exekveringstiden mellan den nya och gamla transformationsalgoritmen på 4k och Full HD upplösning. . . . .	24
5.4	Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer. . . . .	25
5.5	Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på 4k upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer. . . . .	25
5.6	Grupperade stapeldiagram som jämför den genomsnittliga exekveringstiden mellan den nya och gamla kameraläsningsalgoritmen på 4k och Full HD upplösning. . . . .	26
5.7	Normaliserade histogram för tagg sökningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer. . . . .	26

5.8	Normaliserade histogram för tagg sökningstider från 10 000 iterationer från respektive kamera på 4k upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer. . . . .	27
5.9	Grupperade stapeldiagram som jämför den genomsnittliga exekveringstiden mellan den nya och gamla taggsökningsalgoritmen på 4k och Full HD upplösning. . . . .	27
5.10	Normaliserade histogram för frekvensmätning från 166 datapunkter uttaget varje 60:e datapunkt från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer och 664 datapunkter. . . . .	28
5.11	Normaliserade histogram för frekvensmätning från 166 datapunkter uttaget varje 60:e datapunkt från 10 000 iterationer från respektive kamera på 4k upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer och 664 datapunkter. . . . .	28
5.12	Grupperade stapeldiagram som jämför den genomsnittliga frekvensen mellan den nya och gamla algoritmen på 4k och Full HD upplösning. . . . .	29
5.13	CRI för en sammanvävning i 0,75 m/s, punkten för sammanvävning nås av första roboten vid 5 s. . . . .	30
5.14	Tidsskillnaden för robotarna att nå punkten för sammanvävningen vid olika maxhastigheter. . . . .	31
5.15	Hastigheter för robotarna vid ett möte vid korsningen. Orange saktar ner för att försöka släppa förbi den andra utan att stanna helt men stannar då den når korsningen innan den andra lämnat korsningen. . . . .	31
5.16	GTR-värdefördelning och normalfördelning vid K och hastighet 0,5 . . . . .	32
5.17	GTR-värdefördelning vid en hastighet på 0,5 m/s visar att de flesta värdena ligger mellan 40 och 60 ms, med en lång svans som antyder att GTR gör några studsar. . . . .	33
5.18	GTR-värdefördelning vid en hastighet på 0,7 m/s visar att de flesta värdena ligger under 60 ms, med en lång svans som antyder att GTR gör några studsar. . . . .	34
5.19	GTR-värdefördelning vid en hastighet på 1,0 m/s visar att de flesta värdena ligger mellan 30 och 60 ms, med en lång svans som antyder att GTR gör några studsar. . . . .	34

# Tabeller

5.1	Maximala hastigheterna mätt i m/s som robotarna kan köra i Full HD där GulliView kan registrera dess AprilTags med K=16. Med 200 mm taggarna nåddes maxhastigheten på roboten utan förlust av detektion.	29
5.2	Maximala hastigheterna mätt i m/s som robotarna kan köra i 4k där GulliView kan registrera dess AprilTags med K=8. . . . .	29
5.3	K och medelvärde på GTR-värden . . . . .	33
5.4	Hastighet och medelvärde på GTR-värden . . . . .	34

# List of Algorithms

1	Skriv till buffer . . . . .	11
2	Läs från buffer . . . . .	11

# 1

## Introduktion **Kristofer Jakob**

Självkörande fordon implementeras mer och mer i trafiken och höga krav på säkerhet och effektivitet ställs. Vid riskfylda situationer som sammanvävningar, korsningar och körfältsbyte behöver flera fordon förhålla sig till varandra för att utföra manövrar säkert när kritiska beslut behöver tas [2]. Till detta finns det en hel del olika lösningar för att göra systemen bättre. En del fordon använder sig av kameror på fordonen för att kunna skapa sig en bild av omgivningen [3] medan andra använder sig utav kameror i taket i till exempel lagerbyggander [4]. De fordon som har kameror monterade på sig finns oftast på fordon som kör ute i trafiken, på grund av att andra fordon i trafiken oftast inte är autonoma. I lagermiljöer har man möjlighet att ha kameror som kan ge en global vy över området. En sådan lösning gör det möjligt för fordonen att koordinera sig med varandra innan de sett varandra vilket kan leda till bättre synkronisering. Genom att kunna implementera detta på trafik, vid utsatta områden som korsningar och sammanvävningar, skulle fordonen kunna kommunicera med varandra redan innan de nått fram till området.

Fokus i detta arbete är att uppgradera mjukvara för ett inomhuslokaliseringssystem, säkerställa att kommunikation med självkörande robotar upprätthålls, samt att utveckla algoritmer för att hantera dessa kritiska situationer.

### 1.1 Syfte

Syftet med detta arbete är att utveckla och använda en inomhusmiljö för nedskalade fordon för att förbättra nästkommande studenters förutsättningar och upplevelse. Del av detta är att utveckla säkra algoritmer för att undvika kollisioner vid korsningar och sammanvävningar, samt att förbättra kommunikationen och det lokaliseringssystem som finns.

### 1.2 Relaterade arbeten

Labbmiljön bygger på tidigare arbeten, framförallt testmiljön för Gulliver som utvecklades som ett alternativ till komplexa simuleringar för självkörande fordon [5]. Den nuvarande medlemstjänsten har sitt ursprung i tidigare forskning inom detta område. Mer information om dess skapande och bakgrund finns i referenserna [6–13]. Vårt arbete bygger vidare på denna grund och strävar efter att fortsätta att förbättra och utveckla medlemstjänsten.

## 1.3 Vårt bidrag **Jakob Kristofer**

Detta projekt fokuserar på att introducera ny hårdvara och förbättra mjukvaran för labbmiljöns kritiska system, inomhuslokaliseringssystemet GulliView. Genom mjukvaruutveckling, datorseende, multitrådade processer, synkroniseringsmetoder, och datorkommunikation ges nya möjligheter för framtida projekt och för de studenter som använder labbmiljön.

GulliView är ett lokaliseringssystem (beskrivs i 2.1) som gör det möjligt för att planera och kontrollera robotar i en inomhusmiljö. Datorseende används för att detektera AprilTags av storleken 200x200 mm, men GulliView har många brister på grund av föråldrad teknologi i bland annat de kameror som används som endast tillåter 30 bilder per sekund i 1080p upplösning.

Tack vare införandet av nya kameror och förbättrad mjukvara finns nu möjligheten att bättre använda mindre storlekar än 200 millimeter. En snabbare uppdateringsfrekvens uppnås även med ökningar på 315% och 267% för full HD och 4k. Denna prestandaförbättring är en direkt korrelation till den förbättrade transforméringsalgoritmen som tillsammans med multitrådlösningen tillåter robotarna att köra snabbare än tidigare. Den genomsnittliga tiden att transformera en bild på 4k blev 160% snabbare och 600% snabbare på full HD. Kamerans genomsnittliga lästid noterades också bli snabbare på 4k där en tidsförbättring på nästan 44% uppmättes. Slutligen åtgärdades även problem med en tidigare utvecklad sökalgoritm gjord av föregående kandidatarbete, där den nu är applicerbar på ett system med fler än en kamera.

Processen för medlemstjänsten har genomgått betydande förbättringar när det gäller övervakningen av robotarnas position, hastighet, tid och riktning och kommunikationstid. Tidigare var tjänsten endast kompatibel med den äldre versionen av GulliView och hade en genomsnittlig kommunikationstid på cirka 400 ms [1]. Nu kan den fungera med den senaste versionen av GulliView och hantera alla robotar i testmiljön med en genomsnittlig kommunikationstid på cirka 74 ms, vilket är nästan fem gånger snabbare. Denna prestandaförbättring har uppnåtts genom en serie avancerade förbättringar, inklusive användning av samtidig programmering och uppdatering av medlemstjänstens arkitektur med en enkel och stadig design. Denna nya design utnyttjar Interprocesskommunikation och en centralisering för att effektivt hantera kommunikationen mellan olika delar av systemet.

Med avseendet att verifiera uppdateringarna har en sammanvävningsmanöver förbättrats och en ny manöver för att hantera en korsning implementerats med två robotar av typen Wifibot. Den uppgraderade hårdvaran har gjort det möjligt att utföra sammanvävningsmanövern säkert i mycket högre hastighet, 0,75 m/s istället för 0,5 m/s.

Med de förbättringar som gjorts är inomhuslokaliseringssystemet mer robust och har en bättre precision som möjliggör implementation av mer stabila manövrar i

högre hastigheter. Tekniker som föreslås i denna rapport tror vi ytterligare kan förbättra Gulliviews precision och robusthet.

## 1.4 Samhälleliga och etiska aspekter [Soltani Jakob](#)

Arbetet utfördes i en kontrollerad miljö utan behov av etiska överväganden för metoden. Om resultatet tillämpas i verkliga situation behöver dock flera viktiga aspekter beaktas.

Ur forskningssynpunkt är målet att bidra till utvecklingen av självkörande fordon, vilket kan ha betydande positiva samhällseffekter. Sådana fordon kan öka tillgängligheten för icke-förare och frigöra tid för nuvarande förare. Ur miljösynpunkt är självkörande fordon teoretiskt mer energieffektiva, men det finns indikationer på att de faktiskt kan öka utsläppen på grund av ökad användning [14]. Självkörande fordon elimineras en stor del av trafikolyckorna som orsakas av mänskliga fel. Men det kan ställas etiska frågor om ansvar vid olyckor och prioritet för överlevnad [15]. Trots regler och fördelar visar studier att många fortfarande är osäkra på att lita på självkörande system, vilket kan påverka acceptansen och användningen av dem [16].

# 2

## Bakgrund

Labbmiljön som används är utvecklat av tidigare studenter på Chalmers. Det finns ett lokaliseringssystem för att meddela robotar sina positioner. Robotarna har grundfunktioner för att köra mellan olika positioner och ett protokoll finns för robotarnas kommunikation. Detta kapitel ger en djupare inblick i hur dessa system fungerar.

### 2.1 GulliView Keivan Shirzad

GulliView är ett inomhusbaserat positioneringssystem som består av fyra stycken kameror, som är uppskruvade i taket, i en kontrollerad labbmiljö. Tillsammans skapar dessa kameror ett koordinatsystem vars syfte är att identifiera robotarnas position och sedan meddela denna position till robotarna. Systemet drivs med hjälp av en NUC dator som i sin tur använder ett Linux baserad operativsystem tillsammans med programmeringsspråket C++ för logiken av mjukvaran. Kamerorna som används tillåter upp till 4k upplösning med 60 bilder per sekund.

#### 2.1.1 Versioner

Detta projekt är uppdelat i tre olika arbetsområden där två olika versioner av GulliViews mjukvara används. GulliView och Medlemstjänst 2.3 utnyttjar den senare versionen som är betydligt snabbare medan arbetet med manövreringen förhåller sig till den äldre och längsammare versionen. Anledningen till det här är på grund av att koordinatsystemet i den senare versionen inte stämmer mellan övergången av kamerorna, det vill säga den är inte anpassad till att fungera på ett system med fyra kameror.

Den versionen GulliView och Medlemstjänst arbete kommer baseras på använder sig av en algoritm som kallas för “Fast-search” som ett tidigare arbete utvecklat [1]. Den baseras på en algoritm som utför beräkningar på de robotar som befinner sig i bilden för att skapa en matematisk gissning om var de kommer att finna sig i nästa bild. Denna uppfattning används sedan för att avgränsa sökningen till ett bestämt område istället för att söka hela bilden och på så sätt spara tid. Den äldre versionen som manövrarna använder sig av har inte den algoritmen, utan använder sig av tidigare implementationer som grundar sig på “Exhaustive-search”. Med Exhaustive-search innebär det att hela bilden genomsöks för identifieringen av eventuella AprilTags till skillnad från ett specifikt område som Fast-search gör.

### 2.1.2 Mjukvaran

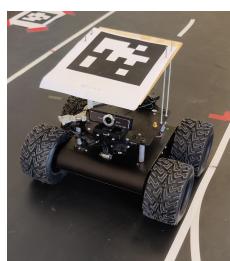
Kamerorna kör sin mjukvara oberoende av varandra, vilket innebär att det skapas fyra instanser av GulliViews mjukvara för fyra kameror. Det innebär också att fyra program behöver exekveras i kontrast till ett enskilt program vilket medför fördelar som exempelvis att ifall en kamera kraschar påverkar det inte övriga kameror. Nackdelar som introduceras med denna metod är bland annat komplexiteten gällande samspelet mellan kamerorna.

GulliViews mjukvara baseras på flertal komplexa beräkningar som utnyttjar både egenimplementerade algoritmer men även utomstående algoritmer. Mjukvaran är tungt beroende av Computer Vision biblioteket "OpenCV" men även av "AprilTag" biblioteket för identifieringen av robotarna och kalibreringen av koordinatsystemet. Mjukvaruprocessen kan delas upp i fem steg:

1. **Kalibrering:** Första steget i mjukvaran är all kalibrering som behövs utföras vid start av systemet. Det innehållar huvudsakligen kalibreringen av kamerorna för bildtransformeringarna, och koordinatsystemet med hjälp av utsatta AprilTags där programmet inte körs förrän fyra stycken blivit registrerade. Slutligen skapas bland annat kommunikationsprotokollen.
2. **Bildhämtning:** Andra steget handlar om att hämta bilderna från kameran och är första steget i huvudloopen.
3. **Bildhantering:** I det tredje steget hanteras bilden som hämtats från kameran där bilden i fråga transformeras för att underlätta identifieringen av robotarna för sökalgoritmen.
4. **Sökalgoritm:** Fjärde steget tar den transformerade bilden och applicerar sökalgoritmen där eventuella identifieringar extraheras.
5. **Kommunikation:** Det femte och sista steget kommunicerar de identifierade positionerna i koordinatsystemet i realtid till robotarna.

## 2.2 Wifibot Jakob Johannes

Robotarna som används i projektet är av typen Wifibot tillverkade av NexterRobotics som kan ses i figur 2.1. De är utrustade med IR-sensorer som mäter avstånd mellan 20–150 cm, och en dator med linux som operativsystem som ansluts över wifi med SSH. Utöver de egna sensorerna är en AprilTag fäst på ovansidan av roboten för att möjliggöra detektering med GulliView.



**Figur 2.1:** Wifibot med AprilTag

### 2.2.1 Robot Operating System (ROS)

Robot Operating System (ROS) är ett open-source ramverk för att kontrollera robotar och integreras i koden för robotarna med Python paketet ROSPY. ROS använder sig av noder som kör olika processer och som kommunicerar genom att publicera eller prenumerera på meddelanden på gemensamma ämnen (jfr en. topics). Ämnen har unika namn som används för att identifiera innehållet i ett meddelande. En av noderna som körs kontrollerar robotens motorer genom att publicera en specifik datastruktur med vektorer för hastighet och vinkel, i ROS heter den `twist_struct`, till ämnet `cmd_vel`. `Cmd_vel` avläses sedan av hjulens motorer som ändrar sin effekt och därmed hastigheten och riktningen som roboten kör i. För avläsning av IR-sensorerna används en prenumeration till ämnet `IR` som returnerar avstånden som sensorerna uppmätt via sin nod.

Kommunikation med GulliView sker genom en server som lyssnar efter UDP meddelanden som skickas på nätverket. När ett meddelande blir registrerat publiceras det till ett ämne, `gv_positions`, som en annan nod läser av och tar ut relevant information för sitt eget id som position och vinkel i rummet.

På samma sätt skickas meddelanden för att ändra robotarnas hastighet. Meddelanden broadcastas över UDP med robotarnas id och vilken hastighet de ska hålla. En server på roboten tar emot meddelandet och publicerar till ämnet `gv_laptop` som i sin tur publicerar vidare till `cmd_vel`.

### 2.2.2 Navigering

En process på robotarna använder sig av de ämnen som nämnts i 2.2.1 för att sammanställa all information och navigera runt i rummet. En lista över punkter används för att skapa en rutt som roboten följer. Genom att avläsa position och vinkel som roboten är riktad kan den köra mot nästa punkt. Avståndet mellan två punkter i koordinatsystemet är små och meddelanden från GulliView skickas ut med en fördröjning. En felmarginal på 200 punkter i en cirkel kring punkterna används för att roboten ska kunna träffa dem och gå vidare mot nästa. Värdena som IR-sensorerna avger används för att stoppa roboten om den kör för nära något, eller om något dyker upp framför den.

En algoritm för sammanvävning finns implementerad baserat på ett arbete av Selpi et al. [17]. Implementeringen körs från en dator som lyssnar på meddelande från GulliView och skickar ut meddelanden till robotarna som lyssnar på ämnet `gv_laptop`. När robotarna är nära en angiven startpunkt i rummet detekteras de av algoritmen och de blir designade som ramp- eller main-robot beroende på var de detekteras. Figur 2.2 visar situationen där robotarnas sammanvävning påbörjas.

Det skickas kontinuerligt meddelanden som ska korrigera robotarnas hastigheter när de kör mot en gemensam sammanvävningspunkt. Meddelandestrukturen för detta meddelande är:

{bot1\_id, bot1\_speed, bot1\_restart, bot2\_id, bot2\_speed, bot2\_restart}



**Figur 2.2:** Sammanvävningens utseende i labbmiljön. Roboten till vänster får designationen main, medan den till höger får ramp.

där `bot_id` är ett heltal med vilken robot som ska lyssna på följande två fält, `speed` och `restart`, där `speed` är ett flyttal med den nya hastigheten roboten ska köra och `restart` är en bool som kan användas för att robotarna ska köra flera varv i rummet och vänta in varandra vid testning. Algoritmen både ökar hastigheten på en av robotarna och sänker hastigheten på den andra för att öka avståndet mellan robotarna till ett angivet målvärde. Denna implementering har problemet att robotarnas hastigheter inte korrigeras tillräckligt snabbt vilket leder till kollisioner.

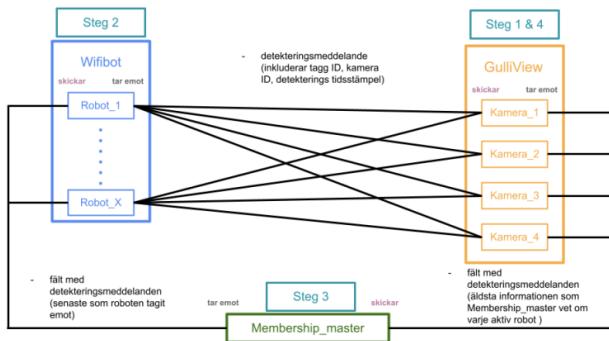
## 2.3 Medlemstjänst **Soltani Jakob**

Kärnuppgiften för medlemstjänsten är att underlätta kommunikationen mellan GulliView och robotarna. Denna kommunikation innefattar övervakning av robotarnas närvaro samt att säkerställa att de är aktiva och anslutna till nätverket. Dessutom ansvarar tjänsten för att samla in och tillhandahålla den senaste informationen, såsom robotarnas position, hastighet och det senaste tidpunkten då GulliView upptäckte robotarna. Eftersom robotarna inte kommunicerar direkt med varandra möjliggör medlemstjänsten att robotarna har tillgång till den senaste informationen om andra robotar som är anslutna till medlemstjänsten.

### 2.3.1 Tidigare arkitektur

Den tidigare implementationen av medlemstjänsten består av tre olika noder: Gulliview-noden, Wifibot-noden och Membership\_master. GulliView är skriven i C++, medan Wifibot och Membership\_master är skrivna i Python. Det är värt att notera att GulliView och Membership\_master körs på samma dator, medan Wifibot körs på varje robot separat.

Medlemstjänstens process utförs i fyra steg som visas i figur 2.3.



**Figur 2.3:** Kommunikationsarkitekturen för den tidigare medlemstjänsten [1]. Återgiven med tillstånd.

Första steget utförs i GulliView-noden. När GulliView detekterar en robot skickar den ett detektionsmeddelande till alla robotar via UDP genom så kallad bredsändning (jfr en: broadcasting). Detektionsmeddelandet inkluderar robotarnas tag\_id, position, hastighet, vinkel och detektionstid.

Andra steget utförs i Wifibot-noden. Här tar robotarna emot data från GulliView. Varje robot håller två fördefinierade fält, en lista med namnet latest\_detections och en matris DetectionMatrix[CameraId][RobotId] som innehåller information om alla robotar samt deras position och senaste detektionstid. När robotarna får ny data från GulliView jämför det nya meddelandet med det befintliga i "latest\_detection" listan och uppdaterar sedan matrisen med den nya informationen. Därefter paketerar varje robot sin tag\_ID och hjärtslagstidsstämpel (tiden när roboten skickar meddelandet till Membershipmaster) med det senaste meddelandet och skickar sedan det vidare till nästa nod Membership\_master, enligt en förbestämd CPU-cykel.

Det tredje steget inleds när Membership\_master mottar de senaste detektionerna från robotarna. De senaste detektionerna kan inkludera positionen för vilken robot som helst. När Membership\_master tar emot denna data, sparas den i en medlemsmatris som representerar robotarna. Denna 3x3-matris representerar de tre robotarna i labbmiljön, där varje rad motsvarar en robot och varje rad består av tre kolumner men systemet har kapacitet att hantera upp till 11 robotar. Den första kolumnen är robotens tagg-ID, den andra kolumnen är hjärtslagstidsstämpel (heart-beat\_timestamp) och den sista kolumnen är en lista över de senaste detekteringarna som är en vektor för varje robot.

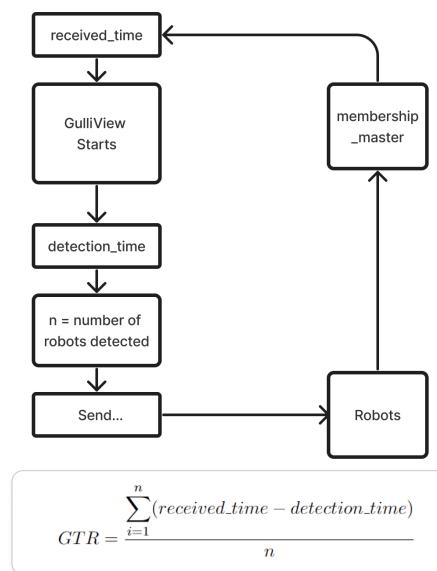
Efter varje nytt meddelande från robotarna uppdaterar "Membership\_master" denna matris. Därefter uppdaterar den en lista över de äldsta detekteringarna. Denna lista fylls med data från matrisen för att få den senaste positionen och tidsstämpelet för en robot, oavsett vilken robot som har den senaste datan. När denna lista uppdateras, skickas den vidare till GulliView.

Det fjärde steget inträffar också i GulliView-noden. I själva verket sker det fjär-

de steget samtidigt som det första steget, eftersom två olika funktionaliteter utförs, och därför delas det upp i två steg. I det fjärde steget erhåller GulliView meddelandet från Membership\_master. Efter detta genomförs beräkningen av Grupp Tur och Returtiden [1].

### 2.3.2 Grupp Tur och Returtiden (GTR)

Beräkningen av GTR (jfr en: Group Round Trip Time (GRTT)) utförs genom att GulliView tar först tiden precis innan den detekterar en robot (detectionTime\_ms) och bifogar den till detekterade meddelanden. Därefter skickar den dessa meddelanden till robotarna. Robotarna tar emot meddelandena och vidarebefordrar dem till Membership\_master. Efter att Membership\_master har mottagit meddelandena, skickar den dem tillbaka till GulliView. GulliView tar sedan tiden efter att den har mottagit och läst meddelandet från Membership\_master för att beräkna tidsintervallet. Slutligen beräknar GulliView det genomsnittliga tidsintervallet genom att ta summan av tidsskillnaden för varje robot och dividera det med antalet robotar (se figur 2.4).



**Figur 2.4:** Flödesdiagrammet illustrerar processen för att beräkna (GTR)

### 2.3.3 Interprocesskommunikation

Interprocesskommunikation (jfr en: Inter-Process Communication (IPC)) är en mekanism för att utbyta data mellan trådar i en eller flera processorer. Med andra ord har olika trådar eller processorer tillgång till gemensamma data. IPC är en betydelsefull teknik inom parallell programmering och ibland kallas det för intertrådkommunikation och interapplikationskommunikation. IPC teknik omfattar olika metoder såsom; File, Signal, Sockets, Semaphore, Pipes, Memory-mapped file, shared memory och så vidare [18]. I detta arbete har metoder som Memory-mapped file och shared memory använts mer frekvent om det kommer till metoden.

# 3

## Metod

Målen för detta arbete var att förbättra GulliViews prestanda genom mjukvaruförbättringar medan för medlemstjänsten bestod av att porta en äldre version av medlemstjänsten för användning av en snabbare GulliView-version. Slutligen att förbättra den befintliga sammanvävningsalgoritmen och att utveckla en lösning för korsningar. Följande kapitel beskriver metoden för att nå dessa mål.

### 3.1 GulliView Johannes Kristofer

Följande sektion utforskar de tekniska frågor som rör användadet av AprilTags och GulliViews prestanda. Möjliga lösningar inkluderar multitrådning samt förbättringar i bildbehandlingen som utförs i GulliView.

#### 3.1.1 Multitrådning

Koden i GulliView exekverades sekventiellt i den föregående arkitekturen vilket innebar att deluppgifter som är möjliga att exekvera parallellt var tvungna att vänta på de resterande steg från 2.1.2 som först måste exekveras. Genom att låta inläsningen från kamerorna ske på separata trådar så är det möjligt att effektivisera systemet därmed öka frekvensen som systemet körs i, samt reducera födröjningen av information i GulliView.

Den nya implementationen läser kontinuerligt in bilder i en separat tråd vilket gör det möjligt att ha en ny bild redan inläst vid iterationens start (se sektion 2.1.2). Detta gör det möjligt att spara tid då kameran kan börja starta inläsningen för nästa iteration innan den föregående iterationen är klar. Detta kan ses som ett klassiskt Producer-Consumer problem. Det finns två trådar och en gemensam buffer som båda trådarna använder sig av för att kommunicera med varandra. Den första tråden är ansvarig för att placera nya bilder i buffern och den andra tråden plockar den nyaste bilden från buffern, givet att den inte är samma som den föregående bilden. Buffern är implementerad som en array av storlek BUFFER\_SIZE. Det finns ingen övre gräns på hur stor BUFFER\_SIZE kan vara, men den undre gränsen behöver vara tillräckligt stor för att inte producer-tråden ska hinna ikapp consumer-tråden eftersom producern är den som bestämmer den maximala hastigheten programmet kan exekveras i. Detta minsta värde går att beräkna genom att mäta exekveringstiderna för båda trådarna separat och sätta dem i ekvation 3.1.

$$MIN\_BUFFER\_SIZE = \frac{time\_producer}{time\_consumer} \quad (3.1)$$

I fallet för GulliView så läser producer-tråden in bilder konstant i 60 FPS (60 bilder per sekund) vilket ger en tid på  $\frac{1}{60} = 16,67ms$ . Den högsta uppmätta tiden för consumer-tråden i GulliView var 250 ms, vilket sker när en exhaustive search genomförs. Dessa värden gav en minimum BUFFER\_SIZE på  $\frac{250ms}{16,67ms} = 15$ . Trots att detta värde ska räcka för att säkerställa att producern aldrig hinner ikapp consumern så användes ett värde på 64 då den extra minneskostnaden ansågs vara försumbar i jämförelse till prestandakostnaden som skulle ske ifall producern hann ikapp consumern.

Synkroniseringen mellan trådarna och buffern är låsfri vilket innebär att synkroniseringstyper som lås och semaforer inte behövs för att låsa buffern, utan algoritmerna garanterar att de båda trådarna inte kommer att behandla samma minnesområde samtidigt. Detta fungerar genom att låta de båda trådarna har varsin global atomisk integer, *producer\_idx* pekar på värdet där den nyaste bilden är placerad, respektive *consumer\_idx* som är positionen som consumer-tråden försöker läsa ifrån. Busy-waiting används då det finns två fall som kan uppstå vilket gör att det är lämpligt att pausa tråden i några millisekunder och detta värde kommer att kallas *t*.

Algoritm 1 visar pseudokoden för tråden som läser data från kameran till buffern.

---

**Algorithm 1** Skriv till buffer

---

**Require:**  $t \geq 0$

```

while true do
    next  $\leftarrow$  (write_idx + 1) mod BUFFER_SIZE
    while next = read_idx do sleep t ms
    end while
    buffer[next]  $\leftarrow$  {camera_frame}
    write_idx  $\leftarrow$  next
end while
```

---

*Next* är positionen efter den nyaste bilden och algoritmen kommer att köra sin busy-wait om denna position är samma som *consume\_idx*. Detta ska inte vara möjligt om *BUFFER\_SIZE* är ett tillräckligt stort värde enligt ekvation 3.1, utan existerar bara för att garantera att datan som consumer-tråden läser inte blir korrupt. Algoritm 2 visar algoritmen som behandlar inläsningen av bilderna från buffern till programmet.

---

**Algorithm 2** Läs från buffer

---

**Require:**  $t \geq 0$

```

while true do
    while write_idx = read_idx do sleep t ms
    end while
    reader_idx  $\leftarrow$  writer_idx
    fetched_frame  $\leftarrow$  buffer[reader_idx]
end while
```

---

Till skillnad från algoritm 1 så är busy-wait loopen nödvändig för att algoritm 2 ska fungera. Algoritmen väntar tills det finns en ny bild placerad i buffern och flyttar sedan sin *read\_idx* till positionen. Om flera bilder har skrivits in i buffern sedan föregående läsning så kommer algoritmen endast att läsa in den nyaste bilden vilket gör att gammal data inte används.

#### 3.1.2 Bildupplösning och bildfrekvens

Innan projektets start så hade GulliViews kameror endast möjlighet att filma i 1080p 30 FPS. Då de nya kamerorna som installerades till detta projekt har möjlighet att ta bilder i upp till 3840p 60 FPS, så gjordes tester för att identifiera möjligheterna med en högre kameraupplösning. Syftet med den ökade upplösningen var att ge GulliView bättre förutsättningar att identifiera fordonen och på så sätt öka samspelet mellan robotarna.

En högre upplösning gör att bilden blir skarpare samtidigt tar bildinläsning, bildbehandling och taggsökning längre tid att utföra. Av detta följer att GulliViews frekvens påverkas negativt av en högre upplösning. Det gjordes därför tester på hur stora dessa tidsökningar var i förhållande till den förbättrade bildupplösningen och hur dessa påverkade frekvensen i helhet. Samtidigt är bildfrekvensen hos kamerorna den begränsade faktorn i systemets prestanda. Om kamerorna kan ta bilder i 60Hz så är detta även den maximala hastigheten GulliView kan köra i.

#### 3.1.3 AprilTags

GulliView använde tidigare AprilTags av storlek 200x200mm vilket kräver att fordonen som taggarna är placerade på är tillräckligt stora för att ha plats för dem. En möjlighet som framkom med de nya kamerorna var användandet av taggar av mindre storlekar. Dessa taggar skulle kunna göra det möjligt att använda sig av mindre fordon än Wifibots som är de som just nu används.

Utöver 200mm så valdes 100mm och 50mm som kandidater för möjliga storlekar på AprilTags då dessa storlekar ansågs vara representativa för olika storlekar på fordon: stora, medel och små. De olika AprilTags testades individuellt för de olika valda upplösningarna från sektion 3.1.2 för att se hur bra GulliView kunde hantera de olika storlekarna.

Testerna som genomfördes bestod först av att se om taggen blev detekterad för de olika upplösningarna. Om detta var fallet så testades den högsta tillåtna hastigheten en AprilTag kunde förflytta sig utan att GulliView tappade bort den. För att testa detta så fick fordonen köra rakt över hela labbmiljön i hastigheten som testas, först åt ena hålet och sedan tillbaka. Om en tagg blev detekterad så höjdes hastigheten med 0,05m/s tills att testet misslyckades. Därefter sänktes hastigheten med 0,01m/s tills att testat passeras. Det sista uppmätta värdet är det som blir fastställt som den högsta hastigheten en tagg kan förflytta sig i GulliView.

### 3.1.4 Bildbehandling

För varje iteration (se sektion 2.1.2) så måste den nuvarande bilden transformeras för att ta bort förvrängningar som uppstår i bilden och orsakas av bland annat kameralinsens form. Detta är en dyr operation som kräver både tid och beräkningskraft. Tidigare användes metoden ”undistort” från OpenCV-biblioteket för att ta bort förvrängningarna. Denna metoden använder kameramatrisen och dess distanskoefficienter för att först beräkna en mappning som den sedan använder för att applicera en transformation på bilden. Detta är dock överflödigt då metoden för att beräkna mappningarna är dyr och ska endast behöva användas om kamerorna förflyttar sig då de inte ger någon ny information annars. Att få bort den konstanta ommappningen som skedde identifierades därför som en möjlig metod till att förbättra GulliViews prestanda.

För att lösa detta så bröts transformeringarfunktionen upp i två delar: en del för mappning av området som sparas under programmets gång, och en som transformrar bilden med hjälp av mappningarna. Fördelen med detta var att de initiala mappningarna nu kan beräknas samtidigt som kamerorna kalibreras vid GulliViews uppstart och sedan sparas i programmet för senare användning. Den nya transformeringsdelen är kvar i kodens while-loop, men då den nu kan använda sig av de redan framtagna mappningarna från kalibreringen så uppskattas beräkningarna att gå fortare.

### 3.1.5 Kamerasammansättning

Den föregående versionen av GulliView hade inte möjlighet till att detektera nya AprilTags som fördes in i ett område där en tagg redan existerade. Detta var en begränsning av hur fast-search algoritmen var designad. För att lösa detta så skapades en konstant *MIN\_GLOBAL\_SEARCH* som kör en exhaustive search om den inte tidigare körts inom *MIN\_GLOBAL\_SEARCH* iterationer i GulliView. Denna metod fungerar genom att garantera att GulliView alltid söker av hela området som minst var *MIN\_GLOBAL\_SEARCH*:te iteration. Detta gör att nya taggar alltid har möjlighet att bli detekterade även om GulliView redan följer andra taggar.

## 3.2 Manövrar Kristofer Soltani

Följande sektion kommer beskriva arbetet kring vidareutvecklingen av den tidigare sammanvävningsalgoritmen och skapandet av en ny algoritm för möte vid korsning. Arbetet startade med att förstå koden och att testa sammanvävningen med robotarna för att hitta potentiella förbättringar i den.

### 3.2.1 Sammanvävningsalgoritmen

Algoritmen som implementerats tidigare har varit baserats på att robotarna ska hålla en viss distans mellan sig. Detta innebär att robotarna med hjälp av GulliView beräknade hur långt kvar varje robot har till en designerad sammanvävpunkt

där robotarna först delar körfält. Bristerna med detta var till exempel när hastigheten på fordonen var snabbare än  $0,3m/s$  så var det hårdkodade säkerhetsavståndet på  $1m$  för kort. Den tidigare koden hade inte heller någon hastighetsimplementering vilket gjorde att roboten som körde fortast och antagligen skulle nå sammanvävningspunkten först var tvungen att bromsa på grund av att den var längre från punkten. För att lösa problemen implementerades hastigheterna de båda robotarna har för att kunna få ut vilken tid de har kvar till sammanvävningspunkten. Genom att bestämma säkerhetsavstånd och prioritet för robotarna baserat på tid istället för sträcka blir systemet mer dynamiskt vilket löser de båda problemen.

En vanlig tumregel för bilister i vanlig trafik att förhålla sig efter är tresekundersregeln [19] som innebär att tre sekunder ska hållas till framförvarande fordon. Denna referens valdes till ett säkerhetsavståndet även i detta arbete. Att använda tidmätning löser även det andra problemet genom att jämföra vilken robot som har minst tid kvar till sammanvävningspunkten och ge den prioritet att köra snabbare medan den andra bromsar in.

Användningen av de nya algoritmerna förklaras genom att varje gång ett meddelande tas emot från GulliView, över UDP, kollar algoritmen om båda robotarna detekterades av kamerorna. Om båda blivit detekterade beräknas avståndet till punkten där de kommer att dela körfält. Avståndet divideras med deras nuvarande hastighet och tiden till punkten fås och används för att jämföra vilken av robotarna som når punkten först. Om tidsskillnaden mellan robotarna är mindre än den önskad konstanten, vilket är tre sekunder i vårt arbete, beräknas den nya hastigheten om för roboten som är längst ifrån punkten tidsmässigt. Denna robot ska minska sin hastighet så att tidsskillnaden uppnås vid sammanvävningspunkten. Innan ett meddelande skickas ut görs en kontroll så att inte någon av hastigheterna har överstigit den max tillåtna hastigheten på vägen. Detta är också hastigheten som den roboten som ligger först meddelas att den ska köra i. När båda robotarna passerat sammanvävningspunkten avslutas beräkningarna.

Ett fel som upptäcktes med programmet som körde föregående implementation var att det skickade meddelanden med föråldrad information. Varje meddelande som togs emot hanterades två gånger, en gång för varje robot, och skickade därför två meddelanden. I den första hanteringen hade den ena robotens position inte uppdaterats i beräkningarna. Detta åtgärdades och innebar att föråldrad information angående vilken hastighet som fordonen skulle hålla inte längre skickades ut. Programmet var också tidigare beroende av ROS men när den nya algoritmen skapades fokuserades det på att ta bort beroendet av ROS för att kunna köra den på en dator utan ROS installerat.

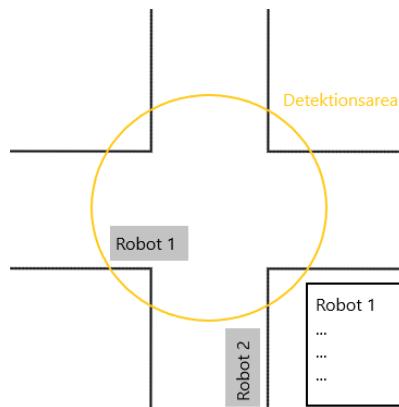
Problem som stöttes på i implementeringen var bland annat hur man bör beräkna sträckan om den involverar kurvor. Avståndsberäkningen räknar fram kortaste sträckan mellan två punkter och eftersom ramp-vägen involverar en sväng i sin väg behövdes detta lösas. Robotarna körde inte heller exakt samma väg varje gång på grund av att det tillåts en felsmärginal på 20cm runt varje vägpunkt. Beslutet togs att

vägen till sammanvävpunkten fick bli uppdelade i två delar för ramp-roboten. När roboten är på rampen så räknas sträckan den har till slutet på rampen adderat med hur långt det är emellan slutet på rampen och sammanvävpunkten. Detta gör att uträkningen blir mer exakt eftersom vägen blir mer lik hur roboten verkligen kör.

### 3.2.2 Korsningar

För att hantera korsningar användes ett arbete av Savic et al. [20] som inspiration. Till skillnad från det arbetet är vår implementation inte distribuerad. Istället fungerar den på liknande sätt som sammanväpningen där en dator övervakar korsningen och meddelar robotarna hur de ska anpassa sin hastighet.

I figur 3.1 visas hur korsningen övervakas. När en robot närmar sig korsningen blir den detekterad och läggs till i en kö med ett unikt id. Om roboten är först i kön har den prioritet och kan fortsätta. Om en robot läggs till i kön och inte är först får den stanna. En utökning gjordes där detekteringen av robotarna gjordes tidigare för att försöka sakta ner robotarna utan att någon behöver stanna helt. Liknande lösningen för sammanväpning i avsnitt 3.2.1 användes avståndet och hastigheten till mittpunkten på korsningen för att avgöra vilken robot som ska sakta ner.



**Figur 3.1:** Bild över korsningen. När en robot upptäcks inom den gula cirkeln läggs de till i en kö. Först i kön får prioritet och kör vidare, senare ankomster stannar.

Första problemet som stöttes på vid skapandet av den nya korsningsalgoritmen var att avgöra var på banan roboten ska börja räkna hur långt det är kvar till korsningen. Det bästa stället att börja räkna på ansågs vara när robotarna gör sina sista svängar in mot korsningen. Vinkeln theta och x-koordinaten markerar när robotarna befinner sig just vid dessa positioner.

Samma algoritm som i sammanvävpunktsdelen användes så roboten som har kortast tid kvar till korsningen tillåts fortsätta utan att ändra hastighet. För den andra roboten beräknas den hastighet den bör hålla så tidsskillnaden ökar till en konstant INTERSECTION\_WINDOW så att risken för kollision minskar. Ett meddelande skickas ut så att roboten kan korrigera sin hastighet. När den första roboten lämnar

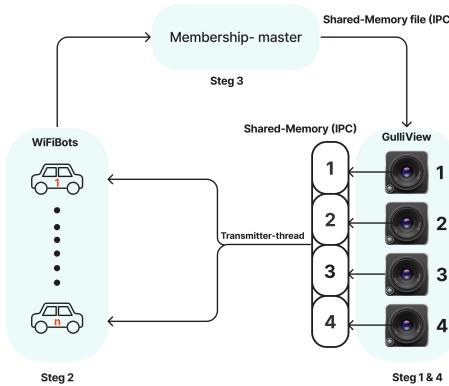
korsningen skickas nästa meddelande att den andra kan köra i den maximalt tillåtna hastigheten.

### 3.3 Medlemstjänsten Shirzad Keivan

Medlemstjänstens utvecklingsprocess genomgick två faser. I den första fasen bestod arbetet av att integrera kodförråden (jfr engelska repositories) "Membership\_service\_2023" och "advanced\_mobility\_model". Detta var nödvändigt eftersom den tidigare medlemstjänsten inte var anpassad till den senaste versionen av GulliView som finns i "advanced\_mobility\_model"-repository. I den andra fasen genomfördes ändringar och förbättringar av medlemstjänstens arkitektur.

#### 3.3.1 Uppdaterad arkitektur Shirzad Keivan

Funktionaliteten hos medlemstjänsten förblir densamma som tidigare, och bearbetningen och beräkningarna fortsätter att ske i fyra steg, vilket beskrivs i sektion 2.3.1. Två förändringar har emellertid implementerats, den första förändringen har skett i kommunikationen mellan noderna "Membership\_master" och "GulliView". I den tidigare implementationen skickade "Membership\_master" meddelanden till GulliView genom multisändning (multicast) via UDP-sockets [1], trots att båda noderna körs på samma dator. Nu sker kommunikationen mellan dessa noder via en så kallad IPC mekanism, där tekniken som används benämns "Memory-mapped file" (se avsnitt 3.3.3). Den andra uppdateringen ägde rum mellan de fyra instanserna av GulliView. I den tidigare lösningen sände varje instans av GulliView ut ett detekteringsmeddelande till robotarna via UDP (jfr en: broadcasting). Nu används istället en metod som kallas för delat minne (jfr en: Shared Memory) (se avsnitt 3.3.2).



**Figur 3.2:** Den reviderade arkitekturen för medlemstjänsten utnyttjar (IPC) tillsammans med en universell sändare.

### 3.3.2 Sammanhängande meddelande via en centraliserad avsändare **Shirzad Keivan**

För att åstadkomma en enskild sändare från GulliView till robotarna istället för att ha fyra avsändare av GulliView som bredsänder (jfr en: broadcast), har en ny metod som använder delat minne implementerats. Som kort nämnts i avsnitt 2.3.3, är delat minne en mekanism inom IPC. Denna mekanism ingår i så-kallad parallelprogrammering (jfr en: concurrent programming), vilket innebär att varje instans av GulliView delar gemensamt minne med en separat tråd eller processor som kör en C++ kod ”Transmitter”.

Varje instans av GulliView skapar ett delade minnet (totalt 4 delade minnen, en för varje instans) och när en robot detekteras så skriver den detektionsmeddelanden till det delade minnet. Samtidigt skapar ”Transmitter” en tom matris för att lagra den senaste datan. När ”Transmitter” läser detektionsmeddelandena från det delade minnet, jämför den meddelandena med varandra och uppdaterar matrisen med det senaste detektionsmeddelandet. Därefter bredsänder den matrisen vidare till roboterna via UDP.

Inom datavetenskap kallas detta fenomen läsare-skrivare (jfr en: Readers-Writers). Eftersom både läsare och skrivare försöker att utföra ändringar i det delade minnet samtidigt uppstår ett problem som kallas läsare-skrivareproblem (jfr en: readers-writers problems) [21]. Problemet uppstår när två eller fler processer försöker utföra en kritisk sektion (jfr en: Critical section) samtidigt i *det här fallet är det delade minnet den kritiska sektionen*, vilket kan leda till inkonsekventa resultat eller kollisioner. För att lösa detta behöver programmet synkroniseras så att endast en process åt gången kan exekvera den kritiska sektionen. För detta ändamål används en kombination av en semafor och en flagga, där semaforen fungerar som en mekanism för att reglera tillgången till den kritiska sektionen och flaggan, benämnd i koden som ”flag”, indikerar om den kritiska informationen har lästs eller inte (där värdet 1 indikerar att det finns data i det delade minnet att läsa och värdet 0 indikerar att informationen har lästs). Genom att använda denna synkroniseringssmetod säkerställs ömsesidig uteslutning (jfr en: Mutual Exclusion), vilket betyder att endast en process åt gången kan exekvera den kritiska sektionen, vilket minskar risken för konflikter och inkonsekventa resultat. Med andra ord, när en instans av GulliView vill skriva i det delade minnet tar den semaforen (jfr en: acquire semaphore), skriver i det delade minnet, sätter flaggan till 1 och släpper sedan semaforen (jfr en: release semaphore). Därefter tar ”Transmitter” semaforen, läser data från det delade minnet, skickar det vidare till roboterna, sätter flaggan till 0 och släpper slutligen semaforen.

### 3.3.3 Övergång från nätverkssändning till delat minne **Soltani Keivan**

Tidigare användes UDP-sockets för kommunikation mellan Membership\_master och GulliView. Denna metod hade sina nackdelar såsom UDP-bufferöverbelastning och

paketförlust, vilket kan uppstå när data inte når sin avsedda destination. För att lösa dessa problem användes en pausfunktion (sleep) i den tidigare implementationen för att synkronisera kommunikationen och undvika överbelastning av UDP-bufferten genom att begränsa datatrafiken, det medför tyvärr att GTR-värdet ökar.

För att adressera dessa utmaningar har en IPC method implementerats, vilken har lett till en förbättrad implementation utan behov av att inaktivera processorn (CPU), vilket i sin tur har resulterat i snabbare kommunikation och förbättrad GTR. För att implementera IPC mekanismen mellan GulliView och Membership\_master, valdes att använda en delad minnesfil. Detta möjliggör kommunikation mellan Python-kod (som utgör Membership\_master) och C++-kod (som utgör GulliView). Fyra instanser av GulliView körs och behöver läsa data från mastern. Eftersom det endast finns en som skriver data ger användningen av en delad minnesfil flera fördelar.

- Enkel datatillgång: GulliView-instanserna kan enkelt läsa data från masterprocessen genom att använda en delad minnesfil, vilket消除ar behovet av att etablera komplicerade dataöverföringsprotokoll.
- Snabb dataöverföring: Delade minnesfiler möjliggör snabb dataöverföring eftersom data delas direkt mellan processer utan behov av att kopiera den.
- Effektiv resursanvändning: Användningen av delad minnesfil är resurseffektiv eftersom den消除ar behovet av att skapa och förstöra datastrukturer för kommunikation mellan processer.

För att implementera detta i Python användes tre bibliotek: posix\_ipc, mmap och os. För att säkerställa synkronisering mellan Membership\_master (skrivare) och GulliView (läsare) användes även en semafor. Detta förhindrar att mastern tar bort data från delat minne samtidigt som GulliView läser. Semaforen implementerades med hjälp av posix\_ipc. Först användes **os** för att skapa en delad minnesfil med en storlek på 1024 byte, tillräckligt stor för att rymma den senaste detektionsmatrisen. Den delade minnesfilen måste sedan mappas till ett Python-objekt med mmap. I en while-loop läser Membership\_master data från UDP och uppdaterar den senaste detektionsmatrisen enligt steg 3 i avsnitt 2.3.1. Denna matris kodas sedan till data som skrivas till den delade minnesfilen. Innan skrivning läser Membership\_master semaforen för att förhindra att GulliView öppnar filen samtidigt. Därefter rensas minnesfilen och ny data skrivas in, varefter semaforen frisläpps. Denna process upprepas kontinuerligt. I GulliView noden används samma semafor och den delade minnesfilen som skapades av Membership\_master. GulliView försöker att få semaforen och väntar om den inte är tillgänglig. När semaforen är tillgänglig läser GulliView data från den delade minnesfilen, konverterar den till lämpliga format (t.ex. heltal eller flyttal), och behandlar den och sedan beräknar GTR enligt 2.3.2.

# 4

## Utvärdering

Detta kapitel tar upp de kriterier och tillvägagångssätt som används för att utvärdera metoden.

### 4.1 GulliView Johannes Soltani

För att bedöma effektiviteten av metoderna för GulliView så togs ett antal utvärderingskriterier fram för att objektivt bedöma förändringarna mellan de olika implementationerna.

#### 4.1.1 Tidsprofilering

För att utvärdera förändringarna av de olika beståndsdelarna i GulliView så användes tidsprofilering där tiden uppmättes i ms. Tidsprofileringen gjorde det möjligt att testa tidsförändringen av delarna i GulliView oberoende av varandra då start och sluttid uppmättes för varje del. Tiden för de olika delarna uppmättes först för den föregående implementationen och sedan för den nuvarande.

Alla tester gjordes med samtliga fyra kameror igång samtidigt där respektive kamera körde mjukvaran 10 000 gånger, vilket gav 40 000 datapunkter för varje test. Dessa resultat sammanställdes och medelvärdet beräknades från datan. Medelvärdet användes sedan för jämförelser mellan implementationer.

En förbättring av en implementation skulle innebära att den nya implementationen har en lägre uppmätt exekveringstid än den föregående. Den minskade tiden innebär en förbättring i koden med avseende på tidseffektivisering om den nya implementationen uppfyller samma effekt som den föregående.

#### 4.1.2 Prestanda

För att göra det möjligt att bedöma prestandaförbättringar i GulliView så behövdes ett mått för att bestämma den totala prestandan i GulliView. Måttet som tidigare används uppmäta hastigheten som GulliView körs är dess frekvens mätt i  $Hz$ .

$$Hz = \frac{1}{t} \tag{4.1}$$

Då frekvensen fluktuerar mellan iterationer så är det inte tillräckligt att bara ta ett värde för frekvensen vid en godtycklig tidpunkt. Den uppskattade frekvensen  $\widehat{Hz}$  som systemet kör i tas fram genom att låta while-loopen köra  $i$  iterationer för att sedan ta medelvärdet av alla uppmätta tider  $t_1 + \dots + t_{fps}$  enligt ekvation 4.2.

Frekvensen av programmet är begränsat till bildfrekvensen av de använda kamerorna vilket betyder att medelvärdet av  $fps$  antal uppmätta tider är en bra uppskattning på den verkliga frekvensen.

$$\widehat{Hz} = \frac{1}{fps} \sum_{k=1}^{fps} \frac{1}{t_k} \quad (4.2)$$

Trots att ekvation 4.2 är en bra uppskattning på programmets frekvens i stunden så finns det egenskaper i programmets funktionalitet som gör att medelvärdet över flera uppskattade värden behövs. Till exempel så används olika algoritmer i GulliView för att söka efter AprilTags vilka skiljer sig till stor del i tidsåtgång. Det enklaste sättet att hantera detta är genom att införa en konstant  $m$  som används för att beräkna medelvärdet av  $m$  stycken uppskattade värden enligt ekvation 4.3.

$$\widehat{Hz} = \frac{1}{fps \cdot m} \sum_{k=1}^m \widehat{Hz}_k \quad (4.3)$$

På samma sätt som tidsprofileringen i sektion 4.1.1 så användes alla fyra kamerorna samtidigt och ködde 10 000 iterationer vardera. Skillnaden i detta fall är att medelfrekvensen beräknas var 60: nde iteration då bildfrekvensen är 60. Antalet datapunkter av frekvensen som fås beräknades med ekvation 4.4.

$$\text{antal datapunkter} = \left\lfloor \frac{\text{iterationer}}{fps} \right\rfloor \quad (4.4)$$

Ett test på 10 000 iterationer och en bildfrekvens på 60 FPS ger då  $\left\lfloor \frac{10000}{60} \right\rfloor = 166$  datapunkter. Eftersom datapunkterna fortfarande är framtagna över 10 000 iterationer så håller datapunkterna en hög noggrannhet. Av denna anledning kan medelvärdet av dessa värden användas för jämförelser mellan implementationer.

### 4.1.3 Maximal hastighet

Apriltaggen startade utanför synfältet för kameran och ködde sedan i testhastigheten i parallell riktning genom fältet tills att taggen var ute igen. För att säkerställa att systemet inte bara följer taggen i en riktning så testades motsatt riktning på samma sätt analogt.

Kravet för att ett test skulle bli godkänt var att GulliView skulle kunna identifiera taggen för varje bild tagen av kameran under hela testets gång. Om GulliView skulle tappa taggen i en bild så kommer testet att underkännas och nästa hastighet i testet startas. Detta innebär att den högsta hastigheten som passerade testet för en viss storlek av en tagg sparas som det bästa uppmätta värdet. Anledningen till att det högsta värdet anses vara det bästa är eftersom om testet för en högsta hastighet  $v_{max} > 0$  passerar, så innebär det även att alla hastigheter  $0 \leq v \leq v_{max}$  passerar testet. Användandet av  $v_{max}$  som utvärderingsvärde säkerställer att om  $v_{max}$  passerar, så leder det till att det maximala antalet hastigheter blir godkända för GulliView.

## 4.2 Manövrar Jakob Shirzad

För att säkerställa att manövrarna utförs säkert och effektivt genomfördes ett antal tester. Robotarna startades från samma startpunkter men deras starthastighet och den maximalt tillåtna hastigheten på den gemensamma vägen varierades. För att utvärdera säkerheten användes Cut-in risk index (CRI) som ger ett värde mellan 0 och 2 där noll visar låg risk för kollision och 2 väldigt hög risk [22]. CRI beräknas med hjälp av time to collision (TTC) och avståndet mellan fordonen.

TTC beräknas enligt följande ekvation där  $d_a$  är avståndet mellan fordonen och  $v$  är deras respektive hastigheter:

$$TTC_{ij} = \frac{d_a}{v_i - v_j} \forall v_i > v_j \quad (4.5)$$

CRI fås därefter av:

$$CRI = \begin{cases} \exp(-TTC * k_a) & \forall d_a \geq 0 \wedge d_b \geq 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.6)$$

där  $k_a$  fås av följande ekvation:

$$k_a = \frac{d_a}{d_a + d_b} \quad (4.7)$$

Vanligtvis används CRI när ett fordon gör en sammanvävning där det finns andra fordon både framför och bakom. Totala CRI beräknas då genom att summa ett enskilt CRI för fordonet framför och för fordonet bakom. I testerna används endast två fordon men för att kunna göra beräkningar med CRI behövs två avstånd. Ett tredje fordon simuleras därför där avståndet  $d_b = 1,5$  m. Avståndet väljs då det är det maximala avståndet robotarna har sensorer för, i det här fallet IR-sensorerna, som föreslaget av författarna till metoden [22]. Värdet för CRI hamnar därmed mellan 0 och 1.

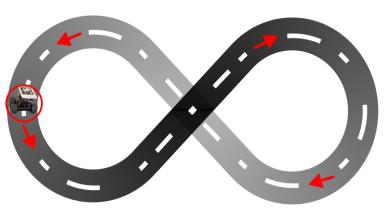
TTC är endast definierat när det bakomliggande fordonet kör i högre hastighet än det framförliggande eller när det finns en risk för att köra in i sidan när fordonet som ska köra in på vägen är i jämnhöjd med fordonet redan där. För att fortsatt få ett CRI används därför den modifierade varianten föreslagen av Selpi et al. [17] där endast de relativta avstånden används när TTC inte är definierat.

CRI förväntas minska ju närmare sammanvävningspunkten robotarna är och borde vara konsekvent i olika hastigheter. Förutom CRI dokumenteras även tiden mellan robotarna vid sammanvävningspunkten för att se att det önskade värdet upprättahålls och för att beräkna en genomströmningshastighet och avvikelsen mot det önskade värdet som i våra experiment sätts till 3 s.

Testerna för korsningen testar att inte två robotar kan köra in i korsningen samtidigt. För att testa ett sådant scenario skapades en bana likt en oändlighetsfigur där robotarna kunde köra i en slinga där de möts i en korsning flera gånger. Dessa utförs med en maxhastighet på 0,3 m/s då de vid högre hastighet missar sin rutt på grund av de skarpa svängarna.

## 4.3 Medlemstjänst **Shirzad Jakob**

Genom att säkerställa medlemstjänstens funktionalitet och erhålla tillförlitliga GTR utfördes ett antal olika tester. Testerna genomfördes genom att skriva ett Python-script för att köra robotarna i oändlighetsfiguren (se figur 4.1) för att erhålla omfattande och trovärdiga resultat. Att köra testerna på detta sätt säkerställde att få en mångsidig resultatuppsättning. Den komplexa testkörningen betonar medlemstjänstens funktionalitet och visar dess prestanda eftersom robotarna körs i oändlighetsfiguren, vilket täcker den möjliga ytan i labbmiljön istället för bara en del av det. Att köra en omfattande test är av yttersta vikt eftersom det visar hur GulliView upptäcker robotarna i olika positioner, såsom i kanterna, mitten och överlappade områden mellan olika kameror. Testerna uppdelades i två delar.



**Figur 4.1:** Oändlighetens symbol som förtydligar en omfattande testkörning.

### 4.3.1 K-värdeseffekt på GTR **Soltani Jakob**

K-värdet i Gulliview representerar MIN\_GLOBAL\_SEARCH (läs sektion 3.1.5). Det bestämmer frekvensen för uttömmande sökning, det vill säga hur ofta den ska köras. Valet av K-värdet är essentiell för GTR. Om K-värdet är för lågt kan GTR öka avsevärt, medan om det är för högt kan nya AprilTags missas. För att undersöka K-värdets påverkan på GTR och hitta det optimala värdet som balanserar både en låg GTR och upptäckt av nya AprilTag, genomfördes sex omgångar av tester vid en hastighet på 0.5 m/s med K-värdena: 1, 2, 4, 8, 16, 32. Testerna utfördes med en Wifibot med hastigheten 0,5 m/s och varje test set kördes tre gånger för få mer data och säkerställa noggrannheten.

### 4.3.2 Hastighetens påverkan på GTR **Shirzad Jakob**

För att undersöka hastighetens påverkan på GTR:n genomfördes tre omgångar av tester vid olika hastigheter: 0.5 m/s, 0.7 m/s och 1.0 m/s. Testerna genomfördes med användning av två Wifibot-enheter som körde med samma hastighet under varje testomgång. Varje testomgång utfördes tre gånger för att säkerställa noggrannheten och minimera felmarginalen vid insamlingen av data, vilket i detta fall representerade GTR-värdet. Vidare var K-värdet konsekvent 16, då utvärderingen av olika K-värden visade att det optimala värdet för att uppnå bästa möjliga resultat för GTR under samtliga testtillfällen var 16.

# 5

## Resultat

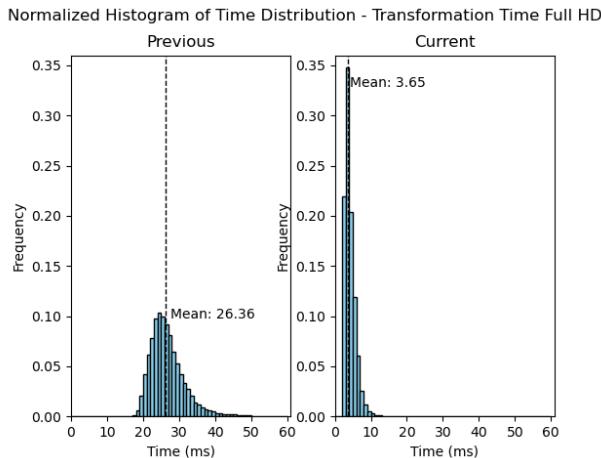
Detta kapitel redovisar resultatet av arbetet med de utvärderingsmetoder som togs upp i föregående kapitel.

### 5.1 GulliView Keivan Jakob

Resultats har tagits fram där tiderna från den äldre och nyare versionen mäts och jämförs i både histogram och genomsnittlig tid. Resultaten presenteras genom att först illustrera tidsfördelningen i form av normaliserade histogram, och därefter jämföra den genomsnittliga tiden som den gamla respektive den nya algoritmen producerar.

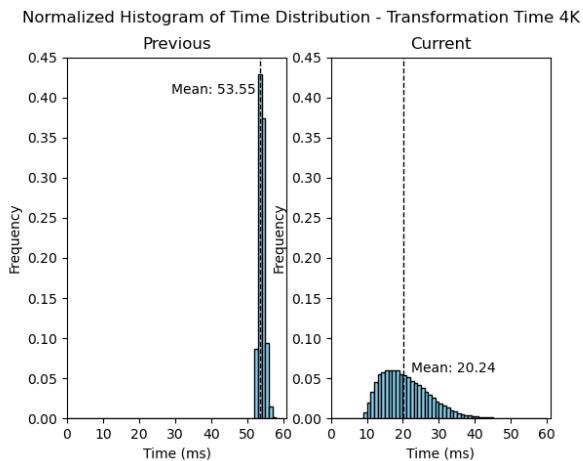
#### 5.1.1 Bildtransformering

Följande figurer kommer att jämföra den nya algoritmens tidsfördelning med den gamla för transformeringen på 4k och Full HD upplösning.



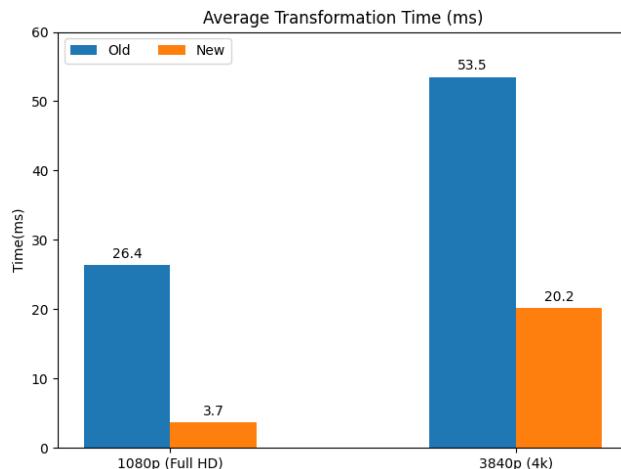
**Figur 5.1:** Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer.

Figur 5.1 visar transformeringstiden på Full HD där den nya algoritmen till höger visar både mindre varians och lägre värden i jämförelse till den gamla till vänster. Figur 5.2 visar att den nya algoritmens transformeringstid har ett bredare intervall på 4k än Full HD 5.1, där stor majoritet är mellan 10 och 30 millisekunder. Det



**Figur 5.2:** Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer.

noteras att den äldre algoritmen till vänster har drygt 99% av sina tider mellan 50 och 60 millisekunder.



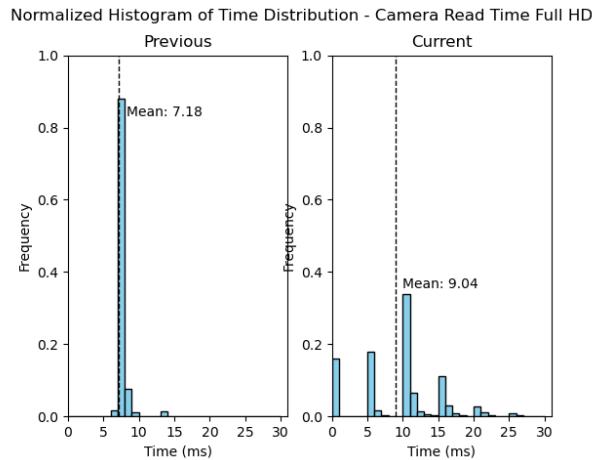
**Figur 5.3:** Grupperade stapeldiagram som jämför den genomsnittliga exekveringstiden mellan den nya och gamla transformationsalgoritmen på 4k och Full HD upplösning.

Figur 5.3 visar den genomsnittliga tiden som har extraherats från varje test med de olika upplösningarna på både den nya och gamla algoritmen i form av grupperade stapeldiagram och gör det således tydligare att se prestandaskillnaderna.

Diagrammen visar att den genomsnittliga tiden på Full HD har gått från 26,4 ms till 3,7 ms, det vill säga den är drygt 7 gånger snabbare än vad den var innan. På 4k har det gått från ett genomsnitt på 53,5 ms till 20,2 ms, det vill säga drygt 2,6 gånger snabbare än vad det var innan. Alltså fastställs en tidsförbättring på 600% och 160% för Full HD respektive 4k upplösningen.

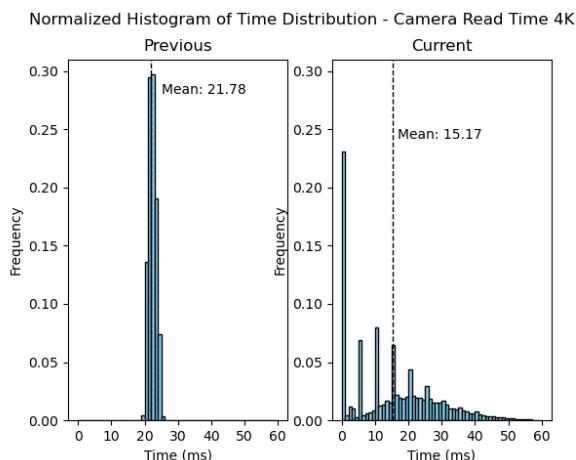
### 5.1.2 Kameräläsning

Följande figurer kommer att jämföra den nya algoritmens tidsfördelning med den gamla för kameräläsningen på 4k och Full HD upplösning.



**Figur 5.4:** Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer.

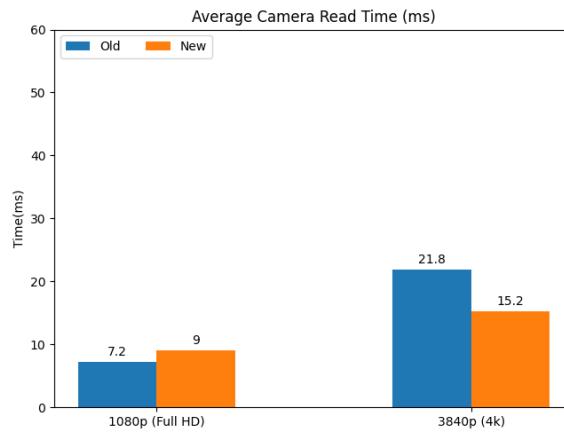
Figur 5.4 visar en ojämn fördelning på Full HD upplösningen för den nya algoritmen till höger. Majoriteten av tiderna är mellan 0 och 10 millisekunder med nästintill alla procentenheter fördelade på 0, 5, 10 och 15 millisekunder. Jämför detta med histogrammet till vänster där det noteras att nästan 90% av tiderna tar 7 sekunder för den gamla algoritmen. Förklaring till detta kommer i diskussionen.



**Figur 5.5:** Normaliserade histogram för kamerans läsningstider från 10 000 iterationer från respektive kamera på 4k upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer.

Figur 5.5 visar fördelningen gällande hur snabbt kamerans bild läses in på 4k upplösning med den nya algoritmen till höger. Det noteras en väldigt jämn kurva med enstaka spikar som sticker ut och en större en spik på 0 ms. Den äldre algoritmen

visar en mer koncentrerad fördelning där drygt 99% av tiderna sker mellan 19 och 26 millisekunder.

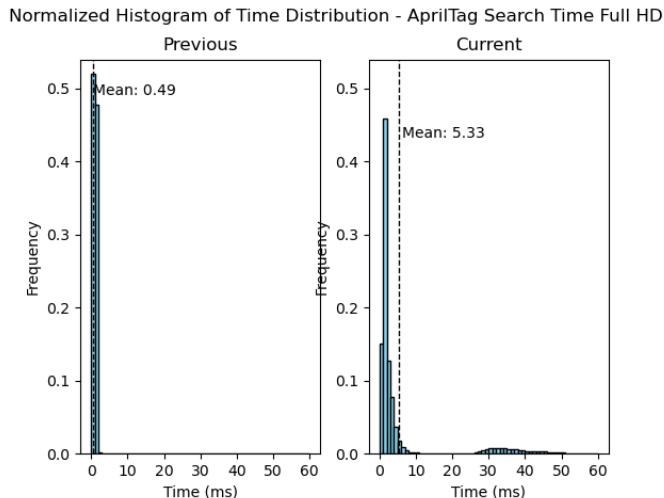


**Figur 5.6:** Grupperade stapeldiagram som jämför den genomsnittliga exekverings-tiden mellan den nya och gamla kameraläsningsalgoritmen på 4k och Full HD upplösning.

Jämförelsen av den genomsnittliga tiden från figur 5.6 visar att den nya algoritmen ökar tiden från 7,2 ms till 9 ms på Full HD, det vill säga en ökning på 25%. På 4k uppfattas det istället att den nya algoritmen sänkt lästiden från 21,8 ms till 15,2 ms, det vill säga den nya algoritmen är nästan 44% snabbare.

### 5.1.3 Taggsökning

Följande figurer kommer att jämföra den nya algoritmens tidsfördelning med den gamla för hur snabbt en AprilTag detekteras på 4k och Full HD upplösning.

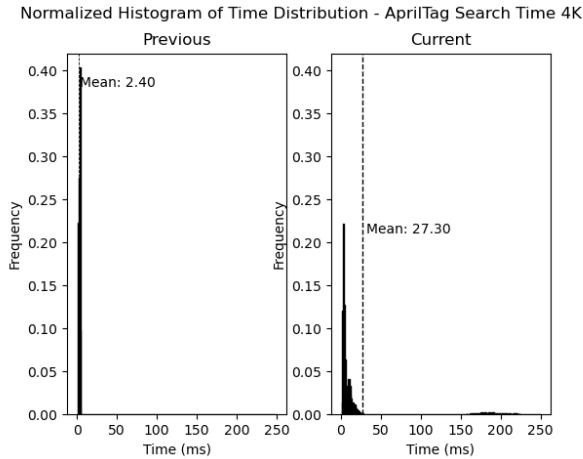


**Figur 5.7:** Normaliserade histogram för tagg sökningstider från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer.

## 5. Resultat

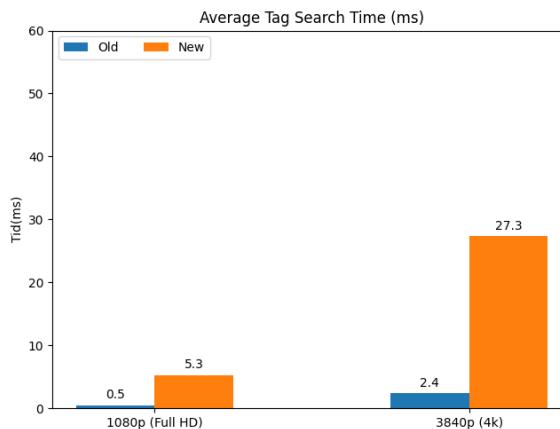
---

I figur 5.7 befinner sig en stor majoritet av tiderna för den nya algoritmen mellan 0 och 5 ms, vilket ses i histogrammet till höger. Det förekommer en låg frekvens av tider mellan strax under 30 och strax över 50 ms. Den äldre algoritmen visar ett jämnare beteende med drygt 99% fördelat på 0 och 1 millisekund.



**Figur 5.8:** Normaliserade histogram för tagg sökningstider från 10 000 iterationer från respektive kamera på 4k upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer.

I figur 5.8 visar den nya algoritmen till höger på 4k en fördelning som liknar resultatet från Full HD. Det noteras ett stort intervall där den stora majoriteten befinner sig mellan 0 och 25 ms. Det finns en låg frekvens mellan 150 och 250 ms. Den äldre algoritmen till vänster visar också här en mer jämn fördelning såsom i figur 5.7.

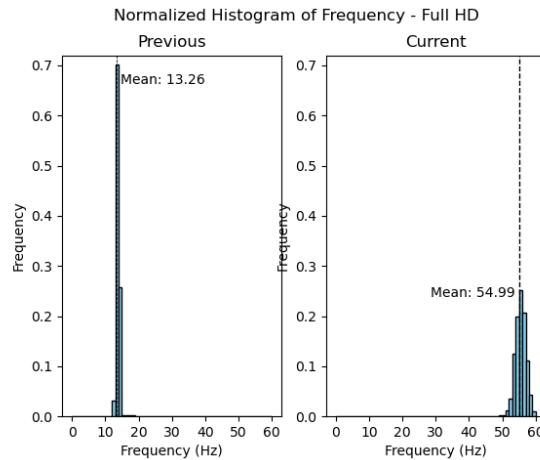


**Figur 5.9:** Grupperade stapeldiagram som jämför den genomsnittliga exekveringstiden mellan den nya och gamla taggsökningsalgoritmen på 4k och Full HD upplösning.

Figur 5.9 visar på sämre resultat i jämförelse med den tidigare versionen. På Full HD upplösningen noteras en ökning på 960%, och på 4k upplösningen noteras det en ökning på 1037,5%.

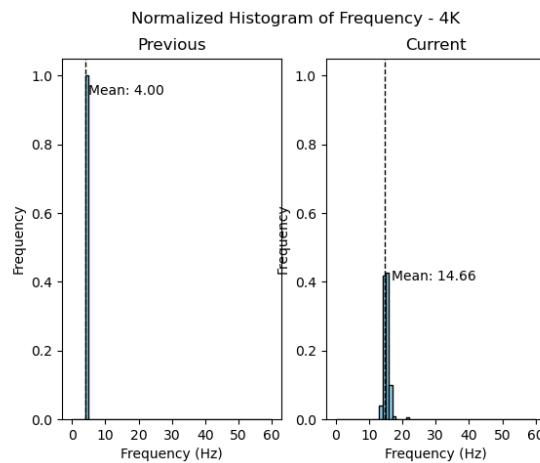
### 5.1.4 Prestanda

Följande figurer kommer att jämföra den nya algoritmens prestanda med den gamla på 4k och Full HD upplösning.



**Figur 5.10:** Normaliserade histogram för frekvensmätning från 166 datapunkter uttaget varje 60:e datapunkt från 10 000 iterationer från respektive kamera på Full HD upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer och 664 datapunkter.

Figur 5.10 jämför den gamla algoritmen till vänster med den nya till höger på Full HD. Det noteras en mer normalfördelad kurva med ett snitt på 55 Hz, medan den gamla algoritmen producerar ett histogram med en väldig liten varians och ett snitt på 13,3 Hz. Det noteras därav en förbättring på drygt 300%. Notera att högre frekvens är bättre.



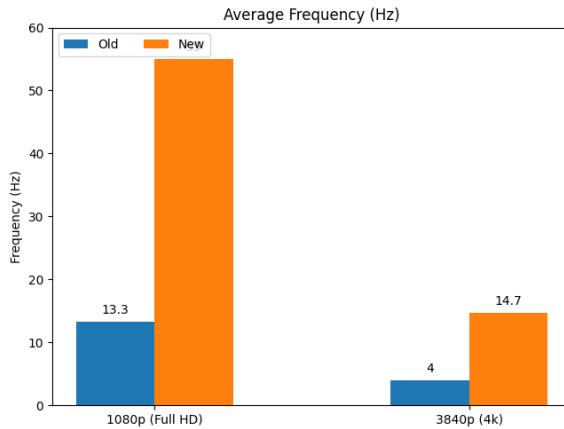
**Figur 5.11:** Normaliserade histogram för frekvensmätning från 166 datapunkter uttaget varje 60:e datapunkt från 10 000 iterationer från respektive kamera på 4k upplösning med nya algoritmen till höger och gamla till vänster, sammanlagt 40 000 iterationer och 664 datapunkter.

Figur 5.11 visar frekvensfördelningen mellan gamla och nya algoritmen på 4k upp-

## 5. Resultat

---

lösning. 100% av frekvensen på den gamla till vänster är på 4 Hz, medan den nya har en fördelning mellan 13 Hz och 20 Hz. Den genomsnittliga frekvensen har ökat från 4 Hz till 14,7 Hz, vilket blir en ökning på 267,5%. Notera att högre frekvens är bättre.



**Figur 5.12:** Grupperade stapeldiagram som jämför den genomsnittliga frekvensen mellan den nya och gamla algoritmen på 4k och Full HD upplösning.

**Tabell 5.1:** Maximala hastigheterna mätt i m/s som robotarna kan köra i Full HD där GulliView kan registrera dess AprilTags med K=16. Med 200 mm taggarna nåddes maxhastigheten på roboten utan förlust av detektion.

Full HD	200 mm	100 mm	50 mm
Previous	no limit measured	0,88	undetected
Current	no limit measured	0,85	undetected

**Tabell 5.2:** Maximala hastigheterna mätt i m/s som robotarna kan köra i 4k där GulliView kan registrera dess AprilTags med K=8.

4K	200 mm	100 mm	50 mm
Previous	0,61	0,54	0,29
Current	0,56	0,6	0,33

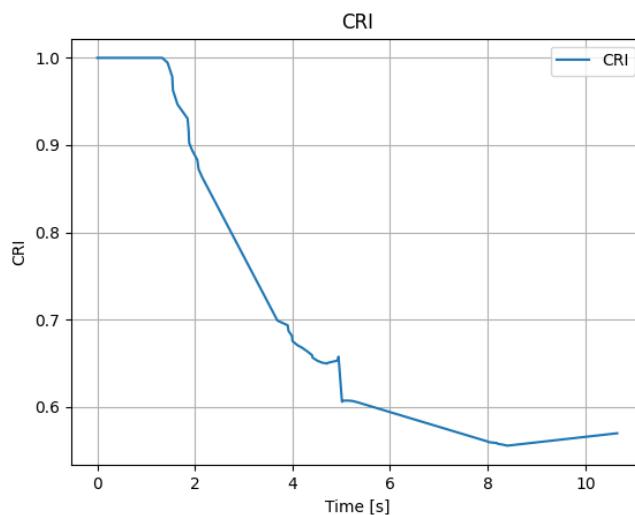
## 5.2 Manövrar Jakob Shirzad

I detta avsnitt presenteras resultaten från experimenten med manövrarna som utförts med två robotar.

### 5.2.1 Sammanvävning

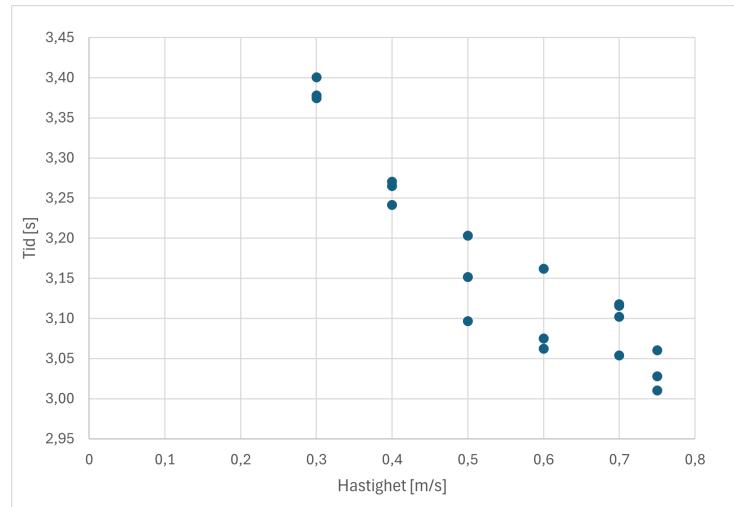
Resultaten för sammanvävningen visas genom två olika typer av plattnar. Den ena består av Cut-in risk index (CRI) och den andra är hur robotarna förhåller sig till den önskade tidskillnaden.

Vid starten av testerna hade robotarna samma avstånd till sammanvävningen och samma starthastighet. Värdet på CRI är då 1 eftersom roboten på rampen riskerar att köra in i sidan på den andra om ingen korrigering görs. Figur 5.13 visar förändringen av CRI vid ett av dessa tester. Desto längre algoritmen får verka desto lägre CRI-värde fås. Precis innan punkten för sammanvävningen svänger roboten som kommer från rampen in på huvudleden och tappar lite hastighet och därfor kan en liten uppgång i CRI observeras innan den andra roboten korrigeras för det nya avståndet mellan dem.



**Figur 5.13:** CRI för en sammanvävning i 0,75 m/s, punkten för sammanvävning nås av första roboten vid 5 s.

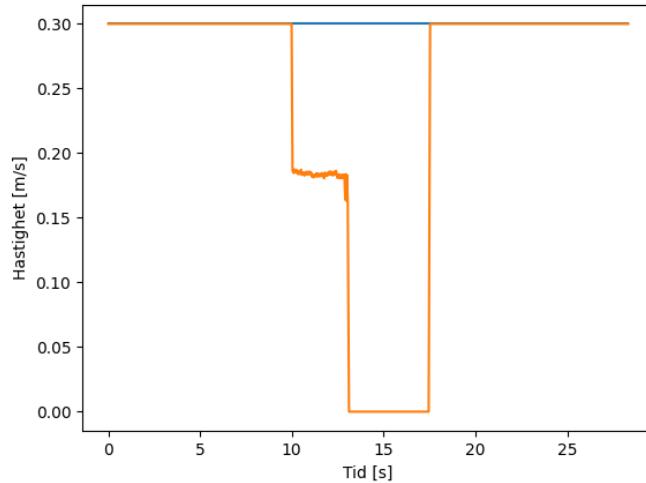
Figur 5.14 visar tidsskillnaden mellan robotarna vid punkten för sammanvävning. Inget av testerna som gjordes visade en tidsskillnad på mindre än det önskade värdet 3 s. En trend kan observeras som går mot 3s i högre hastigheter. Högsta hastigheten som användes var 0,75 m/s. Vid högre hastigheter tappade robotarna kontakten med GulliView och földe inte vägen som var programmerad för dem.



**Figur 5.14:** Tidsskillnaden för robotarna att nå punkten för sammanvävningen vid olika maxhastigheter.

### 5.2.2 Korsning

Figur 5.15 visar förändringen i hastigheterna för robotarna när de kör genom korsningen. Vid tio sekunder detekteras att båda robotarna är på väg mot korsningen den ena av dem saktar ner för att till slut stanna helt när den når detektionszonen som är runt korsningen vid 13 sekunder. Den andra robotten håller en konstant hastighet eftersom den fick prioritet av att den hade kortare tid kvar till korsningen. När robot två lämnat korsningens detektion skickas en ny hastighet ut till den som står still.



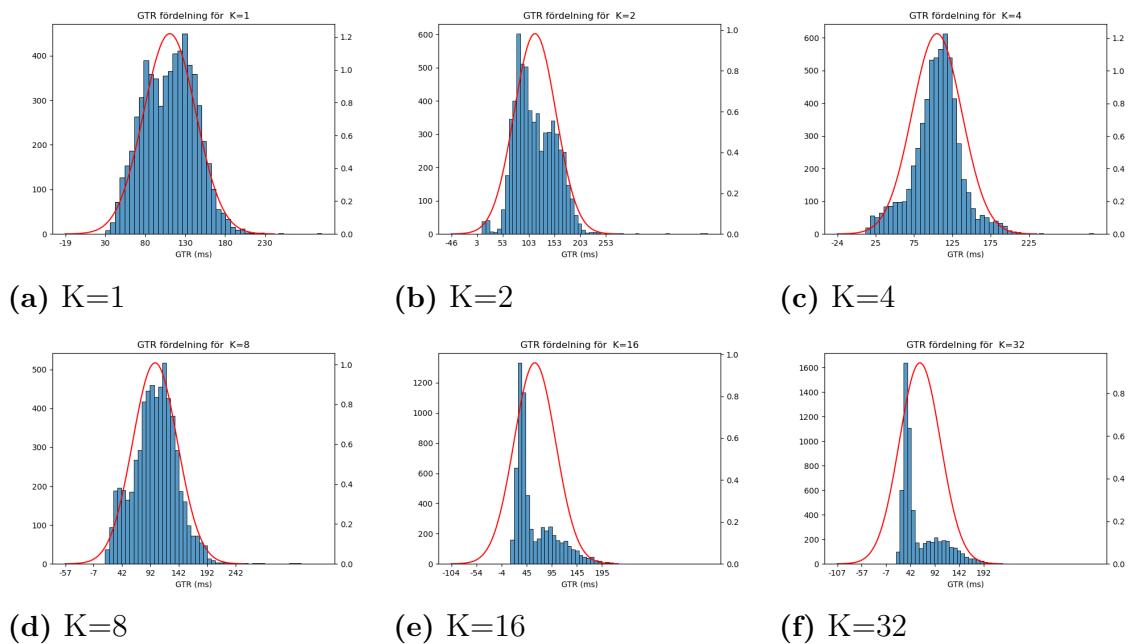
**Figur 5.15:** Hastigheter för robotarna vid ett möte vid korsningen. Orange saktar ner för att försöka släppa förbi den andra utan att stanna helt men stannar då den når korsningen innan den andra lämnat korsningen.

## 5.3 Medlemstjänst Shirzad Jakob

Detta sektion presenterar resultaten av Medlemstjänstens funktionalitet och prestanda. Medlemstjänstens programvara har visat betydande förbättringar tack vare både ny utrustning och särskilt den uppdaterade arkitekturen.

### 5.3.1 K-värdeseffekt på GTR Soltani Jakob

Resultatet av utvärderingen visar en direkt koppling mellan värdet av  $K$  och GTR. Vid ett  $K$ -värdet på 1,2 indikerar majoriteten av GTR-värdena över 100 ms (se figur 5.16a och 5.16b) vilket antyder att låga  $K$ -värden, såsom 1 och 2, inte är effektiva för GTR. Med  $K$ -värdena 4 och 8 observeras en viss förbättring i GTR, men ändå ligger de flesta GTR runt 100 ms (se figur 5.16c och 5.16d). Vid ökning av  $K$  till 16 och 32 noteras en minskning av GTR-värdet, där majoriteten av GTR är under 80 ms. Mellan  $K$ -värdena 16 och 32 är skillnaderna dock inte så stora, vilket antyder att  $K$ -värden över 16 inte nödvändigtvis resulterar i ytterligare förbättringar utan kan till och med försämra resultatet (se figur 5.16e och 5.16f).



**Figur 5.16:** GTR-värdefördelning och normalfördelning vid  $K$  och hastighet 0,5

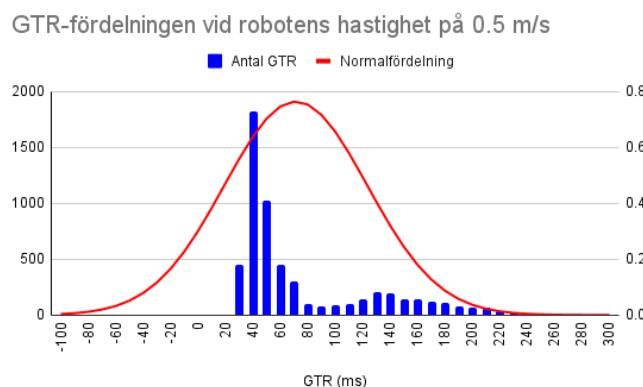
**Tabell 5.3:** K och medelvärde på GTR-värden

K	Medelvärde på GTR-värden (ms)
1	111
2	115
4	105
8	100
16	61
32	62

### 5.3.2 Hastighetens påverkan på GTR Shirzad Jakob

I detta avsnitt redovisas resultaten av testerna som genomfördes vid olika hastigheter.

- **Hastighet = 0,5 m/s:** GTR-fördelningsfiguren visar att de flesta GTR-värdena ligger under 76 ms, med majoriteten mellan 40 och 50 ms. Detta tyder på att vid hastigheten 0,5 m/s är det vanligaste GTR-värdet ungefär 40 ms. Som syns i figur 5.17 har normalfördelningskurvan en lång svans, vilket indikerar att GTR-värdet studsar vid vissa tillfällen. En möjlig orsak till detta kan vara att GulliView inte kan detektera robotarna i vissa positioner.

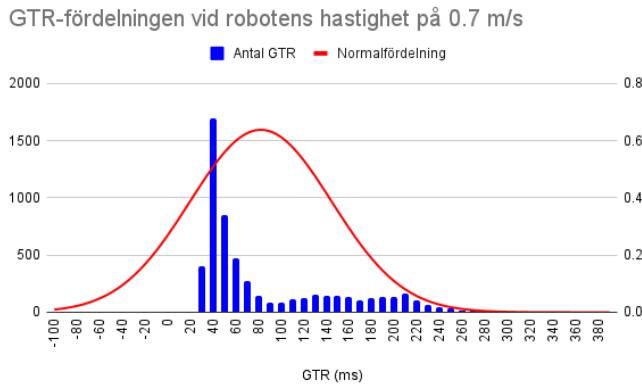


**Figur 5.17:** GTR-värdefördelning vid en hastighet på 0,5 m/s visar att de flesta värdena ligger mellan 40 och 60 ms, med en lång svans som antyder att GTR gör några studsar.

- **Hastighet = 0,7 m/s:** Vid denna hastighet visar GTR-fördelningsfiguren i figur 5.18 att flertalet GTR-värden ligger under 60 ms, med majoriteten runt 40 ms. Denna figur uppvisar också en lång svans, vilket antyder att GulliView misslyckas med att detektera robotarna även vid denna hastighet.

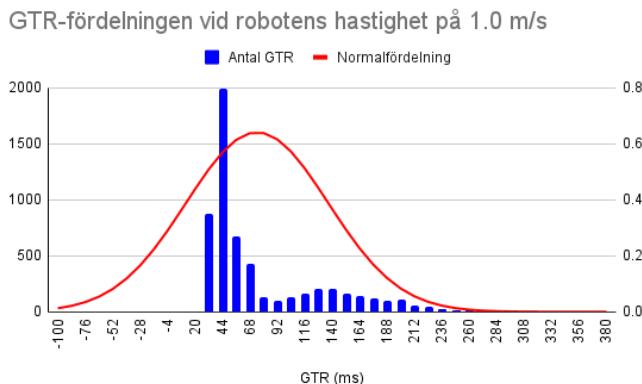
## 5. Resultat

---



**Figur 5.18:** GTR-värdefördelning vid en hastighet på 0,7 m/s visar att de flesta värdena ligger under 60 ms, med en lång svans som antyder att GTR gör några studsar.

- **Hastighet = 1,0 m/s:** I GTR-fördelningsfiguren i figur 5.19 visar det sig att huvuddelen av GTR-värdena ligger mellan 30 och 60 ms. En förlängd svans antyder emellertid att GTR ibland utför avvikeler eller studsar från förväntade beteenden, vilket även här bekräftar att GulliView ibland misslyckas med att detektera robotarna.



**Figur 5.19:** GTR-värdefördelning vid en hastighet på 1,0 m/s visar att de flesta värdena ligger mellan 30 och 60 ms, med en lång svans som antyder att GTR gör några studsar.

**Tabell 5.4:** Hastighet och medelvärde på GTR-värden

Hastighet (m/s)	Medelvärde på GTR-värden (ms)
0.5	71
0.7	82
1.0	74

# 6

## Diskussion

Detta kapitel för en diskussion av resultatet från tidigare kapitel.

### 6.1 GulliView Keivan Soltani

GulliViews mjukvara har haft flera förbättringsområden som resulterat i både snabbare exekveringstid och förbättrad funktionalitet. Trots framstegen finns det fortsatt utrymme och potential för flera förbättringar som kommer ta GulliViews mjukvara till nästa steg.

#### 6.1.1 Bildtransformation

Resultaten i sektion 5.1 är väldigt självförklarande. Figur 5.2 visar en god förbättring av den nya algoritmen på 4k upplösning. Det noteras att den nya algoritmens sämsta tider är lika långsamma som den gamla algoritmens bästa tider. Bättre framsteg noteras med Full HD upplösning där figur 5.1 visar att den nya algoritmens sämsta tider är flertal millisekunder snabbare än den gamla algoritmens bästa tider. Med hjälp av figur 5.3 kan det också fastställas att den genomsnittliga tiden för bågge upplösningar minskat.

Den nuvarande implementationen flyttade mappningen av labbmiljön från loopen till kalibreringsfasen. Detta gav en tydlig förbättring i prestanda (se figur 5.3), men ett eventuellt problem som kan uppstå med denna implementation är ifall kamerorna flyttas under programmets körtid. Vid en eventuell förflyttning av kamerorna betyder det att mappningen kommer att vara densamma medan området är an-norlunda, alltså kommer felaktiga värden att användas till mappningsalgoritmen. Detta är inget problem om GulliView endast är igång under kortare perioder eller i en kontrollerad labbmiljö. Men om GulliView körs under en lång tid där det finns risk att en kamera förflyttas så hade det behövts en implementation för att mappa om området regelbundet.

#### 6.1.2 Kamerläsning

Resultaten från den nya algoritmen gällande kamerans läshastighet är svårtolkad. Den nya algoritmen introducerar en väntmekanism som förklarades i 3.1.1. På grund av denna mekanism registreras det spikar varje tid som är jämt delbart med 5 vilket tydligt registreras i histogrammen, se högersida av figur 5.5 och figur 5.4,

ifall algoritmen inte tillåter en läsning. Jämförelsen mellan histogrammen i figur 5.4 visar att den nya algoritmen nädvändigtvis inte alltid är snabbare, detta kan bero på att väntmekanismen inte tillåter en läsning förrän den faktiskt kan utnyttja bilden den läser. Ännu tydligare är det i figur 5.6 som visar en högre genomsnittlig tid för den nya algoritmen på Full HD där en procentuell tidsökning på 25% noteras vilket betyder att denna implementation inte bör användas med Full HD upplösning då den strikt presterar sämre än den sekventiella implementationen.

Trots att ingen direkt förbättring på Full HD upplösning uppmäts, kan det ses en förbättring på 4k upplösningen. Det vänstra histogrammet i figur. 5.5 presenterar den gamla algoritmens snäva tidsintervall som visar stor majoritet mellan 21 och 23 ms. Det högra histogrammet presenterar ett bredare intervall för den nya algoritmen där stor majoritet fastställs från 0 till 20 ms. Trots det finns det många fall där den nya algoritmen är långsammare än den gamla då värden upp till 60 ms uppmäts, vilket är en betydande födröjning mot det tidigare högsta värdet 23 ms. Om inte systemet är känsligt för dessa eventuella födröjningar så ger denna implementation trots detta en nästan 44% snabbare exekveringstid i snitt i 4K upplösning (se figur 5.6).

En annan anledning som kan förklara det negativa resultatet i Full HD kontra det positiva i 4K är den nya tråden och buffern som används för att synkronisera de båda trådarna. För att buffern ska befina sig i ett korrekt tillstånd så krävs beräkningar vid varje förändring vilket tar tid. Då medelvärdet för inläsningstiden i Full HD endast var 7,18 ms och den genomsnittliga hastigheten för GulliView i Full HD var 54 Hz = 18,5 ms inklusive inläsningstiden, så är det möjligt att användandet av en separat tråd och buffer är för mycket overhead. Detta kan också vara varför 4K till skillnad från Full HD ger en betydande förbättring. Då tiderna för kamerläsning och GulliView i 4K är 21,78 ms respektive 14,66 Hz = 68,21 ms så är det möjligt att den extra synkroniseringstiden i detta fall är försumbar.

### 6.1.3 Taggsökning

Implementationen av Fast-search algoritmen tillsammans med Exhaustive-search algoritmen resulterade i att problemet som uppstod med enbart Fast-search är löst. Robotar kan nu åka mellan kameror där existerande robotar redan befinner sig och bli upptäckta. Detta var inte möjligt med enbart Fast-search algoritmen som tidigare blivit implementerat.

Prestandamässigt innebär det att varje 8:e bild medför en extra processtid som är beroende Exhaustive-search exekveringstid. Figur 5.8 och figur 5.7 skildrar tidsfördelningen med implementationen av Fast-search tillsammans med Exhaustive-search på respektive högersida och det är uppenbart att det Exhaustive-search står för den låga frekvensen i senare delen av intervallet. Den gamla algoritmen, se vänstersida av figur 5.8, visar ett väldigt litet intervall i kontrast till den nya, vilket innebär att den är väldigt konsekvent. Observera att den nya algoritmen hade varit fortsatt konsekvent om inte Exhaustive-search hade implementerats.

Fortsättningsvis ses det hur det breda intervallet påverkar den genomsnittliga tiden i figur 5.9. Viktigt att ha i åtanke här är att den gamla algoritmen endast består av Fast-search, medan den nya består av Fast-search och Exhaustive-search. Det innebär i sin tur att den genomsnittliga tiden kan vara missvisande genom att tro att det är en stor tidsskillnad vid varje sökning när det i själva verket endast blir längsammare varje 8:e bild då Exhaustive-search körs.

#### 6.1.4 Prestanda och Upplösning

I dagsläget har Full HD och 4K två olika roller när det kommer till vilken upplösning som GulliView ska använda. GulliViews prestanda med Full HD har förbättrats avsevärt från 14 Hz till 54 Hz och i 4K ökade den från 4 Hz till 14 Hz (se figur 5.12) vilket gör att fordonens positioner uppdateras mer frekvent och kan därför utföra mer komplexa manövrar i högre hastigheter. Den föregående versionen av GulliView hade ingen implementation för att hantera fordon som kör mellan kameror, vilket gör att prestandan som uppmäts är högre än den skulle vara med en fungerande sökalgoritm enligt figur 5.9. Den reela förbättringen av GulliView är förmodligen alltså ännu större än vad som uppmäts i resultatet.

För Full HD gäller det att den högre frekvensen gör att GulliView klarar av fordon i högre hastigheter för både 200 mm och 100 mm AprilTags vilket kan ses i tabellerna 5.1 och 5.2. Det kan dock noteras att GulliView kör i 54 Hz i Full HD, vilket är nära 60 Hz som är den högsta frekvensen GulliView kan köra med de 60 FPS kameror som används. Det betyder att trots Full HD nuvarande styrkor när det kommer till prestanda, så borde framtida förbättringar med denna upplösning förväntas vara begränsade.

Till skillnad från Full HD så har GulliView i 4K fördelen att mindre AprilTags kan användas. I detta projekt testades storlekar ner till 50 mm, men det är möjligt att det fungerar med ännu mindre taggar. Eftersom labbmiljön är begränsad till  $10 \times 5m$  så skulle mindre fordon göra att det var möjligt att köra flera stycken samtidigt på ett mer komplext vägnätverk. De mindre storlekarna av fordonen gör även att det kan bli billigare att införskaffa en större mängd, vilket gör att kooperativa manövrar kan simulerats mer verklighetstroget.

## 6.2 Manövrar **Kristofer Johannes**

I denna del diskuteras resultaten för manövrarna i 5.2.

### 6.2.1 CRI

CRI-värdena som uppmäts vid sammanväningen visar i figur 5.13 att algoritmen minskar risken för kollision men påvisar att det ändå finns en fortsatt hög risk när robotarna börjar dela samma körfält. Detta är en svaghet hos den modifierade CRI beräkningen då den endast tar avstånd mellan robotarna i beaktning. Det här gör

att en sammanvävning i låg hastighet där det är kortare avstånd mellan robotarna påvisar en högre risk än en sammanvävning i hög hastighet, där det egentligen kanske är mindre eller lika stor risk för kollision.

Anledningen till att CRI inte går ner mot noll är på grund av att i ekvation 4.7 syns det att det bästa värdet man kan få ut av den ekvationen är 1 då  $d_a$  är satt till 0. Detta ger ett CRI-värde på 0,37 vilket då är det absolut bästa värdet som kan fås med denna typ av uträkning. Att sätta  $d_a$  till 0 innebär en kollision mellan fordonen. I ursprungsfunktionen för CRI i [22] så används 'time to collision' (TTC) vilket fungerar som en typ av hävarm som kan observeras i ekvation 4.6. Ifall två av robotarna ligger nära varandra kan CRI-värdet vara lågt så länge TTC är stort. Eftersom beräkningen av TTC i ekvation 4.5 ska kunna användas är ett krav att bakomvarande fordon har högre hastighet än framförvarande vilket algoritmen åtgärdar snabbt för att det inte ska bli någon kollision. Den modifierade CRI-värdet säger inte så mycket om en verlig risk för kollision men visar att algoritmen minskar risken för kollision.

Som man kan se i figur 5.13 sker en minskad risk för kollision under en väldigt kort tid vilket som kan förklaras med att mätningen i distans sker till en punkt medan robotarna tillåts komma inom en viss radie av den punkten för att godkännas som förbikörd. Felmarginalen räknas alltså inte med i mätningen vilket leder till att om fejmarginalen till punkten är 30 cm, vilket är det mest förödande scenariot som kan uppstå, leder detta till ett fel i distansmätningen till sammanvävningspunkten på 60 cm. Det som händer efter sammanvävningspunkten, vid fem sekunder, är mer eller mindre oväsentligt då båda fordonen är i samma körfält och kör i samma hastighet med ett avstånd på tre sekunder.

### 6.2.2 Tidsskillnad mellan robotarna

Diagrammet som visas i figur 5.14 kan tidsskillnaden mellan robotarna observeras. En minskning av tiden mellan robotarna i korrelation till ökad hastighet upptäcktes. Detta beror på fejmätningen som kan uppstå vid sammanvävningspunkten, som förklaras i sektion 6.2.1. Resultatet blir på detta sätt på grund av att när robotarna färdas i lägre hastigheter blir ett hopp på 60cm mer betydande än vad det blir för robotarna i högre hastigheter och därmed mindre fel i CRI-värde. Eftersom ingen av de uppmätta säkerhetstiderna som är mellan robotarna, som visas i 5.14, överstiger 3 sekunder anses algoritmen uppfylla kravet för säkerhet.

Målet för robotarna är att hålla ett avstånd till varandra som både är säkert men också effektivt. Därför användes tresekundersregeln som är baserad på vad bilförare rekommenderas hålla i tid till framförvarande fordon i trafiken [19]. Att förhålla sig till tresekundersregeln anses vara ett bra avstånd på grund av många anledningar men det som skiljer från en människa kontra robotarna är reaktionstiderna. Systemet har mycket snabbare reaktionsförmåga än vad människor har vilket skulle troligtvis kunna innehålla ett minskat säkerhetsavstånd mellan fordonen. En analys av reaktionsförmågan på robotarna skulle vara lämplig för att öka genomströmningen men

fortfarande bibehålla en låg risk för kollision.

### 6.2.3 Korsningar

Korsningsalgoritmen utförs med säkerhet som visas i figur 5.15 eftersom endast en robot är i rörelse i korsningen. Som resultatdelen nämnde så har två typer av inbromsningskommandon implementerats. Första inbromsningen vill minska farten med så lite som möjligt för att undvika kollision. Den andra är ifall båda fordonen befinner sig inom korsningsområdet så ska den som har längst kvar till mittpunkten stanna och vänta tills den andra lämnat området. Däremot är det ineffektivt att stanna helt och vidare arbete skulle behövas så att robotarna slipper stanna helt.

I både algoritmen för sammanvävning och för korsningar förändras hastigheterna direkt utan att ta hänsyn till acceleration. De låga hastigheterna och att robotarna använder sig av elmotorer, som kan ändra sin hastighet mer eller mindre direkt, gör att det är rimligt att göra denna typ av hastighetsförändringar. Meddelanden skickas med 75 millisekunders mellanrum, vilket gör att även om den nya hastigheten inte uppnåtts så skulle algoritmen med hjälp av nästa meddelande räkna ut en ny hastighet för förhållandena i den nya situationen. För att bättre spegla verkligheten skulle en spärr kunna införas för att max tillåta en viss acceleration.

## 6.3 Medlemstjänst Soltani Keivan

Studien fokuserade på att förbättra Medlemstjänsten genom två huvudfaser av utveckling. Den första fasen innebar integrationen av två olika repositories, som tidigare tagits upp, för att anpassa Medlemstjänsten till den senaste versionen av GulliView. Detta var avgörande eftersom den nya metoden för att hitta AprilTags i GulliView var betydligt snabbare än den tidigare metoden som användes av Medlemstjänsten. Den andra fasen involverade förändringar i medlemstjänstens arkitektur för att förbättra dess prestanda och funktionalitet. Genom olika steg och metoder, såsom skapande av ny arkitektur, sammanhangande meddelande via en centraliserad avsändare och optimering av kommunikationen genom övergång från nätverkssändning till delat minne, uppnåddes detta.

För att utvärdera Medlemstjänstens funktionalitet och prestanda genomfördes en serie tester med varierande parametrar. Dessa tester inkluderade att köra robotarna i en oändlig figur för att simulera olika scenarier och bedöma Medlemstjänstens förmåga att hantera kommunikationen vid olika hastigheter och med olika K-värden.

Enligt data från Tabell 5.3 framgår det att det längsta medelvärdet för GTR är 61 ms, vilket uppnås vid  $K = 16$ . Vid en ökning av  $K$  till 32 ökar medelvärdet för GTR till 62 ms. Denna ökning antyder en överanpassning och tyder på att ytterligare ökning av K-värdet inte nödvändigtvis kommer att förbättra GTR, men kan istället resultera i att nya AprilTags inte detekteras. Vidare framkom att K-värdet hade en stor effekt på GTR, och det bästa K-värdet var 16, medan hastigheten också påverkade GTR men inte i samma utsträckning som K-värdet.

Ytterligare visar Tabell 5.4 att hastigheten påverkar GTR-värdet. Generellt sett ökar GTR-värdena med ökande hastighet, men förändringen är marginell. Om vi betraktar medelvärdet på GTR-värdena enligt tabell 5.4, noterar vi att vid hastigheten 0,7 m/s ökar medelvärdet mer än det gör vid 1,0 m/s. Anledningen till denna avvikelse är oklar; trots flera testkörningar framkom samma resultat. Å andra sidan resulterar en fördubbling av hastigheten från 0,5 m/s till 1,0 m/s endast i en marginell ökning av GTR-värdet med endast 3 ms.

Resultaten av experimenten indikerade primärt att förändringar i arkitekturen för medlemstjänsten, särskilt beträffande kommunikationsmetoderna mellan dess noder, resulterade i märkbara förbättringar av prestanda. Genom att migrera från en bredsändnings strategi via UDP-sockets till IPC mekanismer som delat minne kunde kommunikationslatensen reduceras och systemets effektivitet förbättras betydligt. Detta berodde på att behovet av att inaktivera CPU för att synkronisera olika noder elimineras. Resultaten av testerna visade också hur olika faktorer som hastighet och K-värde påverkade Medlemstjänstens förmåga att hantera kommunikationen mellan olika noder och beräkna GTR.

Resultaten visade tydligt hur dessa faktorer påverkade Medlemstjänstens prestanda och funktionalitet. Dessutom visade resultaten en signifikant förbättring av den nya Medlemstjänsten jämfört med den tidigare versionen. Till exempel var snittet av GTR för den tidigare Medlemstjänsten för olika hastigheter från 0,25 m/s till 1 m/s mer än 400 ms [1], medan det för den nya Medlemstjänsten för hastigheterna 0,5, 0,7 och 1 m/s var 75 ms, vilket visar en femfaldig förbättring. Dessa förbättringar är delvis på grund av det nya kamerasystemet, men i hög grad tack vare den nya arkitekturen. I den nya arkitekturen behöver Medlemstjänsten aldrig inaktivera processorn för att kunna synkronisera olika noder. Istället hanteras allt samtidigt utan att behöva vänta på processorer, vilket är möjligt tack vare den centraliserade avsändaren och IPC-kommunikationen mellan master och GulliView.

Sammanfattningsvis har studien framgångsrikt bidragit till förståelsen och förbättringen av Medlemstjänstens prestanda genom en kombination av noggrant genomförda tester och analys av resultaten. Genom att adressera utmaningar i systemets arkitektur och utvärdera dess prestanda under olika förhållanden har vi ökat kunskapen om hur man optimerar och effektivisera liknande distribuerade system.

# 7

## Slutsats **Keivan Johannes**

Arbetet som har genomförts detta projekt har bidragit med god framgång inom samtliga arbetsområden. Det betyder att framtida grupper som tar detta arbete vidare kommer ha goda förutsättningar att producera bra resultat.

### 7.1 Framtida arbeten **Keivan Kristofer**

Här presenteras ett antal förslag till vidare arbeten inom de olika processerna som detta projekt använt sig av.

#### 7.1.1 GulliView **Keivan Kristofer**

GulliView innehåller flera aspekter där ens kreativitet tillåter en att lösa dessa problem med flera olika tillvägagångssätt. Ett tydligt exempel på detta är lösningen för Fast-search problemet som efter detta arbete samarbetar med Exhaustive-search för att låta fordonen röra sig mellan kameror. Detta är bara en av flera lösningar. Andra intressanta metoder som tillåter förbättrad exekveringstid av algoritmen är exempelvis implementationen av en algoritm som endast kollar utkanten av bilderna för att hitta nya robotar som passerar in i kameran. Det innebär att sökområdet minimeras ordentligt i förhållande till Exhaustive-searchs sökområde samtidigt som funktionaliteten fungerar med det förväntade beteendet. Ett steg längre med denna implementation är att tillåta kamerorna att kommunicera med varandra, och endast när en robot befinner sig vid utkanten ska kameran i fråga meddela detta till kameran bredvid. Endast då kommer den kameran att söka i utkanten vid eventuell passering in i kamerans område.

Ett annat område som kan undersökas är kamerornas placering och kvantitet. I dagsläget är testytan fördelad över fyra kameror, men en omplacering i form av vertikal förflyttning av kamerorna hade inneburit att testytan fördelas över fler kameror, exempelvis åtta stycken. Fördelen med detta är bland annat att mindre AprilTags kan användas i högre hastigheter vilket med mjukvaran idag är begränsat. Nackdelen med detta blir att fler processor behövs köras, och därav krävs det mer datorkraft, vilket enkelt blir en begränsning.

#### 7.1.2 Manövrar **Jakob Kristofer**

Det finns stora möjligheter för vidareutveckling av olika manövrar, både vidareutveckling av de befintliga där det blir mer sofistikerade hanteringar på grund av en

snabbare uppdatering av deras positioner men också nyare mer avancerade då mer precisa kommandon skulle kunna ges.

Manövrarna utförs just nu med hjälp av att en server separerad från Gulliview och robotarna. Genom att utnyttja medlemstjänsten där alla robotar är medvetna om varandra skulle distribuerade manövrar kunna utvecklas där robotarna istället skickar meddelanden till varandra för att ta beslut när kritiska situationer uppstår som föreslaget i [20].

### 7.1.3 Medlemstjänst Shirzad Kristofer

Ett specifikt område som förtjänar särskild uppmärksamhet är beräkningen av GTR. Just nu finns det en medvetenhet om att dessa beräkningar inte är helt korrekta eller optimala. Genom att rikta in sig på att beräkna GTR individuellt för varje robot öppnas dörrar till en mer anpassad och effektiv styrning av hastigheter under korsningar och sammanvävningar. Ytterligare, att utforska användningen av GTR-värden för att styra uttömmande sökningar istället för att använda fasta K-värden är en intressant tankeväckande strategi som kan öppna upp för en mer adaptiv och intelligent hantering av olika situationer. Istället för att förlita sig på ett förutbestämd K-värde kan systemet dynamiskt anpassa sökalgoritmerna baserat på reeltidsdata om GTR.

Exempelvis, om GTR plötsligt ökar till 100 ms så kan GulliView-systemet automatiskt växla till en mer resurskrävande sökalgoritm för att säkerställa att den upptäcker och hanterar hinder eller situationer på ett mer effektivt sätt. Denna anpassningsförmåga gör att systemet kan vara mer flexibelt och anpassa sig till förändrade förhållanden i realtid, vilket i sin tur kan leda till en förbättrad prestanda och användarupplevelse.

Genom att integrera sådana adaptiva strategier kan Medlemstjänsten utvecklas till att vara mer proaktiv och intelligent i sin hantering av olika scenarier, vilket är avgörande för att möta kraven och utmaningarna i en dynamisk och föränderlig miljö. Detta öppnar upp för nya möjligheter och innovationer inom området distribuerade system och autonoma system.

## 7.2 Sammanfattning

Sammanfattningsvis har detta arbete uppnått framsteg inom samtliga tre områden: GulliView, manövar och medlemstjänst. Betydande förbättringar har gjorts, inklusive en snabbare och mer exakt GulliView, en effektivare och mer responsiv medlemstjänst, samt en algoritm för manövrering som kan hantera olika trafiksituации med högre hastighet och säkerhet än tidigare.

GulliView-systemet ger nu mer pålitlig och snabb lokaliseringssdata, vilket är avgörande för att robotarna ska kunna navigera korrekt och säkert i sin omgivning. Medlemstjänsten har förbättrats så den säkerställer en snabbare och mer tillförlitlig

kommunikation mellan Wifibotarna och GulliView.

Dessa tekniska framsteg för projektet öppnar upp nya möjligheter för framtida utveckling. Framtida studenter som kommer att arbeta med detta projekt har en solid grund att bygga vidare på. Med dessa förbättringar kommer ett ännu snabbare och effektivare system utvecklas.

# Litteraturförteckning

- [1] A. Enliden, D. Ferm, V. Hui, M. Johansson, A. Malla, and V. Olsson, “Förbättringar av lokaliseringssystem för nedskalade fordon,” 2023, bachelor’s thesis, Computer Science and Engineering, Chalmers University of Technology.
- [2] R. Azimi, G. Bhatia, R. Rajkumar, and P. Mudalige, “V2v-intersection management at roundabouts,” 2013-04-08. [Online]. Available: <https://www.sae.org/publications/technical-papers/content/2013-01-0722/>
- [3] *Tesla Model X Owners Manual*, 2024-04-02, (Senast besökt 20 september 2024). [Online]. Available: [https://www.tesla.com/ownersmanual/modelx/us/Owners\\_Manual.pdf](https://www.tesla.com/ownersmanual/modelx/us/Owners_Manual.pdf)
- [4] “Automated guided vehicles: Top 5 main things to know,” 2019-09-13, (Senast besökt: 20 september 2024). [Online]. Available: <https://www.flexqube.com/en-gb/news/automated-guided-vehicle-systems-5-things-to-know-before-introducing-agvs-to-your-shop-floor/>
- [5] M. Pahlavan, M. Papatriantafilou, and E. M. Schiller, “Gulliver: a test-bed for developing, demonstrating and prototyping vehicular systems,” in *Proceedings of the 9th ACM International Symposium on Mobility Management and Wireless Access*, ser. MobiWac ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1–8. [Online]. Available: <https://doi.org/10.1145/2069131.2069133>
- [6] S. Dolev, T. Petig, and E. M. Schiller, “Self-stabilizing and private distributed shared atomic memory in seldomly fair message passing networks,” *Algorithmica*, vol. 85, no. 1, pp. 216–276, 2023.
- [7] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, “Renaissance: A self-stabilizing distributed SDN control plane using in-band communications,” *J. Comput. Syst. Sci.*, vol. 127, pp. 91–121, 2022.
- [8] Z. Georgiou, C. Georgiou, G. Pallis, E. M. Schiller, and D. Trihinas, “A self-stabilizing control plane for fog ecosystems,” in *UCC*. IEEE, 2020, pp. 13–22.
- [9] A. Casimiro, E. Ekenstedt, and E. M. Schiller, “Self-stabilizing manoeuvre negotiation: The case of virtual traffic lights,” in *SRDS*. IEEE, 2019, pp. 354–356.
- [10] S. Dolev, C. Georgiou, I. Marcoulis, and E. M. Schiller, “Practically-self-stabilizing virtual synchrony,” *J. Comput. Syst. Sci.*, vol. 96, pp. 50–73, 2018.
- [11] S. Dolev and E. Schiller, “Communication adaptive self-stabilizing group membership service,” *IEEE Trans. Parallel Distributed Syst.*, vol. 14, no. 7, pp. 709–720, 2003.
- [12] ———, “Self-stabilizing group communication in directed networks,” *Acta Informatica*, vol. 40, no. 9, pp. 609–636, 2004.

- [13] S. Dolev, E. Schiller, and J. L. Welch, “Random walk for self-stabilizing group communication in ad hoc networks,” *IEEE Trans. Mob. Comput.*, vol. 5, no. 7, pp. 893–905, 2006.
- [14] M. Moneim, I. Reza, S. M. Rahman, S. M. H. Abdullah, A. Jamal, and F. S. Al-Ismail, “Impacts of autonomous vehicles on greenhouse gas emissions—positive or negative?” *International Journal of Environmental Research and Public Health*, 2021. [Online]. Available: <https://doi.org/10.3390/ijerph18115567>
- [15] P. Lin. (2015) Why ethics matters for autonomous cars. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-3-662-45854-9\\_4](https://link.springer.com/chapter/10.1007/978-3-662-45854-9_4)
- [16] Ericsson. (2017) The self-driving future. [Online]. Available: <https://www.ericsson.com/4ac611/assets/local/reports-papers/consumerlab/reports/2017/ericsson-consumerlab-driving-report.pdf>
- [17] E. Andreotti, Selpi, and M. Aramrattana, “Cooperative merging strategy between connected autonomous vehicles in mixed traffic,” *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 825–837, 2022.
- [18] Z. Spasov and A. Madevska Bogdanova, “Inter-process communication, analysis, guidelines and its impact on computer security.” Institute of Informatics, Faculty of Natural Sciences and Mathematics, Ss . . . , 2010.
- [19] “Tresekundersregeln,” 2024, (Senast besökt: 20 september 2024). [Online]. Available: <https://trafiko.se/faktabank/ordlista/tresekundersregeln>
- [20] V. Savic, E. M. Schiller, and M. Papatriantafilou, “Distributed algorithm for collision avoidance at road intersections in the presence of communication failures,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1005–1012.
- [21] L. Lamport, “The mutual exclusion problem: part i—a theory of interprocess communication,” in *Concurrency: the Works of Leslie Lamport*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 227–245. [Online]. Available: <https://doi.org/10.1145/3335772.3335937>
- [22] M. Aramrattana, T. Larsson, C. Englund, J. Jansson, and A. Nåbo, “A novel risk indicator for cut-in situations,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6.