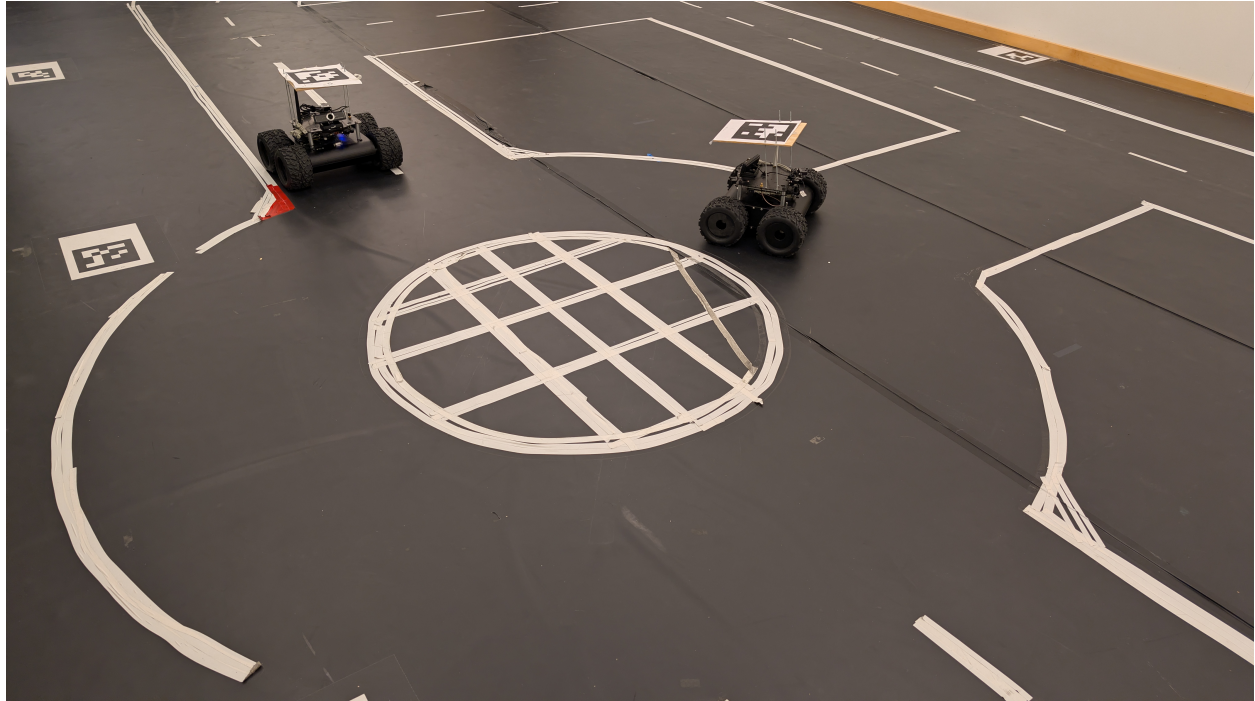




Implementation och evaluering av kooperativa manövrer för autonoma robotar

Kandidatarbete inom data- och informationsteknik

NOUR ALDEEN DABORA
JAKOB GREVLE
ALBIN JOHANSSON
ELIAS NILSSON
ANDERS SKOGLUND
TONY TRAN

KANDIDATARBETE 2026

Implementation och evaluering av kooperativa manövrer för autonoma robotar

NOUR ALDEEN DABORA

JAKOB GREVLE

ALBIN JOHANSSON

ELIAS NILSSON

ANDERS SKOGLUND

TONY TRAN



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2026

Implementation och evaluering av kooperativa manövrer för autonoma robotar

NOUR ALDEEN DABORA JAKOB GREVLE ALBIN JOHANSSON ELIAS
NILSSON ANDERS SKOGLUND TONY TRAN

© NOUR ALDEEN DABORA, JAKOB GREVLE, ALBIN JOHANSSON,
ELIAS NILSSON, ANDERS SKOGLUND, TONY TRAN 2026.

Handledare: Elad Schiller, institutionen för data- och informationsteknik

Examinator: Arne Linde, institutionen för data- och informationsteknik

Kandidatarbete 2026

Institutionen för data- och informationsteknik

Chalmers tekniska högskola och Göteborgs universitet

SE-412 96 Göteborg

Telefon +46 31 772 1000

Omslag: Bild på två WiFiBotar som kör på testbädden i laboratoriet.

Sätts i L^AT_EX

Göteborg, Sverige 2026

Implementation och evaluering av kooperativa manövrer för autonoma robotar

NOUR ALDEEN DABORA, JAKOB GREVLE, ALBIN JOHANSSON,
ELIAS NILSSON, ANDERS SKOGLUND, TONY TRAN

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

Abstract

This thesis presents the design, implementation, and evaluation of a unified software architecture for cooperative autonomous driving using WiFiBot robots within a controlled indoor environment in a laboratory at Chalmers University of Technology. The project focuses on three cooperative maneuvers: platooning, intersection coordination and roundabout coordination. In addition, an emergency braking system is implemented to ensure collision avoidance across all maneuvers.

The system combines distributed communication, shared localization through the GulliView positioning system, and modular maneuver-specific control algorithms implemented using ROS. Platooning is implemented using Cooperative Adaptive Cruise Control (CACC), intersection coordination is handled through Virtual Traffic Light (VTL) and reservation-based coordination, and roundabout coordination is achieved through distributed gap-acceptance logic inspired by real traffic behavior. All maneuvers share a common segment-based path representation and communication structure.

The implemented system was evaluated experimentally using two robots in the laboratory evaluation environment. Results showed that the emergency braking system prevented collisions at speeds up to 0.8 m/s. Platooning achieved stable convergence within 20.4 s at 0.3 m/s and 12.5 s at 0.5 m/s. The intersection coordination handled speeds up to 1.1 m/s for reservation-based coordination and 0.6 m/s for VTL coordination. The roundabout coordination handled speeds up to 0.6 m/s.

The experiments demonstrate that the proposed system fulfills the defined safety requirements while enabling cooperative maneuvers to complete in finite time. The project contributes a reusable and extensible software architecture intended to support future student projects in the laboratory.

Keywords: autonomous, robots, cooperative, maneuvers, platooning, intersection, roundabout

Sammandrag

Detta kandidatarbete presenterar design, implementation och evaluering av en enhetlig mjukvaruarkitektur för kooperativ autonom körning med WiFiBot-robotar i en kontrollerad inomhusmiljö vid ett laboratorium vid Chalmers tekniska högskola. Projektet fokuserar på tre kooperativa manövrer: kolonnkörning, korsningskoordinering och rondellkoordinering. Utöver detta implementeras ett nödstopp för att säkerställa att kollisioner inte uppstår under samtliga manövrer.

Systemet kombinerar distribuerad kommunikation, gemensam lokalisering genom lokaliseringssystemet GulliView samt modulära styralgoritmer för respektive manöver implementerade med hjälp av ROS. Kolonnkörning implementeras med hjälp av Cooperative Adaptive Cruise Control (CACC), korsningskoordinering hanteras genom Virtual Traffic Light (VTL) och reservationsbaserad koordinering, medan rondellkoordinering implementeras genom ett distribuerat system för att bestämma om en tillräckligt stor tidslucka finns, inspirerat av verkligt trafikbeteende. Alla manövrer använder en gemensam segmentbaserad ruttmodell och kommunikationsstruktur.

Det implementerade systemet evaluerades experimentellt med två robotar i laboratoriets evalueringsmiljö. Resultaten visade att nödstoppet förhindrade kollisioner vid hastigheter upp till 0,8 m/s. Kolonnkörning uppnådde stabil konvergens inom 20,4 s vid 0,3 m/s och 12,5 s vid 0,5 m/s. Korsningskoordineringen hanterade hastigheter upp till 0,6 m/s med VTL-koordinering och upp till 1,1 m/s med reservationsbaserad koordinering. Rondellkoordineringen hanterade hastigheter upp till 0,6 m/s.

Experimenten visar att det föreslagna systemet uppfyller de definierade säkerhetskraven samtidigt som de kooperativa manövrerna kan genomföras inom ändlig tid. Projektet bidrar med en återanvändbar och utbyggbar mjukvaruarkitektur avsedd att stödja framtida kandidatarbeten inom laboratoriet.

Tackord

Vi vill först och främst tacka vår handledare Elad Schiller för hans värdefulla vägledning och stöd under projektets gång. Hans engagemang har varit till stor hjälp för både planering och genomförande av arbetet.

Ett varmt tack riktar vi även till vår TA Hasan Zahid för hans tillgänglighet och hjälpsamma råd.

Slutligen vill vi tacka doktoranden Torbjörn Livén för hans konstruktiva feedback på vår rapport, vilket bidrog till en ökad kvalitet på vårt arbete.

Nour Aldeen Dabora, Jakob Grevle, Albin Johansson, Elias Nilsson,
Anders Skoglund, Tony Tran, Göteborg, juni 2026

Innehåll

Figurer	xi
Tabeller	xiii
1 Inledning	1
1.1 Uppdragsbeskrivning	1
1.2 Relaterat arbete	2
1.3 Vårt bidrag	2
2 Bakgrund	5
2.1 Evalueringsmiljö	5
2.2 WiFiBot	6
2.3 AprilTags	6
2.4 GulliView	6
2.5 UDP	7
2.6 ROS	7
2.7 Kolonnkörning	7
3 Metod	9
3.1 Nödstopp	9
3.1.1 Krav	9
3.1.2 Vald metod och modell	10
3.1.3 Argument för kravuppfyllelse	11
3.2 Kolonnkörning	11
3.2.1 Krav	11
3.2.2 Vald metod och modell	11
3.2.3 Beslutslogik	12
3.2.4 Identifiering av ledare	13
3.2.5 Argument för kravuppfyllelse	13
3.3 Korsningskoordinering	14
3.3.1 Krav	14
3.3.2 Vald metod och modell	14
3.3.3 Representation av korsningar och zoner	15
3.3.4 Beslutslogik	16
3.3.5 Argument för kravuppfyllelse	19

3.4	Rondellkoordinering	19
3.4.1	Krav	19
3.4.2	Vald metod och modell	20
3.4.3	Beslutslogik	21
3.4.4	Argument för kravuppfyllelse	23
4	Enhetlig mjukvaruarkitektur och implementation	25
4.1	Översikt över systemet	25
4.2	Ruttmodell	27
4.3	Kolonnkörning	28
4.3.1	Mjukvaruarkitektur	28
4.3.2	Rollbestämning i uppdateringscykeln	28
4.3.3	Identifiering av ledare	29
4.3.4	Implementation av CACC-regulatorn	29
4.4	Korsningskoordinering	29
4.4.1	Mjukvaruarkitektur	29
4.4.2	Uppdateringsflöde och lokalt korsningstillstånd	30
4.4.3	Medlemskapsbild	31
4.4.4	Applicering av signal	31
4.4.5	Implementation av VTL-logik	31
4.4.6	Implementering av reservationslogik	32
4.5	Rondellkoordinering	33
4.5.1	Mjukvaruarkitektur	33
4.5.2	Beslutsflöde och inkörningslogik	33
5	Evalueringsplan	35
5.1	Forskningsfrågor	35
5.2	Evalueringsmiljö	35
5.3	Testfall	35
5.4	Evalueringskriterier	38
5.5	Experiment	38
6	Resultat och diskussion	43
6.1	Nödstopp	43
6.2	Kolonnkörning	44
6.3	Korsningskoordinering	46
6.4	Rondellkoordinering	49
7	Slutsats	53
7.1	Framtida arbete	53
	Litteratur	55
A	Segmentkarta för testbädden	I
B	Källkod - Bitbucket	III

Figurer

2.1	Testbädden inom evalueringsmiljön	5
2.2	WifiBot med AprilTag-ID 4. Försedd med en kamera, infraröda sensorer och en unik AprilTag	6
3.1	Geometrisk beskrivning av nödstoppets utvärderingsmodell. Figuren visar avståndet fram till den andra roboten (d_{fram}), det laterala avståndet (d_{sid}), samt det kortaste fria avståndet (gap) i relation till den definierade nödstoppzonen.	10
3.2	Visualisering av tillåtna rörelser i korsning 1.	15
3.3	Zonindelning för de tre korsningarna i evalueringsmiljön. Grönt markerar infartszoner, rött markerar utfartszoner, gult markerar konfliktzoner och orange markerar den gemensamma in- och utfartszonen i korsning 3.	16
3.4	Placering av koordinater och vinklar	23
4.1	Uppdelning och programflöde. (Egen figur, genererad med Mermaid (mermaid.ai) från egen kod.)	26
4.2	Illustration av hur testbäddens körbara delar representeras i systemet.	28
5.1	Testfall 1: Robot R körs mot ett stationärt hinder och förväntas stanna innan kollision.	36
5.2	Testfall 2: Följaren F ska anpassa sin hastighet för att hålla det önskade avståndet g_{des} till den framförvarande roboten L. g_0 är initialavståndet.	36
5.3	Testfall 3 för korsningspassage. I (a) korsning 2 kör Robot 1 från öst mot norr och Robot 2 från syd mot norr. I (b) korsning 3 kör Robot 1 från öst mot norr och Robot 2 från syd mot öst.	37
5.4	Testfall 4: Två robotar anländer till rondellen samtidigt och förväntas passera utan kollision. Robotarnas initiala avstånd till rondellens infart betecknas med d_1 för R1 (1,0 m) och d_2 för R2 (2,15 m) och används för att sätta upp testfallet.	37
6.1	Figuren visar hastigheten för robotarna och avståndet till hindret vid de olika försöken. De mörka linjerna är ett medelvärde av de fem olika försök som syns i bakgrunden.	44

6.2	Avståndsfelet $g_{\text{err}}(t)$ från experiment 2 för fyra ledarhastigheter. Tjocka linjer är medelvärde, tunna linjer är enskilda körningar. Det gröna bandet markerar hypotesens ± 10 cm. Prickad vertikal linje är medianen av konvergenstiden t_c	45
6.3	Hastigheter från experiment 3. Svart linje är ledarens hastighet och färgad linje är följarens. Tjocka linjer är medelvärde, tunna linjer är enskilda körningar. Vertikal streckad linje markerar tidpunkten då ledarens hastighet ändrades. Rutan i övre högra hörnet visar medianen av strängstabilitetskvoten $\hat{v}_{\text{följare}}/\hat{v}_{\text{ledare}}$	46
6.4	Genomsnittligt minsta avstånd mellan Robot 1 och Robot 2 för olika hastigheter i korsning 2 och korsning 3.	48
6.5	Genomsnittlig passagetid för Robot 1 och Robot 2, för olika hastigheter i korsning 2 respektive korsning 3.	48
6.6	Resultat av rondellprestanda för experiment 6. I stapeldiagrammet representerar blått Robot 1 och orange Robot 2.	50
6.7	Resultat av rondellsäkerheten för experiment 6 vid olika hastigheter. Grafen illustrerar distansen över tid mellan de två robotarna. De skarpa linjerna visar medeldistansen av de fem testerna som gjordes i den hastigheten, medan de skuggade linjerna representerar varje enskilt test i den hastigheten.	51
A.1	Översikt över testbäddens segment. Figuren visar segmentnamn, körbanor och kopplingar mellan olika delar av testbädden.	I
B.1	Översikt över Bitbucket-repositoriets olika delar samt läsrekommendationer.	III

Tabeller

6.1	Kollisionsstatus vid nödstopp över fem försök per hastighet.	43
6.2	Kollisionsstatus för korsning 2 och korsning 3 över försöken per initialhastighet.	47
6.3	Kollisionsstatus vid rondellmanövern över fem försök per initialhastighet.	49

1

Inledning

Vid Chalmers tekniska högskola har det under flera år bedrivits studentprojekt i ett laboratorium där kooperativa manövrer för autonoma robotar studeras. Detta projekt syftar till att implementera ett enhetligt system för autonomt körande i laboratoriet. Projektet innefattar tre trafikscenarier: kolonnkörning, rondellkoordinering och korsningskoordinering. Kolonnkörning innebär att två eller flera robotar kör i en konvoj med ett litet avstånd. Rondellkoordinering innebär att koordinera två eller flera robotar genom en rondell utan att det uppstår risk för kollision. Korsningskoordinering förklaras på samma sätt fast robotar ska istället koordineras genom en korsning. För att kunna evaluera dessa manövrer utan risk för skador på robotar behövs även ett nödstopp. Det har till uppgift att omedelbart stanna roboten om en kollision är på väg att inträffa. Projektet implementerar dessa manövrer eftersom de är de vanligast förekommande manövrerna inom en trafikmiljö.

1.1 Uppdragsbeskrivning

Detta projekt fokuserar huvudsakligen på säkerheten för varje manöver. De presenterade lösningarna kan visa sig vara ineffektiva i vissa scenarier men anses vara lyckade så länge säkerheten är garanterad. Projektets implementation bygger på att trafikscenarierna utförs i en kontrollerad evalueringsmiljö, fri från oförutsägbara faror som fotgängare eller andra objekt som plötsligt kan blockera vägbanan. Eftersom varje trafikscenario är olika i sin natur definieras det krav i kapitel 3 för varje manöver. Kraven säkerställer ett säkert genomförande och definierar när manövern anses vara fullföljd. För att visa att kraven är uppfyllda för varje manöver ställs två forskningsfrågor.

F-1: I vilken utsträckning uppfyller de implementerade manövrerna säkerhetskraven i respektive trafikscenario?

F-2: I vilken utsträckning uppfyller de implementerade manövrerna kraven om framkomlighet i respektive trafikscenario?

Forskningsfrågorna besvaras genom experiment i laboratoriets testbädd, vilket

innebär att resultaten endast är giltiga för den studerade miljön och de förutsättningar som råder där. Resultaten kan därför inte utan vidare generaliseras till verkliga trafikmiljöer, där ytterligare faktorer såsom väderförhållanden, varierande vägunderlag och oförutsägbara trafikanter påverkar systemets prestanda. Eftersom de utvecklade systemen är avsedda för användning i en kontrollerad laboratoriemiljö bedöms projektets etiska och samhällsliga konsekvenser vara begränsade. Projektets påverkan är i huvudsak avgränsad till laboratoriets verksamhet och syftar främst till att stödja fortsatt forskning och utveckling inom området.

1.2 Relaterat arbete

Detta projekt bygger vidare på två tidigare kandidatarbeten som genomförts inom samma laboratorium. Kandidatarbetet av Malmentun et al. [1] vidareutvecklade laboratoriets system för inomhuslokalisering och kooperativa manövrer genom förbättringar av GulliView och implementering av distribuerade kooperativa manövrer. Kandidatarbetet av Asp et al. [2] utvecklade styrsystem för autonoma skalade fordon och lade grunden för senare projekt inom laboratoriets testbädd. I detta projekt används motorstyrningen samt kommunikationslösningen från [2]. Strukturen på systemet från de kooperativa manövrerna av [1] används som grund för projektets implementation.

Lösningen för kolonnkörning baseras på ett tidigare arbete av Ploeg et al. [3]. Ploeg et al. undersöker design och experimentell utvärdering av Cooperative Adaptive Cruise Control (CACC), där fordon samarbetar genom kommunikation för att förbättra stabilitet och trafikflöde. Trafiksimulatorn SUMO innehåller en öppen implementation av Ploegs CACC-modell [4], som studeras som referens vid implementation av kolonnkörning i detta projekt.

Flera tidigare arbeten används som inspiration för implementationen av korsningskoordinering. Casimiro et al. [5] används främst som grund för Virtual Traffic Light (VTL), där rörelser med korsande färdvägar inte ges inträde samtidigt. Savic et al. [6] används som inspiration för arbetets behandling av distribuerad kollisionsundvikning i korsningar, särskilt genom principen att dela upp korsningsområdet i delområden och hantera osäker information konservativt.

För rondellkoordinering hämtas inspiration från studien av Fakirah et al. [7]. Fakirah et al. presenterar ett centraliserat trafikstyrningssystem för autonoma fordon i flerfiliga cirkulationsplatser baserat på kommunikation mellan fordon och väginfrastruktur.

1.3 Vårt bidrag

Detta projekt behandlar ett problem inom laboratoriet för autonoma och kooperativa fordon vid Chalmers. Genom att standardisera grundläggande

kooperativa manövrer och samla dem i ett enhetligt system tillsammans med lösningar för kommunikation och styrning kan framtida kandidatarbeten återanvända samma arkitektur. Detta förenklar vidare utveckling inom laboratoriet och möjliggör att framtida projekt kan fokusera på nya forskningsproblem istället för att implementera grundläggande funktionalitet från grunden.

I projektet studeras manövrerna kolonnkörning, korsningskoordinering och rondellkoordinering. Forskningsfrågorna besvaras genom att formulera säkerhetskrav och implementera lösningar för varje trafikscenario. Lösningarna bygger på tidigare forskning men har anpassats till laboratoriets evalueringsmiljö, kommunikationsstruktur och robotplattformar. Projektet introducerar dessutom en enhetlig mjukvaruarkitektur där samtliga manövrer använder samma segmentbaserade vägmodell och gemensamma kommunikationsprinciper. För varje manöver motiveras varför algoritmerna förväntas fungera säkert. Genom experiment visas det att kraven är uppfyllda och på så vis att forskningsfrågorna är besvarade. Specifikt visar experimenten följande:

- Nödstoppet kan stanna roboten innan kollision vid hastigheter upp till 0,8 m/s.
- Kolonnkörning konvergerar mot stabil körning vid hastigheter upp till 0,5 m/s, med en konvergenstid på 20,4 s vid 0,3 m/s och 12,5 s vid 0,5 m/s.
- Korsningskoordineringens genomsnittliga passagetid för VTL-lösningen är {7,97; 5,58; 4,55} s vid hastigheterna $v \in \{0,3; 0,5; 0,6\}$ m/s. VTL-lösningen kan hantera hastigheter upp till 0,6 m/s.
- Korsningskoordineringens genomsnittliga passagetid för den reservationsbaserade lösningen är {7,49; 5,50; 4,55} s vid hastigheterna $v \in \{0,3; 0,5; 0,6\}$ m/s. Reservationsbaserade lösningen kan hantera hastigheter utan kollision upp till 1,1 m/s.
- Rondellkoordineringens passagetid är {11,39; 9,73; 8,55} s vid hastigheterna $v \in \{0,4; 0,5; 0,6\}$ m/s. Manövern kan hantera hastigheter upp till 0,6 m/s varefter säkert genomförande inte längre kan garanteras.

Experimenten visar att de implementerade systemen uppfyller respektive manövers säkerhetskrav och kan genomföra manövrerna inom ändlig tid. Arbetet förväntas underlätta framtida kandidatarbeten inom laboratoriet genom att erbjuda ett återanvändbart system för kooperativa manövrer, evaluering och säkerhetskritisk koordinering av autonoma robotar.

2

Bakgrund

Detta kapitel ger den tekniska bakgrund som krävs för att förstå systemet och de manövrer som projektet baseras på. Först beskrivs den evalueringsmiljö som har använts för evaluering av varje manöver. Därefter presenteras tekniker som har använts genomgående under projektet.

2.1 Evalueringsmiljö

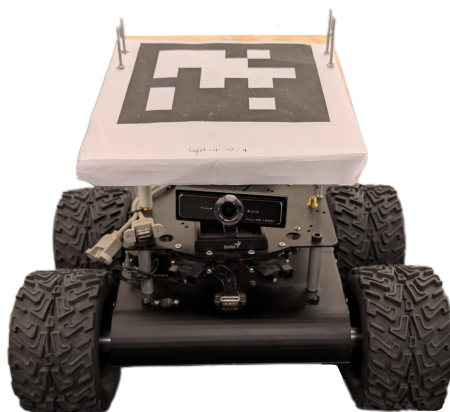
Projektet utförs i en kontrollerad inomhusmiljö för nedskalade autonoma fordon. För att utveckla och evaluera manövrer används en testbädd vid Chalmers tekniska högskola som successivt har utvecklats genom flera kandidatarbeten. Testbädden är utformad för att efterlikna en verklig trafikmiljö med raksträckor, korsningar och rondellområden, vilket gör det möjligt att studera vanliga trafikscenarier under reproducerbara förhållanden, se bild 2.1. I evalueringsmiljön används GulliView för att fastställa position och riktning för robotar på testbädden, se avsnitt 2.4. Lokala väderstreck definieras för att standardisera riktningar inom evalueringsmiljön. Den östliga riktningen definieras som mot fönstren i rummet, den västliga som mot whiteboardtavlan, den sydliga som mot borden och den nordliga som motsatt riktning till den sydliga. Se figur 2.1 för referens. Kommunikationen i evalueringsmiljön sker genom ett WiFi-nätverk vid namn Rostig.



Figur 2.1: Testbädden inom evalueringsmiljön

2.2 WiFiBot

WiFiBot är en robotplattform som används i evalueringsmiljön som nedskalade autonoma fordon. WifiBot är en prisvärd öppen mobil plattform för att utveckla eller lära sig robotik och artificiell intelligens [8]. Plattformen består av ett chassi utrustat med kamera, enkortsdator, WiFi-kort samt möjlighet att montera olika sensorer. Roboten har fyra hjul men modelleras som ett differentialstyrt system, där drivningen på respektive sida (höger och vänster) styrs gemensamt per sida. [2]. Varje robot är markerad med en unik AprilTag, se avsnitt 2.3 monterad ovanpå roboten, vilket gör den identifierbar för lokaliseringssystemet GulliView, se avsnitt 2.4. På varje robots AprilTag finns ett AprilTag-ID som används som dess identifieringsnummer. Bild 2.2 visar utseendet av en WiFiBot och placeringen av AprilTagen.



Figur 2.2: WifiBot med AprilTag-ID 4. Försedd med en kamera, infraröda sensorer och en unik AprilTag

2.3 AprilTags

AprilTags är visuella markörer, liknande QR-koder till sin struktur, som är utformade för att enkelt kunna detekteras av kameror och användas för att bestämma position och riktning i en bild [9]. I evalueringsmiljön fungerar de som referenspunkter som gör det möjligt för lokaliseringssystemet GulliView att lokalisera robotar eller andra objekt försedda med en AprilTag på testbädden.

2.4 GulliView

GulliView är ett lokaliseringssystem som utvecklats vid Chalmers tekniska högskola under flera tidigare arbeten. Systemet använder fyra takmonterade kameror kopplade till en central dator för att lokalisera AprilTags. För varje AprilTag som detekteras beräknas en position och riktning. GulliViews koordinatsystem har origo i det nordvästra hörnet av testbädden. x -axeln ökar i östlig riktning och y -axeln ökar i sydlig riktning. I detta projekt används

GulliView versionen `gulliview_matei` som använder UDP för att distribuera lokaliseringsdata över WiFi-nätverket `Rostig`. Varje meddelande innehåller position, riktning, AprilTag-ID och vilken kamera som informationen kommer från.

2.5 UDP

User Datagram Protocol (UDP) är ett transportlagerprotokoll som möjliggör anslutningslös och best-effort-baserad överföring av data över IP-nätverk [10]. Best-effort betyder att det inte finns någon garanti att skickad data kommer fram till mottagaren. UDP används av GulliView för att skicka lokaliseringsdata för robotar och för kommunikation mellan robotar.

2.6 ROS

Robot Operating System (ROS) är ett mjukvaruramverk som används för utveckling av robotapplikationer. Det är inte ett operativsystem i traditionell mening, utan fungerar som ett kommunikationslager ovanpå operativsystem och möjliggör strukturering av robotsystem [11]. ROS kan underlätta utveckling genom att dela upp funktionalitet i mindre, modulära delar som kan samverka. Dessa delar kallas ROS-noder vilket är exekverbara processer som utför en specifik uppgift, exempelvis sensordatahantering, beräkningar eller styrning av robotar [11]. Ett robotsystem består av flera noder som körs parallellt och tillsammans utgör systemets funktionalitet. ROS-topics används för kommunikation mellan noder. En ROS-topic är en namngiven kanal där noder kan skicka och ta emot meddelanden enligt publish-subscribe-modellen [11]. Noder kan producera data utan att känna till vilka noder som tar emot den, vilket ger en lös koppling mellan systemets komponenter. Med hjälp av ROS blir det möjligt att separera delar av systemet så som utförandet av manövrer och ruttplanering vilket gör systemet mer modulärt och på så vis lättare att förstå och vidareutveckla.

2.7 Kolonnkörning

Kolonnkörning, även kallat platooning, innebär att flera fordon körs tätt efter varandra i en konvoj [12]. I mer avancerade former av kolonnkörning tillämpas trådlös kommunikation för sammankoppling av fordon via system som Cooperative Adaptive Cruise Control (CACC). Genom att fordonen kontinuerligt skickar data om inbromsning och acceleration elimineras den mänskliga reaktionstiden, vilket möjliggör tidsluckor mellan fordonen ned till 0,2 sekunder [12]. I projektet är CACC en lämplig metod eftersom det i evalueringsmiljön redan finns en etablerad kommunikationsstruktur där data kan skickas mellan robotarna.

3

Metod

Detta kapitel beskriver de metoder som används för att möjliggöra kooperativ autonom körning inom evalueringsmiljön. Systemet är uppbyggt kring en modulär och distribuerad arkitektur där flera manöverspecifika algoritmer samverkar för att hantera olika trafiksituationer. För att säkerställa säker och robust drift kombineras kontinuerlig positionsdelning, distribuerad kommunikation och lokala beslutsalgoritmer. Kapitlet presenterar först den övergripande säkerhetsfunktionen för nödstopp, följt av metoder för korsningskoordinering, rondellkoordinering och kolonnkörning. För varje delsystem beskrivs i detalj krav, vald modell, beslutslogik samt argument för kravuppfyllelse.

3.1 Nödstopp

Nödstoppet är en övergripande och oberoende säkerhetsmekanism som övervakar situationen för att minska risken för kollisioner. Funktionen utvärderas i varje uppdateringscykel oberoende av annan styrlogik, såsom kolonnkörning eller farthållning. Syftet är att agera som ett sista skyddsnät som har högsta prioritet och åsidosätter alla andra hastighetskommandon för att förhindra en kollision.

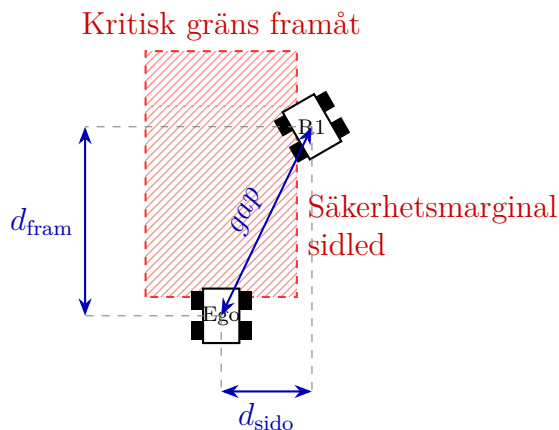
3.1.1 Krav

För nödstoppsfunktionen formulerades följande krav:

- **E1: Prioritet** - Nödstoppet ska alltid utvärderas först. Om en kollisionsrisk detekteras ska all annan logik åsidosättas omedelbart.
- **E2: Kollisionsavvärjning** - Systemet ska stoppa roboten ifall avståndet blir för litet, eller den relativa hastigheten indikerar en kollision.
- **E3: Stabilitet** - Systemet får inte oscillera mellan aktiverat och inaktiverat tillstånd vid gränsvärdena.

3.1.2 Vald metod och modell

Nödstoppet bygger på en modell som utvärderar förhållandet till alla andra robotarnas position i förhållanden till sin egen position. Genom GulliView erhålls positionen och riktningen för de andra robotarna. Systemet räknar sedan ut tre värden till varje robot: avståndet (gap), avståndet rakt fram (d_{fram}) och avståndet i sidled (d_{sido}), se figur 3.1. Ifall något av dessa värden understiger den gränsen som definieras aktiveras nödstoppet.



Figur 3.1: Geometrisk beskrivning av nödstoppets utvärderingsmodell. Figuren visar avståndet fram till den andra roboten (d_{fram}), det laterala avståndet (d_{sido}), samt det kortaste fria avståndet (gap) i relation till den definierade nödstoppszonen.

Utöver de rena avstånden används också en tid till kollision faktor (TTC). Genom att beräkna den relativa hastigheten ($v_{relativ}$) mellan robotarna och avståndet till den andra roboten kan tiden till en kollision uppskattas. Ekvationen för t_{tc} är:

$$TTC = \frac{d_{fram}}{v_{relativ}}, \quad \text{för } v_{relativ} > 0. \quad (3.1)$$

Nödstoppet aktiveras utifrån tre kriterier relaterade till avstånd och kollisionsrisk. Systemet aktiveras om en framförvarande robot befinner sig i färdvägen och avståndet (gap) underskrider det fördefinierade kritiska gränsvärdet. För att hantera objekt som närmar sig med hög relativ hastighet aktiveras stoppet om den beräknade tiden till kollision (TTC) faller under ett visst gränsvärde. Slutligen utlöses nödstoppet vid sidoinkräktning, vilket sker när en annan robot överlappar den egna säkerhetszonen från sidan och avståndet därigenom blir kritiskt.

För att hantera kravet på stabilitet så tillämpas hysteres. Detta är en teknik som innebär att när nödstoppet har aktiverats räcker det inte enbart att den andra roboten hamnar precis utanför det kritiska avståndet. För att nödstoppet ska släppas krävs det att framsidan är fri ($d_{fram} > \text{säkerhetsmarginal}$) samt att avståndet (gap) är större än det kritiska avståndet plus en fördefinierad buffert.

3.1.3 Argument för kravuppfyllelse

Genom att nödstoppet utvärderas oberoende och har prioritet över de andra manövrerna uppfylls E1 kravet på prioritet. För att stanna roboten, vilket är krav E2, kontrollerar systemet flera olika variabler som aktiverar stoppet. Detta gör systemet mer robust och säkrare för att det finns flera olika kontroller som kan aktivera stoppet. För krav E3 på stabilitet tillämpas hysteres som teknik, vilket gör att roboten inte kommer att starta och stanna med korta mellanrum utan väntar tills den uppfattade risken har minskat tillräckligt innan den börjar köra igen.

3.2 Kolonnkörning

Kolonnkörning används när flera robotar färdas efter varandra och behöver anpassa sina hastigheter. Den robot som kör bakom, följaren, anpassar sin hastighet efter den framförvarande roboten, ledaren. Syftet är att hålla ett önskat avstånd till ledaren, utan kraftiga eller oscillerande hastighetsvariationer.

3.2.1 Krav

För kolonnkörning formuleras följande krav:

- **P1: Säkerhet** - När en framförvarande robot identifieras får det uppmätta avståndet g aldrig understiga det kritiska avståndet 0,35 m.
- **P2: Framkomlighet** - När en framförvarande robot identifieras ska följaren nå ett tillstånd där avståndsfelet går mot 0 och hastigheten närmar sig ledarens.

3.2.2 Vald metod och modell

För kolonnkörning används en CACC lösning baserad på Ploegs modell [3]. CACC valdes eftersom regulatorn tar hänsyn till både följarens tillstånd och ledarens rörelse, till skillnad från en traditionell adaptiv farthållning som endast utgår från följarens egen sensordata. Att CACC även använder ledarens hastighet och acceleration möjliggör tätare och stabilare kolonnkörning.

Modellen bygger på att det önskade avståndet till ledaren växer med följarens hastighet. Detta beskrivs med ett tidsavstånd, betecknat h i Ploegs modell enligt

$$g_{des} = d_0 + h \cdot v_{ego} \quad (3.2)$$

där g_{des} betecknar det önskade avståndet, d_0 stillastående säkerhetsavståndet, och h tidsavståndet till ledaren. Variabeln v_{ego} är hastigheten för ego-robot, vilket

motsvarar följaren, det vill säga den robot som kör CACC-regulatorn. Det uppmätta avstånd till ledaren betecknas g , och avståndsfelet definieras som

$$g_{err} = g - g_{des} \quad (3.3)$$

Regulatorn använder avståndsfelet, samt hastighet och acceleration för följare och ledare för att beräkna en accelerationsändring enligt Ploegs CACC-modell:

$$\Delta a = \frac{1}{h} (-a_{ego} + k_p \cdot g_{err} + k_d(v_{pred} - v_{ego} - h \cdot a_{ego}) + a_{pred}) \cdot dt \quad (3.4)$$

I ekvationen betecknar v_{pred} och a_{pred} ledarens hastighet och acceleration, medan v_{ego} och a_{ego} gäller för följaren. Konstanterna k_p och k_d bestämmer hur starkt regulatorn reagerar på avståndsfel och relativa hastighetsskillnader. Den beräknade accelerationsändringen Δa används för att ta fram en ny referenshastighet för följaren.

3.2.3 Beslutslogik

Algoritm 1 beskriver logiken för kolonnkörning som körs på varje enskild robot i varje uppdateringscykel.

Algoritm 1 Uppdateringslogik för kolonnkörning

```

1: procedure UPDATE
2:                                     ▷ First check if needs to stop
3:   if should_emergency_stop(my_speed) then
4:     Set speed limit to 0
5:     return
6:   end if
7:   predecessor ← find closest bot in front with same heading
8:   if predecessor is None then
9:                                     ▷ No leader found, follow set speed
10:    Set speed limit to base_speed
11:  else
12:                                     ▷ Use CACC to calculate speed
13:    gap ← calculate gap to predecessor
14:    desired_speed ← CACC(gap, ego_state, predecessor_state)
15:    Set speed limit to desired_speed
16:  end if
17: end procedure

```

Beslutslogiken kan delas upp i tre sekventiella kontroller. Först sker ett säkerhetsstopp som kontrollerar ifall en robot är på väg att kollidera med en annan roboten. Om en kollisionsrisk detekteras stannas roboten omedelbart och all annan logik åsidosätts. Se avsnitt 3.1 för detaljer. Därefter om ingen kollisionsrisk

detekteras kontrollerar roboten ifall det finns någon annan robot framför som färdas i samma riktning. Om inget sådant fordon identifieras antas vägen vara fri, och roboten sätter sin hastighet till den förinställda bashastigheten. Identifieras en framförvarande robot övergår logiken till kolonnkörning med CACC, där följaren börjar följa ledaren. Avståndet till ledaren beräknas och används som insignal till CACC-regulatorn, som beräknar en referenshastighet för följaren som justeras mot ledarens för att bibehålla ett säkert avstånd.

3.2.4 Identifiering av ledare

För att CACC-regulatorn ska aktiveras, måste varje robot kontinuerligt detektera en ledare framför. Identifiering sker utifrån positions- och riktningsdata från GulliView. Detta innebär att följaren kan upptäcka framförvarande robot så länge båda är synliga för lokaliseringssystemet. För varje kandidat beräknas avståndet framåt d_{fram} och i sidled d_{sido} relativt till följaren enligt

$$d_{fram} = \Delta x \cos \theta_{ego} + \Delta y \sin \theta_{ego} \quad (3.5)$$

$$d_{sido} = | -\Delta x \sin \theta_{ego} + \Delta y \cos \theta_{ego} | \quad (3.6)$$

där Δx och Δy är positionsskillnaden mellan kandidatroboten och följaren, och θ_{ego} är följarens färdriktning. En kandidat betraktas som en möjlig ledare om den befinner sig framför följaren ($d_{fram} > 0$), har samma färdriktning och har ett AprilTag-ID som tillhör robotarna. Uppfyller flera kandidater villkoren, väljs den med kortast d_{fram} . När en ny ledare väljs återställs CACC-regulatorns interna tillstånd så att information från den föregående ledaren inte påverkar den nya regleringen.

3.2.5 Argument för kravuppfyllelse

Krav P1 säkerställs av CACC-regulatorns konstruktion. Det önskade avståndet $g_{des} = d_0 + h \cdot v_{ego}$ är konstruerat så att det aldrig blir mindre än det kritiska avståndet $d_0 = 0,35$ m. Det uppmätta avståndet mellan robotarna g kommer att hållas ovanför kritiska gränsen så länge regulatorn beräknar det önskade avståndet. Vid hastighetsändringar, exempelvis vid inbromsning, kan g tillfälligt falla under g_{des} men regulatorns framkoppling gör att följaren reagerar snabbt och att avvikelsen förblir liten enligt Ploegs CACC-modell [3]. Det komplementerande säkerhetslagret är nödstoppet som sätter hastighetsgränsen till noll om g blir kritiskt, se avsnitt 3.1.

Framkomlighetskravet P2 följer samma regulator. Ploegs CACC-modell [3] är konstruerad för att driva avståndsfelet g_{err} mot noll och följarens hastighet mot ledarens. Eftersom regulatorn använder ledarens hastighet och acceleration ges en snabbare återkoppling vid hastighetsförändringar jämfört med en regulator som enbart utgår från det uppmätta avståndet.

3.3 Korsningskoordinering

Korsningskoordinering används när en självkörande robot närmar sig en korsning och behöver avgöra om den kan passera utan att hamna i konflikt med andra robotar. Syftet är att skapa en beslutsmodell som både förhindrar kollisioner och samtidigt begränsar onödig väntetid.

3.3.1 Krav

För korsningslogiken formuleras två krav:

- **K1: Säkerhet** - Robotar med korsande färdvägar ska inte ges tillträde till konfliktområdet samtidigt. Lösningen ska även hantera saknad, fördröjd eller gammal information eftersom besluten bygger på distribuerad kommunikation mellan robotarna.
- **K2: Framkomlighet inom ändlig tid** - Lösningen ska säkerställa framkomlighet genom att begränsa väntetid, onödiga stopp och situationer där en robot nekas passage under obestämd tid.

3.3.2 Vald metod och modell

I evalueringsmiljön delas korsningsområdet in i tre separata trevägskorsningar utifrån testbaddens geometri. Dessa benämns korsning 1, korsning 2 och korsning 3. Eftersom korsningarna har olika utformning används inte samma beslutsmodell för samtliga korsningar. Korsning 1 och 2 hanteras med VTL-reglering, medan korsning 3 hanteras med reservationslogik.

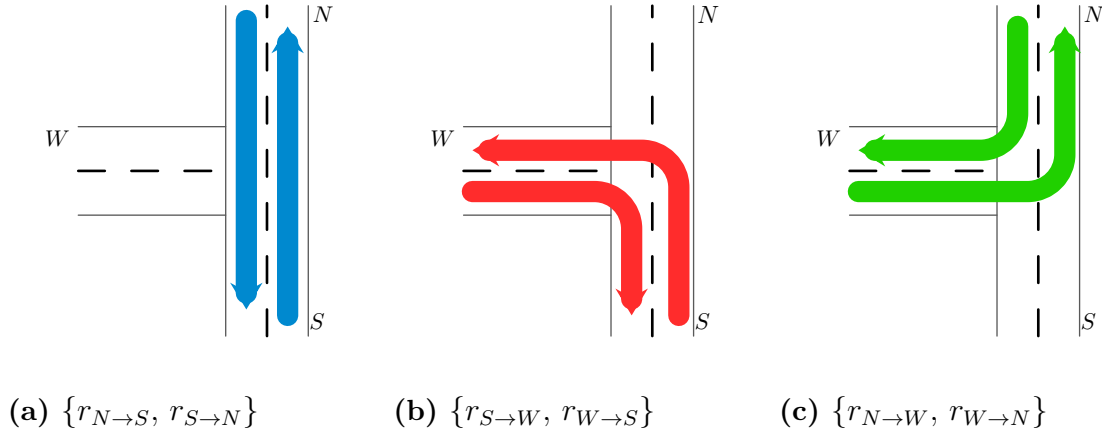
I detta arbete innebär VTL-reglering att robotarna inte följer fysiska trafikljus, utan i stället använder en gemensam logik för att avgöra vilka färdvägar som tillfälligt får prioritet. Den gruppbaseade VTL-regleringen är inspirerad av Protocol 1 hos Casimiro et al. [5], där VTL beskrivs som en prioritetstilldelning som säkerställer att fordon med samma prioritet inte har korsande färdvägar. I detta arbete tillämpas denna princip genom signalgrupper. En signalgrupp definieras som den mängd rörelser som kan tillåtas samtidigt utan att robotarnas färdvägar korsar varandra i konfliktområdet. För att beskriva rörelserna och zonerna (se avsnitt 3.3.3) används beteckningarna N , S , E och W som lokala namn i evalueringsmiljön. De motsvarar norr, söder, öst och väst, men ska inte tolkas som geografiska väderstreck. En rörelse betecknas som $r_{A \rightarrow B}$, där A är vägarmen roboten kommer från och B är vägarmen roboten kör mot. Med vägarm avses den anslutande körbanan som leder in till eller ut från korsningen.

För korsning 1 används vägarmarna N , S och W , se figur 3.2. Signalgrupperna definieras som

$$\mathcal{G}^{(1)} = \{ \{r_{N \rightarrow S}, r_{S \rightarrow N}\}, \{r_{S \rightarrow W}, r_{W \rightarrow S}\}, \{r_{N \rightarrow W}, r_{W \rightarrow N}\} \}. \quad (3.7)$$

För korsning 2 används vägmarna N , S och E . Signalgrupperna definieras som

$$\mathcal{G}^{(2)} = \{\{r_{N \rightarrow S}, r_{S \rightarrow N}\}, \{r_{S \rightarrow E}, r_{E \rightarrow S}\}, \{r_{N \rightarrow E}, r_{E \rightarrow N}\}\}. \quad (3.8)$$



Figur 3.2: Visualisering av tillåtna rörelser i korsning 1.

Vid varje uppdatering av VTL-logik finns en aktiv signalgrupp, med det menas den grupp av rörelser som har prioritet i korsning. Robotar vars planerade rörelse tillhör den aktiva gruppen får passera med signalen *Green*. Robotar vars rörelse inte tillhör den aktiva gruppen nekas inträde med signalen *Red*. Vid övergång mellan två grupper används signalen *Yellow*, vilket motsvarar ett övergångstillstånd.

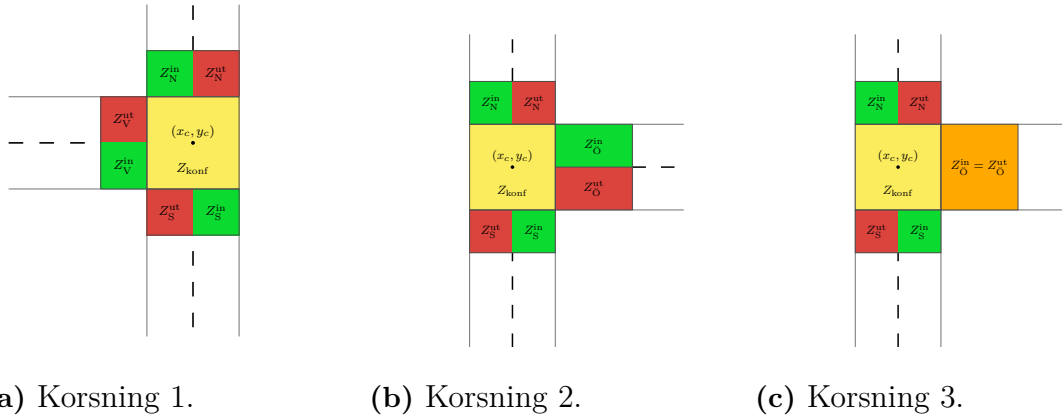
Korsning 3 hanteras i stället med reservationslogik. Denna logik är inspirerad av distribuerade korsningsmetoder där fordon koordinerar passageordning och identifierar konfliktområden för att undvika kollision [6]. Anledningen är att korsning 3 har en gemensam östlig vägmarm som används både som infart och utfart, se figur 3.3c. Situationen liknar en korsning med väjningsplikt, där en robot behöver vänta tills korsningen är fri. Beslutet i korsning 3 baseras därför på om konfliktzonen och den gemensamma vägmarmen kan reserveras utan konflikt.

3.3.3 Representation av korsningar och zoner

För att koppla korsningslogiken till konkreta positioner modelleras varje korsning med ett centrum (x_c, y_c) , en konfliktzon Z_{konf} samt in- och utfartszoner $Z^{in/ut}$ för de vägmarmar som ansluter till korsningen. Den zonbaserade representationen används för att avgöra om en robot närmar sig korsningen, befinner sig i konfliktområdet eller håller på att lämna korsningen. Denna typ av uppdelning är inspirerad av distribuerade korsningsmetoder där korsningsområdet delas in i delområden för att avgöra om planerade rörelser ger upphov till konflikt [6]. I detta arbete anpassas principen till testbadden genom separata infarts-, utfarts- och konfliktzoner.

För korsning 1 används vägmarna norr, söder och väst. För korsning 2 används vägmarna norr, söder och öst. I dessa två korsningar behandlas infart- och

utfartzoner som separata zoner för varje vägmarm. Korsning 3 skiljer sig från de övriga genom att den östra vägmarmen fungerar som både infart och utfart. Därför modelleras den östra vägmarmen som en gemensam zon som endast får användas av en robot åt gången. Zonindelningen för samtliga korsningar visas i figur 3.3.



Figur 3.3: Zonindelning för de tre korsningarna i evalueringsmiljön. Grönt markerar infartszoner, rött markerar utfartszoner, gult markerar konfliktzoner och orange markerar den gemensamma in- och utfartszonen i korsning 3.

3.3.4 Beslutslogik

Beslutslogiken körs kontinuerligt i en uppdateringscykel för varje robot. I varje cykel avgörs först robotens aktuella tillstånd i förhållande till korsningen. Därefter bedöms om roboten ska tillåtas passera eller vänta. Eftersom korsning 1 och 2 tillämpar gruppbasead VTL-logik, medan korsning 3 tillämpar reservationslogik, beskrivs beslutsflödet med två separata algoritmer.

Algorithm 2 Korsning 1 och 2 uppdateringslogik

```

1: procedure UPDATE
2:   Determine the planned movement from the robot route
3:   Determine the zone state from the robot position
4:   Read available information about nearby robots
5:   if the robot is outside, crossing, or leaving the intersection then
6:     Allow continued passage
7:     return
8:   end if
9:   if the planned movement does not belong to the current intersection then
10:    Allow continued driving without intersection regulation
11:    return
12:   end if
                                     ▷ The robot is approaching the intersection
13:   if information about relevant robots is reliable then
14:     Select the active signal group from the shared VTL plan
15:     if the same signal group has been prioritized repeatedly and another
group is waiting then
16:       Select another waiting signal group
17:     end if
18:   else
19:     Select the active signal group from a fixed rotating schedule
20:   end if
21:   if the planned movement belongs to the active signal group then
22:     if the active signal group is in a transition phase then
23:       Set signal to Yellow
24:       Reduce speed
25:     else
26:       Set signal to Green
27:       Allow passage
28:     end if
29:   else
30:     Set signal to Red
31:     Deny entry
32:   end if
33: end procedure

```

Algorithm 3 Korsning 3 uppdateringslogik

```
1: procedure UPDATE
2:   Determine the planned movement from the robot route
3:   Determine the zone state from the robot position
4:   Read available information about nearby robots
5:   if the robot is outside, crossing, or leaving the intersection then
6:     Allow continued passage
7:     return
8:   end if
9:   if the planned movement does not belong to intersection 3 then
10:    Allow continued driving without intersection regulation
11:    return
12:   end if
13:   if the robot has just entered the approach zone then
14:     Wait briefly to collect recent information
15:   end if
16:   if the waiting period has not elapsed then
17:     Deny entry
18:     return
19:   end if
20:   if information about relevant robots is not reliable then
21:     Deny entry
22:     return
23:   end if
24:   if another relevant robot is crossing or leaving the intersection then
25:     Deny entry
26:     return
27:   end if
28:   if another relevant robot occupies the shared road arm then
29:     Deny entry
30:     return
31:   end if
32:   if another waiting relevant robot has higher priority then
33:     Deny entry
34:     return
35:   end if
36:   Allow passage
37: end procedure
```

Algorithm 2 och algorithm 3 visar den övergripande beslutslogiken. Båda algoritmerna utgår från robotens position, planerade rörelse och tillgänglig information om andra robotar. Vilken algorithm som används beror på vilken korsning roboten närmar sig.

Först avgörs om roboten behöver regleras av korsningslogiken. Robotar som är utanför korsningen, redan befinner sig i konfliktzonen eller är på väg ut ur korsningen tillåts fortsätta, eftersom ett stopp i dessa lägen kan blockera

korsningen. Därefter kontrolleras om robotens planerade rörelse tillhör den aktuella korsningen.

Vid korsning 1 och 2 används gruppbaserad VTL-logik. Roboten får passera om dess rörelse tillhör den aktiva signalgruppen, annars nekas inträde. Vid övergångstillstånd mellan signalgrupperna sänks hastigheten för de robotar som befinner sig i infartzonen. Vid korsning 3 används reservationslogik. Roboten får köra in endast om informationen om andra robotar är tillförlitlig, konfliktområdet är fritt och den gemensamma vägarmen inte är upptagen.

3.3.5 Argument för kravuppfyllelse

Säkerhetskravet K1 uppfylls genom att roboten endast får köra in i korsningen när dess planerade rörelser är tillåtna enligt den aktuella regleringen. För VTL-logiken innebär detta att rörelsen behöver tillhöra den aktiva signalgruppen. Eftersom signalgrupperna består av rörelser som inte korsar varandra minskar risken för att två robotar ges tillträde till samma konfliktområde samtidigt. För reservationslogiken hanteras säkerhetskravet genom att roboten nekas inträde om en annan relevant robot redan befinner sig i korsningen, lämnar korsningen eller upptar den gemensamma vägarmen. Om informationen om andra robotar är gammal eller saknas används den inte som grund för ett fritt inträdesbeslut. För VTL-logiken används ett fast roterande schema mellan signalgrupperna i stället för en efterfrågestyrd plan. För reservationslogiken nekas inträde tills tillförlitlig information finns tillgänglig. På detta sätt prioriteras säkerhet framför framkomlighet när kommunikationsläget är osäkert.

Framkomlighetskravet K2 behandlas på två sätt. För det första tillåts robotar som redan befinner sig i konfliktzonen eller i utfartszonen fortsätta ut ur korsningen även om signalgruppen har Red, vilket minskar risken för blockering. För det andra växlar VTL-logiken prioritet mellan signalgrupper. Om samma signalgrupp har prioriterats flera gånger och en annan signalgrupp väntar, flyttas prioriteten till den väntande gruppen. Detta minskar risken för att en signalgrupp dominerar korsningen under en längre tid. I reservationslogiken får roboten passera så snart ingen relevant robot reserverar korsningsområdet.

3.4 Rondellkoordinering

Rondellmanövern ska kunna hantera de situationer som uppstår när flera självkörande robotar interagerar i en rondell. Precis som i vanlig trafik bygger manövern på principen att fordonet alltid ska lämna företräde åt cirkulerande trafik och endast köra in i rondellen när det kan ske utan risk för kollision.

3.4.1 Krav

För att systemet ska kunna fatta tillförlitliga och kollisionsfria beslut utifrån denna modell ställs följande krav på dess funktion:

- **R1: Säkerhetsbedömning** - Robotarna ska kunna genomföra rondellpassagen utan fysisk kollision.
- **R2: Framkomlighet inom ändlig tid** - Algoritmen eftersträvar god framkomlighet genom att begränsa väntetiden, onödiga stopp och situationer där en robot nekas passage under obestämd tid.

3.4.2 Vald metod och modell

Många befintliga algoritmer är bristfälliga utifrån detta projekts förutsättningar. Flera tillvägagångssätt förlitar sig på en centraliserad arkitektur, där ett separat system hanterar rondellens trafiklogik, snarare än att låta de enskilda robotarna agera autonomt [7]. Andra modeller fokuserar uteslutande på att teoretiskt maximera genomströmningen i rondellen genom komplexa optimeringsalgoritmer, men saknar direkt tillämpbar manövreringslogik för enskilda robotar.

Lösningen i föregående kandidatarbete [1] bygger på en återanvändning av befintlig logik för vanliga korsningar, där varje infart till rondellen i praktiken behandlas som en isolerad korsning. Detta tillvägagångssätt bedöms vara otillräckligt för att åstadkomma ett dynamiskt och realistiskt trafikbeteende i en rondell.

En avgörande insikt från granskningen av tidigare arbeten är att algoritmerna ofta förlitar sig på explicit robot-till-robot-kommunikation. I sådana system tvingas robotarna kontinuerligt sända ut dedikerade statusmeddelanden till varandra gällande huruvida de befinner sig inuti eller utanför rondellen. Denna metod bedöms vara ineffektiv, i synnerhet då meddelanden ofta skickas även när andra robotar inte befinner sig i närheten eller ens interagerar med rondellen. Eftersom systemet redan kontinuerligt tillhandahåller alla robotars globala positioner via positioneringssystemet GulliView, väljs den explicita kommunikationen bort. Lösningen utformas därmed så att en robot, först när den faktiskt anländer till en infart, itererar över och utvärderar övriga robotars positioner. Genom att utnyttja robotarnas position kan roboten lokalt avgöra om det finns robotar i rondellen och självständigt fatta ett kollisionsfritt beslut om att antingen köra in eller stanna och lämna företräde.

3.4.3 Beslutslogik

Algoritm 4 beskriver logiken som körs på varje enskild robot i varje uppdateringscykel.

Algoritm 4 Rondellens uppdateringslogik

```
1: procedure UPDATE
2:   if my_bot is not in roundabout_zone then
3:     if was_driving_in_roundabout then
4:       Log event: "Exiting roundabout"
5:       Set driving_in_roundabout to false
6:       Reset speed limit to base speed
7:     end if
8:     return
9:   end if
10:  if driving_in_roundabout then
11:    Log event: "Driving in roundabout, driving to exit"
12:    return
13:  end if
14:     $\triangleright$  Robot is currently at the entry zone and wants to enter
15:  bots_in_roundabout  $\leftarrow$  list of all other bots inside roundabout_zone
16:  closest_entry  $\leftarrow$  find closest entry zone to my_bot
17:  if safe to enter roundabout (closest_entry, bots_in_roundabout) then
18:    Set driving_in_roundabout to true
19:    Set speed limit to roundabout speed
20:    Log event: "Entering roundabout"
21:  else
22:    Set driving_in_roundabout to false
23:    Set speed limit to 0
24:    Log event: "Not safe to enter roundabout"
25:  end if
26: end procedure
```

Utvärderingen sker kontinuerligt i en uppdateringscykel och hanteringen anpassas utifrån tre huvudsakliga tillstånd baserade robotens aktuella position. Först och främst kontrolleras huruvida roboten befinner sig utanför den definierade rondellzonen. Om roboten har registrerats som körande inne i rondellen men nu befinner sig utanför, tolkas detta som att den har kört ut. Rondellmanövern anses då vara slutförd och robotens hastighetsgräns återställs till systemets standardhastighet. Skulle roboten istället finnas inuti rondellzonen och variabeln *driving_in_roundabout* redan är satt till sant, sker inga ytterligare uppdateringar av infartsvillkoren. Roboten fortsätter då helt enkelt sin planerade rutt fram mot utfarten utan att utföra nya säkerhetsprovningar för själva infarten.

Det tredje tillståndet inträffar när roboten anländer till rondellen och ska göra en säkerhetsbedömning vid infarten. Systemet identifierar och listar då alla övriga

robotar som för närvarande befinner sig inne i rondellen. Genom att jämföra dessa robotars positioner med den närmaste infartszonens position bedömer systemet om den tillgängliga tidsluckan (*timetoentry*) överstiger de tre sekunder som krävs för en kollisionsfri inkörning genom rondellen. Värdet tre sekunder fastställs empiriskt, eftersom kortare tidsluckor leder till ökad kollisionsrisk. Om tidsluckan bedöms vara tillräcklig ändras robotens variabel *driving_in_roundabout* till sann. Skulle tidsluckan däremot vara för liten på grund av annan trafik i rondellen stannar roboten omedelbart och inväntar en tillräckligt stor tidslucka för att köra in.

För att bestämma den tillgängliga tidsluckan t_{lucka} används den beräknade båglängden $d_{\text{båge}}$ mellan en robot som befinner sig i rondellen och den infart som är närmast den väntande roboten. För att bestämma båglängden beräknas först vinkeln mellan rondellens mittpunkt $(x_{\text{center}}, y_{\text{center}})$ och robotens position $(x_{\text{robot}}, y_{\text{robot}})$. För att säkerställa korrekt vinkel i samtliga kvadranter används funktionen atan2 enligt

$$\theta_{\text{robot}} = \text{atan2}(y_{\text{robot}} - y_{\text{center}}, x_{\text{robot}} - x_{\text{center}}) \quad (3.9)$$

Motsvarande vinkel beräknas för rondellens infartspunkt $(x_{\text{infart}}, y_{\text{infart}})$ enligt

$$\theta_{\text{infart}} = \text{atan2}(y_{\text{infart}} - y_{\text{center}}, x_{\text{infart}} - x_{\text{center}}) \quad (3.10)$$

Vinkelskillnaden beräknas enligt

$$\Delta\theta = (\theta_{\text{robot}} - \theta_{\text{infart}}) \bmod 2\pi \quad (3.11)$$

där modulooperatorn säkerställer att vinkeln representeras inom intervallet $[0, 2\pi]$.

Om roboten redan har passerat infarten kommer $\Delta\theta$ att ligga nära 2π . För att representera sådana robotar med negativa avstånd transformeras därför vinkelskillnaden enligt

$$\Delta\theta_{\text{korrr}} = \begin{cases} \Delta\theta - 2\pi, & \Delta\theta > \pi \\ \Delta\theta, & \text{annars} \end{cases} \quad (3.12)$$

vilket ger ett intervall $[-\pi, \pi]$. Ett positivt värde innebär att roboten ännu inte har nått infarten, medan ett negativt värde innebär att roboten redan passerat den. Båglängden längs rondellens cirkulära bana bestäms därefter enligt

$$d_{\text{båge}} = r_{\text{rondell}} \cdot \Delta\theta_{\text{korrr}} \quad (3.13)$$

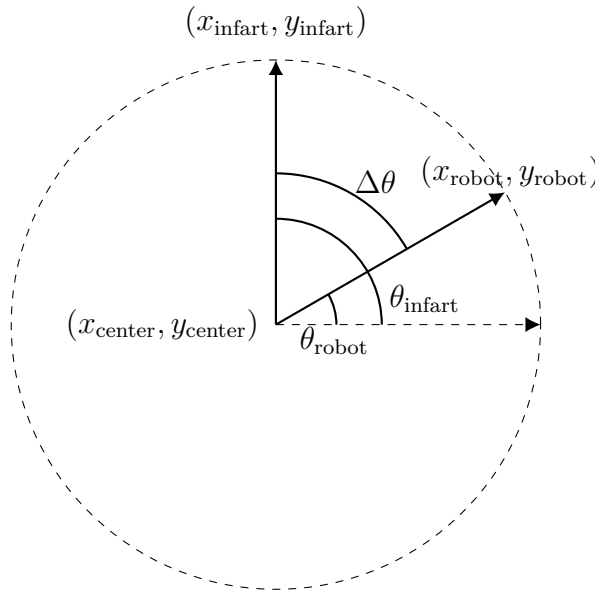
där r_{rondell} är rondellens radie. Om båglängden är negativ innebär detta att roboten redan passerat infarten. Om roboten precis passerat infarten betraktas tidsluckan konservativt som noll ($t_{\text{lucka}} = 0$) sekunder av säkerhetsskäl. Annars adderas rondellens omkrets för att erhålla det framåtriktade avståndet tills roboten åter når samma infart:

$$d_{\text{båge,korr}} = d_{\text{båge}} + 2\pi r_{\text{rondell}} \quad (3.14)$$

Den slutliga tidsluckan bestäms därefter enligt

$$t_{\text{lucka}} = \frac{d_{\text{båge,korr}}}{v} \quad (3.15)$$

Figur 3.4 illustrerar hur positionerna av robotar och infarten används för att härleda riktningar i form av vinklar relativt rondellens centrum.



Figur 3.4: Placering av koordinater och vinklar

Vinklarna i figur 3.4 motsvarar definitionerna i Ekvation 3.9 och Ekvation 3.10, medan $\Delta\theta$ motsvarar den normaliserade skillnaden i Ekvation 3.12.

3.4.4 Argument för kravuppfyllelse

Krav R1 behandlas genom att säkerställa att en robot inte tillåts köra in i rondellen förrän en tillräcklig tidslucka föreligger. För att bestämma tidsluckan antar roboten att trafiken i rondellen håller maximal hastighet och att robotarna följer en cirkelbåge vars radie motsvarar den inre cirkelns radie. På grund av detta

minimeras den beräknade tidsluckan, vilket innebär att robotar inom rondellen inte kan komma till infarten tidigare än den beräknade tiden.

Krav R2 säkerställs genom att förhindra blockering i rondellen samt onödiga väntetider. Om robotens position är inom en meter från rondellens kant börjar den kontrollera huruvida det finns andra robotar i rondellen och huruvida det finns en tillräckligt stor tidslucka för att köra in. Roboten kontrollerar situationen kontinuerligt, och om tidsluckan är otillräcklig för säker inkörning stannar den när den når rondellens infart. Detta säkerställer att roboten är medveten om situationen i rondellen redan innan inkörningen och förhindrar att den stannar för sent och blockerar trafiken i rondellen. För att förhindra onödig väntetid tillåts den väntande roboten köra in i rondellen så fort en tillräckligt stor tidslucka upptäcks. I en rondell kan en robot nekas passage under obegränsad tid om det finns ett kontinuerligt trafikflöde i rondellen. Detta betraktas inte som ett brott mot kravet, eftersom det är en grundläggande egenskap hos rondeller att cirkulerande trafik har företräde.

4

Enhetlig mjukvaruarkitektur och implementation

Det implementerade systemet bygger vidare på flera tidigare kandidatarbeten [1], [2]. Projektets implementation syftar till att integrera dessa i en enhetlig lösning där alla kritiska komponenter är samlade. Implementation av systemet kan delas upp i flera moduler. Implementationen under dessa moduler presenteras nedan.

4.1 Översikt över systemet

Systemet bestod initialt av två separata delar: projektets egen kodbas samt den externa motorstyrningsmodulen `roswifibot`. Dessa kompilerades skilt från varandra. För att förenkla uppstart och tydliggöra systemets beroenden integrerades `roswifibot` i projektets kodbas. Koden delades upp i separata moduler med avgränsade ansvarsområden för att realisera den modulära systemarkitekturen. Den slutgiltiga arkitekturen tog form i följande samverkande moduler och filer, se figur 4.1:

Motorstyrning (`roswifibot`): Modulen sköter styrningen av robotens fysiska motorer genom ROS. Den utvecklades under ett tidigare kandidatarbete [2].

Navigering och rörelse (`go_to_goal.py`): Filen beräknar vinkelskillnaden mellan robotens färdriktning och en given målpunkt. Målpunkter ges som segment och segmenten representeras som en graf, se avsnitt 4.2.

Koordinering (`cooperative_control.py`): Filen initierar och uppdaterar de processer som ansvarar för utförandet av manövrerlogiken.

Manöverlogik (`maneuvers`): Modulen innehåller den specifika logiken för manövrer.

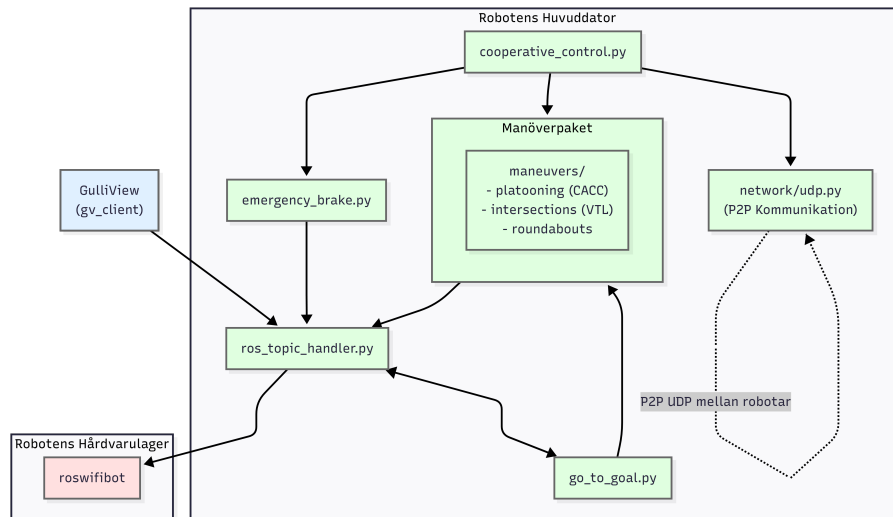
Nödstopp (`emergency_brake.py`): Filen bevakar avståndet till de andra robotarna och stannar roboten ifall den är på väg att krascha.

ROS abstraktion (ros_topic_handler.py): Filen fungerar som ett abstraktionslager mellan manövrer och ROS-topics. Filen sparar robotarnas positioner och används för att skicka den önskade hastigheten till motorstyrningen.

UDP-server/klient(udp.py): Filen implementerar ett UDP-baserat server/klient gränssnitt för kommunikation mellan robotar.

Gemensamma moduler (common): Modulen innehåller klasser som används i hela kodbasen.

Kommunikation av positioner (gv_client): Modulen tillhandahåller en UDP-baserad klient/server som tar emot global positioneringsdata för alla robotar från GulliView och publicerar det på en ROS-topic vid namn `gv_positions`.



Figur 4.1: Uppdelning och programflöde. (Egen figur, genererad med Mermaid (mermaid.ai) från egen kod.)

Den ursprungliga implementationen var svåröverskådlig på grund av bristande dokumentation och funktionsuppdelning. Varje modul granskades, och större funktioner delades upp i mindre, mer avgränsade funktioner för att förbättra kodens läsbarhet och underhållbarhet. Därefter försågs varje funktion med tydlig dokumentation som beskriver funktionens syfte och parametrar.

Utöver dokumentation i koden saknades även övergripande dokumentation för projektet. För att underlätta för framtida projekt att navigera i kodbasen och förstå varje moduls syfte lades en **README**-fil till i varje mapp. Där framgår vilka moduler som finns i respektive mapp och vilken funktion de fyller.

4.2 Ruttmodell

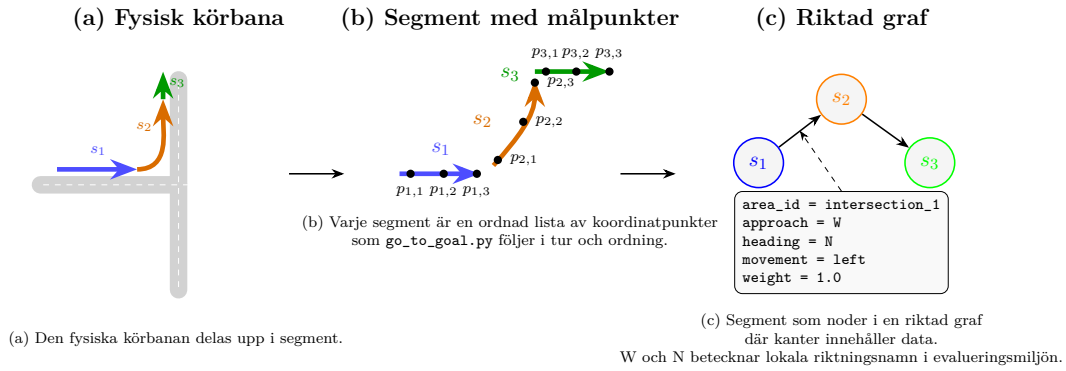
För att en robot skulle kunna köra autonomt i testbädden behövde systemet en intern modell av vilka vägar som fanns och hur roboten fick röra sig mellan dem. Den fysiska testbädden bestod av markerade körbanor, korsningar och rondellområden, men robotens mjukvara kunde inte tolka dessa vägmarkeringar direkt. Därför representerades de körbara delar som en ruttmodell, beskriven med koordinatpunkter i Gulliviews koordinatsystem, vilket `go_to_goal.py` använde för att styra roboten mot nästa målpunkt. I detta arbete organiserades koordinatpunkterna i segment. Ett segment motsvarar en kortare sammanhängande del av körbanan, exempelvis en raksträcka eller en sväng, och innehåller en ordnad lista av målpunkter. Uppdelning gjorde det möjligt att bygga upp flera färdvägar utan att manuellt definiera varje komplett rutt, eftersom samma segment kunde återanvändas i flera rutter. En översikt över segmentens placering på testbädden visas i bilaga A, figur A.1.

Ett segment S_i beskrevs som en sekvens av punkter $S_i = (p_{i,1}, p_{i,2}, \dots, p_{i,n})$, där varje punkt $p_{i,k} = (x_{i,k}, y_{i,k})$ gavs i Gulliviews globala koordinatsystem. Punkterna ordnades enligt robotens körriktning, så att den följde $p_{i,1}, p_{i,2}, \dots$ i tur och ordning enligt segmentet.

För att beskriva hur roboten fick fortsätta efter ett segment användes en riktad graf $G = (V, E)$, där V är mängden segment och E mängden tillåtna övergångar. En kant $e_{ij} = (S_i, S_j)$ anger att roboten fick fortsätta från segment S_i till segment S_j .

I implementationen definierades segmenten i `paths.py` genom datastrukturen `SEGMENTS`. Varje segment hade ett namn, exempelvis `INTERSECTION_PATH` eller `WEST_ENTRY_ROUNDABOUT` och en lista av koordinatpunkter. De tillåtna övergångarna definierades i samma fil genom `TRANSITIONS`. Varje kant innehöll dessutomfälten `area_id`, `approach`, `heading`, `movement` och `weight`, som angav vilket kontrollerat område övergången hörde till, robotens in- och utriktning samt vilken manöver som skulle utföras. Fältet `weight` användes som vikt vid val mellan flera möjliga övergångar. Vikten 0 innebar att övergången inte valdes och kunde användas för att tillfälligt stänga av övergångar under testning.

Grafen skapades av modulen `path_graph.py`, som lade in varje segment från `SEGMENTS` som en nod och varje övergång från `TRANSITIONS` som en kant, se figur 4.2. Under körning läste `go_to_goal.py` in ett startsegment, styrde roboten mot dess koordinatpunkter en i taget, och valde därefter nästa segment utifrån de utgående kanterna. För att andra delar av systemet skulle kunna använda den planerade rutten publicerade `go_to_goal.py` den återstående rutten på ROS-topicen `path` och den aktiva segmentövergången på `active_route_edge`. Detta gjorde det möjligt för manöverlogiken att avgöra vilken korsning roboten närmade sig och vilken rörelse den planerade, vilket användes för beslutslogiken för korsningspassage, se avsnitt 4.4.2.



Figur 4.2: Illustration av hur testbäddens körbara delar representeras i systemet.

4.3 Kolonnkörning

I avsnitt 3.2 beskrevs den teoretiska grunden för CACC enligt Ploegs modell. Detta avsnitt redogör för hur den teoretiska modellen och dess beslutslogik realiserades till en fungerande mjukvaruarkitektur, samt vilka praktiska anpassningar som gjordes för att hantera systemets begränsningar.

4.3.1 Mjukvaruarkitektur

Logiken för kolonnkörning delades upp i två separata klasser för att skilja den övergripande tillståndshanteringen från den rent matematiska regleringen av hastigheten. I huvudcykeln `cooperative_control.py`, uppdateras klassen `Platoon_manager` kontinuerligt. Denna klass agerar som en tillståndsmaskin och hanterar säkerhet, val av ledare och applicering av hastighetsbegränsningar.

Beräkningen av den önskade hastigheten sker i klassen `CACC_Controller`. I denna klass tillämpas Ploegs CACC-modell för att beräkna en accelerationsändring enligt ekvation 3.4, och returnerar en referenshastighet till `Platoon_manager`. Eftersom nödstoppet alltid utvärderas före CACC-regulatorn är de två lagren oberoende, och regulatorns utdata kan inte leda till kollision även om dess interna tillstånd är felaktigt.

4.3.2 Rollbestämning i uppdateringscykeln

Klassen `Platoon_manager` implementerar beslutslogiken som beskrivs i algoritm 1 och läser av den senaste positions och hastighetsinformation från GulliView. Metoden `Update()` i `Platoon_manager` avgör robotens roll genom en sekvens av kontroller. Först kontrolleras nödstoppet, genom klassen `Emergency_brake`, om nödstoppet returnerar sant sätts hastighetsgränsen till noll och CACC-logiken hoppas över. Systemet stödjer två lägen. Ett dynamiskt läge där varje robot själv söker efter en ledare, och ett läge där man i förhand kan bestämma ledaren som används vid experiment. I dynamiskt läge bestäms rollen av en ledaridentifierare och hittas ingen ledare antar roboten rollen som ledare och håller en konstant

bashastighet. Annars får den rollen som följare och regleras med CACC-regulatorn i `cacc_controller.compute()` och följer en ledare.

4.3.3 Identifiering av ledare

Identifiering av ledare sker i metoden `_select_predecessor` i `Platoon_manager`. Metoden itererar över alla giltiga AprilTags och beräknar för varje kandidat avståndet längs robotens färdriktning, d_{fram} , och avstånd i sidled, d_{sid} , enligt formlerna i avsnitt 3.2.4. Sidledsintervallet är satt till 0,8 m för att välja robotar i ungefär samma körfält. För att undvika att ledarvalet växlar fram och tillbaka vid korta kamerabortfall behåller metoden den senaste kända ledaren under en tidsperiod på 1,0 s innan den förkastas. När en ny ledare väljs, anropas `cacc_controller.reset()`. Detta nollställer regulatorns interna tillstånd så att kvarvarande tillståndsvärden inte påverkar regleringen mot den nya ledaren.

4.3.4 Implementation av CACC-regulatorn

Klassen `cacc_controller` tillämpar avståndsregleringen för kolonnkörning enligt Ploegs CACC-modell, se ekvation 3.4. Vid varje uppdateringscykel används konstanter som är satta till $h = 0,8$ s, $d_0 = 0,35$ m, $k_p = 0,2$ och $k_d = 0,7$. Metoden `compute()` beräknar först önskat avstånd enligt ekvation 3.2. Därefter beräknas en accelerationsändring enligt ekvation 3.4. Med accelerationsändringen Δa beräknar `compute()` en referenshastighet, enligt

$$v_{\text{ref}} = v_{\text{ego}} + \Delta a \Delta t. \quad (4.1)$$

och returnerar referenshastigheten v_{ref} till `Platoon_manager`. Tillstånden för framförvarande robot hämtas från de positions- och hastighetsuppdateringar som GulliView returnerar. Eftersom GulliView inte mäter acceleration direkt uppskattas a_{pred} numeriskt som differensen mellan två på varandra följande hastighetsestimat dividerat med tidsdifferensen. För att minska påverkan från mätbrus dämpas snabba variationer i accelerationen genom ett lågpasfilter innan den används i regulatorn.

4.4 Korsningskoordinering

I avsnitt 3.3 beskrevs den metod som används för korsningskoordinering. Där definierades både den gruppbaseade VTL-logiken för korsning 1 och 2 samt reservationslogiken för korsning 3. I detta avsnitt beskrevs hur dessa modeller realiserades i mjukvarusystemet.

4.4.1 Mjukvaruarkitektur

Korsningslogiken implementerades som en separat modul. På så sätt kunde korsningslogiken utvecklas och testas separat. Modulen delades upp i flera filer med avgränsade ansvarsområden. Den geometriska beskrivningen av korsningen

placerades i `vtl_map.py`. Där skapades de tre korsningsobjekten `INTERSECTION_1`, `INTERSECTION_2` och `INTERSECTION_3` utifrån den zonmodell som beskrevs i avsnitt 3.3.3. Konfigurationsvärden, såsom signaltider, kommunikationstidsgränser och signalgrupper samlades i `vtl_config.py`. Datatyper för korsningstillstånd, rörelser och signaler definierades i `vtl_types.py`. Medlemskapsbilden byggdes i `vtl_membership.py`, medan den övergripande beslutslogiken placerades i `vtl_controller.py`.

Huvudklassen i modulen var `VTLController`. I `cooperative_control.py` skapades en separat instans av denna klass för varje korsning i evalueringsmiljön. Varje instans fick sin egen information om respektive korsning men delade samma åtkomst till robotpositioner, UDP-kommunikation och `ros_topic_handler.py`. Detta gjorde att samma klass kunde användas för alla korsningar oavsett vilken logik applicerades VTL-eller reservationslogik.

4.4.2 Uppdateringsflöde och lokalt korsningstillstånd

Korsningslogiken uppdaterades i huvudcykeln via `VTLController.update()`. Vid varje uppdatering hämtades först robotens position från `Gulliview` och ruttinformation från `go_to_goal.py`, se avsnitt 4.2. Ruttinformationen innehöll vilken korsning roboten var på väg mot, vilken vägarm den kom ifrån och vilken vägarm den skulle fortsätta mot. Denna information omvandlades därefter till ett `VTLIntent`-objekt som används av korsningslogiken i de efterföljande beslutsstegen. Efter att rörelsen hade tolkats från ruttinformationen klassificerades robotens position mot korsningens zoner. Detta gjordes med hjälp av `detect_spatial_phase()` som returnerade en fas av typen `VTLPhase`. Fasen kunde vara `OUTSIDE`, `APPROACHING`, `CROSSING` eller `LEAVING`. Denna fas användes sedan för att avgöra om korsningslogiken skulle reglera roboten eller den befann sig i ett läge där den skulle tillåtas fortsätta.

För att undvika instabila beslut användes två former av låsning. Den första var en låst rörelse, lagrad i `_active_intent`. Rörelsen låstes när roboten befann sig i den infartszon som motsvarade den planerade rörelsen. Därefter behölls samma rörelse så länge roboten befann sig inom korsningsområdet. Detta gjordes för att undvika att roboten skulle byta beslutsgrund under passage. Ett sådant byte kunde annars uppstå om `go_to_goal.py` bytte till nästa ruttsegment innan roboten helt lämnat korsningen. Den andra låsningen användes för att minska effekten av brus i positionsdata från `Gulliview`. Eftersom robotens uppmätta position kunde variera nära en zongräns kunde zonfasen för roboten hoppa mellan exempelvis `APPROACHING` och `LEAVING`. För VTL-logiken behölls därför infartsfasen under en kortperiod om roboten tillfälligt klassificeras i en annan fas. För reservationslogik behölls på motsvarande sätt den senaste giltiga zonfasen under en kort period. Dessa mekanismer gjorde att enstaka positionsvariationer inte direkt ändrade korsningsbeslutet.

4.4.3 Medlemskapsbild

Efter att det lokala korsningstillståndet hade byggts skickades det till övriga robotar via UDP. Utsändningen skedde i `_broadcast_membership_state()`, där robotens AprilTag-ID, aktuell korsning, zonfas, aktuell vägarm, låst rörelse och tidsstämpel packades i ett VTL-meddelande. Meddelandet gjorde det möjligt för andra robotar att avgöra om den lokala roboten var relevant för deras korsningsbeslut. Inkommande VTL-meddelanden togs emot i `bot_communication()` och lagrades i `_peer_vtl_states`. Dessa mottagna tillstånd användes tillsammans med den lokala robotens eget tillstånd för att bygga en `VTLMembershipView`. Medlemskapsbilden fungerande som en lokal sammanställning av vilka robotar som var relevanta för den aktuella korsningen.

Först användes tidsstämpeln i varje mottaget meddelande för att avgöra om informationen fortfarande var aktuell. Om det mottaget tillstånd var äldre än den tillåtna tidsgränsen betraktades informationen från den roboten som gammal och därmed kunde medlemskapsbilden inte bildas. Annars användes robotens korsning, zonfas och låst rörelse för att avgöra om roboten påverkade det aktuella korsningsbeslutet. En robot betraktades som relevant om den hörde till samma korsning och befann sig i infartszonen, konfliktzonen eller utfartszonen. Robotar som befann sig i infartszonen betraktades dessutom som väntande, eftersom de kunde påverka valet av signalgrupp eller reservationsordningen. På detta sätt behövde beslutslogiken inte tolka varje UDP-meddelande direkt, den kunde istället använda medlemskapsbilden som en filtrerad lägesbild av korsningen. Om relevant information saknades eller var gammal behandlades situationen konservativt beroende på vilken logik det var.

4.4.4 Applicering av signal

När en signal hade valts omvandlades den till en hastighetsgräns i `_apply_signal()`. Om signalen var RED sattes hastighetsgränsen till 0. Om signalen var YELLOW sattes hastigheten till bashastigheten multiplicerad med `VTL_YELLOW_SPEED_FACTOR`, som var 0,5. Om signalen var GREEN sattes hastighetsgränsen till bashastighet. Hastighetsgränsen registrerades i `ros_topic_handler.py` med korsningens namn som källa.

4.4.5 Implementation av VTL-logik

För korsning 1 och 2 användes medlemskapsbilden från föregående steg som beslutsunderlag för den gruppbaseade VTL-logiken. Signalgrupperna som definierades i 3.3 lades i `VTL_SIGNAL_GROUPS` i `vtl_config.py`. Dessa grupper användes som en gemensam konfiguration för både det fasta reservschemat och den planbaseade VTL-logiken, istället för hårdkoda separata regler för varje korsning.

Två scheduler-klasser användes. `VTLSignalScheduler` implementerade ett fast roterande schema och användes när medlemskapsbilden inte kunde bildas. `VTLProtocol1Scheduler` implementerade den planbaseade logiken med fälten

`now`, `next` och `prelim`. Som introducerades i avsnitt 3.3 inspirerades den strukturen av Protocol 1 hos Casimiro et al.[5], men anpassades till evalueringsmiljöns signalgrupper och tidsintervallen.

Vid varje uppdatering valdes en koordinator med `_protocol1_coordinator_id()`. Koordinatoren var roboten med lägst AprilTag-ID bland de kända robotarna, vilket gjorde att alla robotar kunde räkna samma koordinator utan separat förhandling. Innan `VTLProtocol1Scheduler` användes kontrollerades medlemskapsbilden med `_protocol1_readiness()`. Om den saknades användes `VTLSignalScheduler` istället.

När koordinatoren kunde använda den planbaserade logiken beräknades efterfrågan per signalgrupp genom att väntande robotars lästa rörelser jämfördes med de konfigurerade signalgrupperna. Gruppen med högst efterfrågan valdes som preliminär kandidat. Vid övergångstillståndet mellan grupperna flyttades planen fram så att `next` blev aktiv grupp och får `Green` och `prelim` blev `next`. För att minska risken för svält kunde schedulern byta till en annan väntande grupp om samma signalgrupp hade valts fler gånger i rad. Gränsen styrdes av `VTL_PROTOCOL1_MAX_CONSECUTIVE_DEMAND_SLOTS` vilket i implementationen var två. När koordinatoren hade uppdaterat planen skickades den till övriga robotar med `_broadcast_protocol1_plan()`. Mottagande robotar kontrollerade att planen kom från rätt koordinator och installerade den med `set_plan()`. Om planen blev äldre än `VTL_PROTOCOL1_PLAN_TIMEOUT_s` som sattes till 1.5 sekunder, användes den inte längre och roboten gick tillbaka till fast roterande reservschema.

4.4.6 Implementering av reservationslogik

I enlighet med reservationsprincipen som beskrevs i avsnitt 3.3 och som inspirerades av distribuerade korsningsmetoder [6] implementerades korsning 3 separat i `_get_intersection_3_signal()`. Beslutet baserades direkt på medlemskapsbilden och robotarnas aktuella positioner i zonerna. Om roboten befann sig utanför korsningen, redan var i konfliktzonen eller höll på att lämna korsningen returnerades `GREEN`. Detta gjordes för att roboten inte skulle stanna i korsningen. Om roboten däremot befann sig i infartszonen aktiverades först en kort väntetid. Denna väntetid styrdes av `VTL_INTERSECTION_3_ENTRY_HOLD_S`, som sattes till en sekund. Syftet var att roboten skulle hinna ta emot nya tillstånd från andra robotar innan den fattade sitt inträdesbeslut.

Efter väntetiden kontrollerades medlemskapsbilden. Om den inte kunde bildas på grund av gammal information gavs `RED`. Om en annan relevant robot befann sig i konfliktzonen, lämnade korsningen eller upptog den gemensamma östra vägarmen gavs också `RED`. Om fler robotar väntade samtidigt användes AprilTag-ID som prioritet. Roboten med lägre AprilTag-ID fick företräde, medan den andra väntade tills korsningen blev fri.

4.5 Rondellkoordinering

I avsnitt 3.4 beskrevs den metod som används för rondellkoordinering. Där definierades algoritmen för inkörningsbeslut samt den underliggande matematiken för att beräkna säkra tidsluckor. I detta avsnitt beskrivs hur denna teoretiska modell realiserades i mjukvarusystemet, samt vilka praktiska anpassningar som gjordes för att hantera positioneringssystemets begränsningar.

4.5.1 Mjukvaruarkitektur

För att representera rondellen och hantera kritiska beräkningar implementerades en fristående **Zone**-klass. Denna klass utformades för att erbjuda ett flexibelt och modulärt sätt att definiera rondellens fysiska utsträckning, såväl som koordinater och gränser för tillhörande infarter (*entry zones*). Genom att inkapsla denna logik underlättades bedömningen av huruvida en robot, baserat på dess globala position från Gulliview, befinner sig inuti rondellzonen.

Systemets beslutsfattande och uppdateringslogik byggdes därefter samman i huvudklassen **Roundabout**. Grundtanken bakom designen var att upprätthålla en uppdateringsmetod, som anropas kontinuerligt vid varje iteration i systemets huvudcykel. För att säkerställa hög kodkvalitet och god läsbarhet utformades `update()`-metoden för att hållas så kort och koncis som möjligt. Istället för att inkludera komplex beräkningslogik i själva uppdateringscykeln, delegerades logiken till separata hjälpmetoder med beskrivande funktionsnamn. Huvudmetoden ansvarar därmed primärt för den övergripande logiken som beskrivs i algoritm 4.

4.5.2 Beslutsflöde och inkörningslogik

Vid varje uppdatering utvärderas först robotens fysiska position i relation till rondellens definierade zon. Eftersom positionsdata från Gulliview är prestandabegränsad, fanns en risk att roboten tillfälligt skulle tolkas befinna sig utanför rondellzonen, vilket skulle leda till snabbt skiftande tillstånd. För att hantera detta implementerades en tidsbaserad låsningsmekanism via `_can_exit_roundabout()`. Genom att spåra hur länge roboten har befunnit sig utanför rondellzonen säkerställs att ett formellt utträde från rondellen genomförs först när ett bestämt tidsmarginal har passerats.

Sen fastställs vilken infart roboten är på väg mot. Detta görs genom `_find_closest_entry_zone()`, som itererar över de konfigurerade infartszonerna och beräknar den euklidiska distansen från robotens aktuella position för att identifiera den närmaste infartszonen.

Med infarten identifierad sker själva säkerhetsbedömningen för inkörning. Systemet filtrerar fram vilka andra kända robotar som fysiskt befinner sig inne i rondellen, och beräknar därefter ankomsttiden för var och en av dessa till den aktuella infarten. Detta sker i två steg. Först används `_calculate_circular_distance()`

för att fastställa avståndet längs med rondellens krökning. Metoden implementerar den geometriska modell som beskrevs i avsnitt 3.4. Här realiseras även den tidigare beskrivna intervallkorrigeringen ($\Delta\theta_{korr}$), vilket i mjukvaran används för att ge robotar som har passerat infarten ett negativt avstånd. Om en robot i rondellen precis har passerat infarten (ett litet negativt avstånd) sätts den beräknade tidsluckan explicit till noll. Målet med detta är att förhindra den väntande roboten från att omedelbart accelerera och riskera en sidokollision med roboten som just körde förbi infarten. Först när den passerande roboten har kommit tillräckligt långt bort (definierat av gränsvärdet `ROUNDABOUT_PASSED_ENTRY_DISTANCE`) upphävs spärren och inkörningen kan ske säkert.

När det cirkulära avståndet är fastställt, evalueras avståndet gentemot robotens bashastighet i funktionen `_time_to_entry_zone()`. Resultatet blir en uppskattad tid tills roboten inuti rondellen korsar infartszonen. Denna skattade tid utgör slutligen kärnan i inkörningsbeslutet, som utförs av `_should_enter_roundabout()`. Om avståndet i tid är mindre än säkerhetsmarginalen (`SAFE_TIME_THRESHOLD`) betraktas inkörningen som osäker och robotens hastighet sätts till noll. Om tidsluckan däremot överskrider `SAFE_TIME_THRESHOLD` godkänns manövern, systemtillståndet uppdateras till att roboten kör inne i rondellen, och den tillåts köra in i rondellen.

5

Evalueringsplan

Detta kapitel beskriver hur de metoder som presenterats i kapitel 3 evalueras. Evalueringen utgår från två forskningsfrågor som behandlar säkerhet och framkomlighet. Evalueringen syftar till att undersöka i vilken utsträckning de implementerade manövrerna uppfyller respektive krav. Kapitlet presenterar först de forskningsfrågor som ligger till grund för evalueringen, därefter den övergripande evalueringsstrategin, den miljö i vilken experimenten genomförs samt de kriterier och experiment som utformats för nödstopp, kolonnkörning, rondellkoordinering och korsningskoordinering.

5.1 Forskningsfrågor

För att kunna evaluera de implementerade manövrerna formuleras två övergripande forskningsfrågor.

F-1 I vilken utsträckning uppfyller de implementerade manövrerna säkerhetskraven i respektive trafikscenario?

F-2 I vilken utsträckning uppfyller de implementerade manövrerna kraven om framkomlighet i respektive trafikscenario?

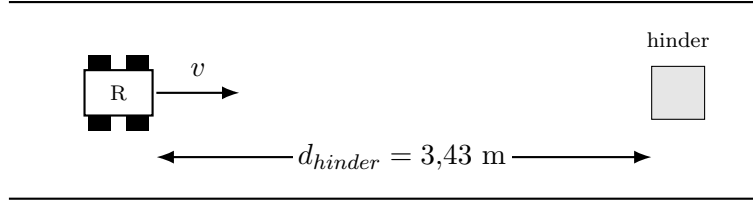
5.2 Evalueringsmiljö

Alla experiment utförs i ett laboratorium på Chalmers, utvecklad för nedskalade autonoma fordon. Vid test används två WifiBots med Apriltag-ID 4 och 5. I laboratoriet används lokaliseringssystemet GulliView, och specifikt för detta projektet körs version GulliView Matei.

5.3 Testfall

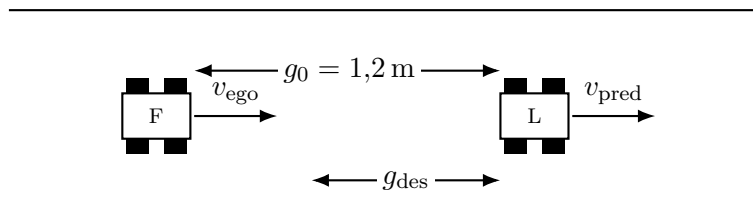
Testfall 1: Nödstopp mot statiskt hinder Testfallet i figur 5.1 testar att roboten stannar innan den kolliderar mot ett stationärt hinder. En WiFiBot körs längs en rak bana mot en stationär kartonglåda med en AprilTag som placeras

3,43 meter framför WifiBoten. Roboten ska enligt kraven i avsnitt 3.1.1 stanna helt innan den kolliderar med lådan och sedan inte starta igen.



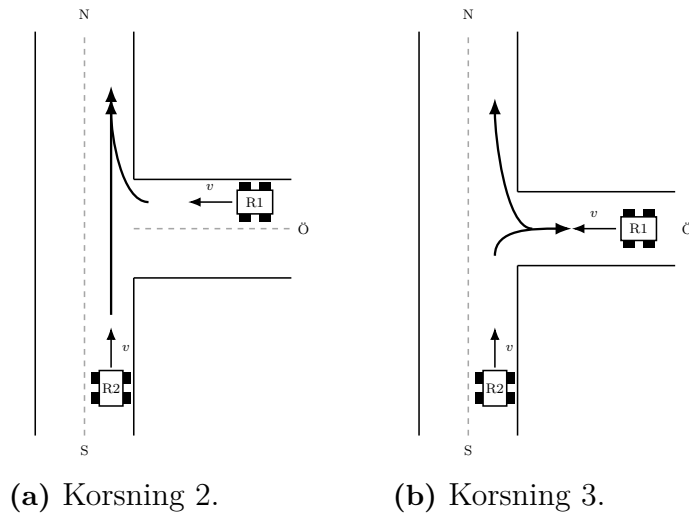
Figur 5.1: Testfall 1: Robot R körs mot ett stationärt hinder och förväntas stanna innan kollision.

Testfall 2: Kolonnkörning med ledare och följare Testfallet i figur 5.2 testar att följaren F håller ett önskat avstånd till framförvarande robot L. Två robotar färdas efter varandra på samma bana. Framförvarande robot styrs med en given hastighet v_{pred} , medan följaren anpassar sin hastighet v_{ego} utifrån avvikelsen mellan det uppmätta avståndet g och det önskade avståndet g_{des} , enligt avsnitt 3.2.2. Scenariot kopplas till kraven för kolonnkörning i avsnitt 3.2.1.



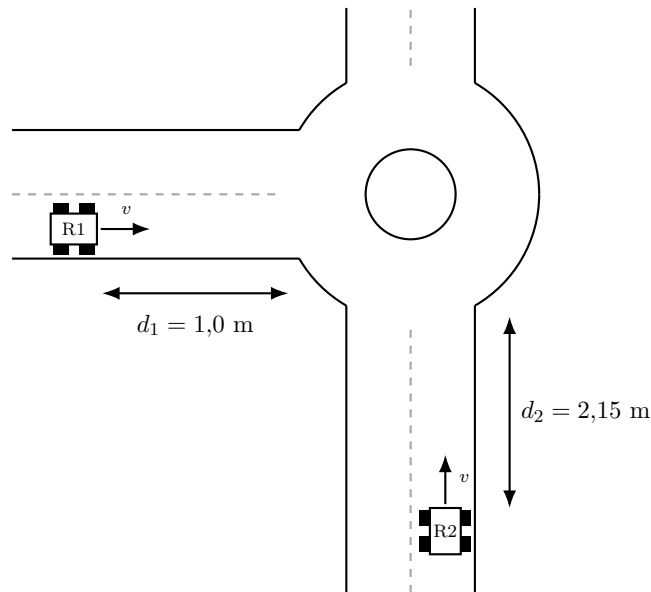
Figur 5.2: Testfall 2: Följaren F ska anpassa sin hastighet för att hålla det önskade avståndet g_{des} till den framförvarande roboten L. g_0 är initialavståndet.

Testfall 3: Korsning med två robotar Testfallet i figur 5.3 testar att korsningslogiken hanterar två robotar som närmar sig samma korsningsområde från olika riktningar med samma konstanta hastighet v . Robotarna ska passera enligt deras tilldelade rutter. Testfallet består av två scenarier. I korsning 2 testas den gruppbaseade VTL-logiken där två robotar körs enligt figur 5.3a. I korsning 3 testas reservationslogiken där två robotar kör enligt figur 5.3b. Scenarierna kopplas till kraven för korsningskoordinering i avsnitt 3.3.



Figur 5.3: Testfall 3 för korsningspassage. I (a) korsning 2 kör Robot 1 från öst mot norr och Robot 2 från syd mot norr. I (b) korsning 3 kör Robot 1 från öst mot norr och Robot 2 från syd mot öst.

Testfall 4: Synkroniserad ankomst till rondell Testfallet i figur 5.4 testar att rondellooken hanterar två robotar som anländer samtidigt från olika infarter. Två robotar ankommer till rondellen med samma konstanta hastighet v och försöker passera genom rondellen. Robotarna placeras enligt figur 5.4. Scenariot kopplas till kraven för rondellkoordinering i avsnitt 3.4.1.



Figur 5.4: Testfall 4: Två robotar anländer till rondellen samtidigt och förväntas passera utan kollision. Robotarnas initiala avstånd till rondellens infart betecknas med d_1 för R1 (1,0 m) och d_2 för R2 (2,15 m) och används för att sätta upp testfallet.

5.4 Evalueringskriterier

Säkerhet och framkomlighet mäts med ett antal gemensamma kriterier. Dessa används inte i varje experiment, utan väljs utifrån vilken manöver som utförs.

Det minsta avståndet mellan två robotar eller mellan robot och hinder betecknas som d_{min} och beräknas som det euklidiska avståndet mellan objektens mittpunkter enligt

$$d_{min} = \|d(t)\| = \sqrt{(x_1(t) - x_2(t))^2 + (y_1(t) - y_2(t))^2} \quad (5.1)$$

Måttet används som primär säkerhetsindikator vid samtliga manövrer. Vid arbeten inom kooperativa manövrer är tid till kollision (TTC) det vanligaste kriteriet. I detta projekt används istället det minsta avståndet mellan robotar på grund av att den använda versionen `gulliview_matei` inte kan producera stabil nog data för position och riktning. Detta gör så att TTC värdena blir opålitliga vid de hastigheter som används i experimenten. Därför används d_{min} och kollisionsflagga istället som säkerhetsmått.

För kolonnkörning används avståndsfelet g_{err} , som är skillnaden mellan det uppmätta avståndet g till framförvarande robot och det önskade avståndet g_{des} definierat i avsnitt 3.2.2. Ett lågt g_{err} innebär att följaren håller önskad marginal.

För kolonnkörning används även en strängstabilitetskvot för att beskriva hur hastighetsändringen överförs till följaren. Kvoten jämför amplituden i följarens hastighetsavvikelse med amplituden i ledarens hastighetssteg:

$$\text{kvot} = \frac{A_F}{A_L}, \quad (5.2)$$

Om kvoten < 1 dämpas hastighetsändringen, vilket ger jämnare kolonnkörning. Om kvoten ≈ 1 följer följaren ledarens förändring ungefär lika mycket. Om kvoten > 1 förstärks hastighetsändringen, vilket kan ge ryckigare körning och större avståndsfel längre bak i kolonnen.

För korsning och rondell används passagetid ΔT , som definieras som tiden från att roboten når en startpunkt tills den når en slutpunkt:

$$\Delta T = T_{slut} - T_{start} \quad (5.3)$$

5.5 Experiment

Experimenten genomförs för att besvara evalueringsens forskningsfrågor och utgår från de definierade testfallen. Varje experiment innehåller en kort beskrivning av syftet, vilka variabler som ändras, hur många upprepningar som genomförs, vilken hypotes som testas samt ett genomförande som beskriver hur datainsamlingen och analysen utförs. Eftersom varje hastighet endast testas ett begränsat antal gånger bör resultaten främst ses som experimentella indikationer på systemets funktion i evalueringsmiljön, snarare än som säkerställda prestandagränser.

Experiment 1: Effekt av hastighet på nödstopp Detta experiment använder Testfall 1 för att undersöka forskningsfråga F-1 gällande säkerhet och evaluerar kraven för nödstopp, se avsnitt 3.1.1. Variabeln är robotens hastighet $v \in \{0,4; 0,8; 1,1; 1,2; 1,6\}$ m/s, och varje hastighet körs fem gånger.

Hypotesen är att WiFiBoten stannar helt innan kontakt vid de lägre hastigheterna, vilket motsvarar att kravet för kollisionssavvärjning E2 uppfylls. Vid de högre hastigheterna förväntas prestandabegränsningen i GulliView medföra att bromskommandot inte hinner verkställas i tid, vilket resulterar i en kollision. Det som mäts är kollisionsstatus och avståndet till hindret när roboten har stannat.

Experimentet genomförs genom att GulliView och roboten startas enligt dokumentationen för systemet. En kartonglåda med en AprilTag placeras på avståndet d_0 framför roboten, med AprilTagen riktad mot den. Det säkerställs att inget blockerar kamerorna i taket. Roboten startas och körs längs en rak bana mot lådan, efter första körning kontrolleras att loggfunktionen för **emergency_brake** är aktiverad, annars starta den. Efter varje körning antecknas eventuell kollision, loggdata exporteras och stegen upprepas för respektive hastighet v .

Experiment 2: Konvergens vid konstant ledarhastighet Detta experiment använder Testfall 2 för att undersöka forskningsfrågorna F-1 och F-2 för kolonnkörningsmanövern och evaluerar krav P1 och P2, se avsnitt 3.2.1. Variabeln är ledarhastigheten $v_{\text{ledare}} \in \{0,3; 0,5; 0,7; 0,9\}$ m/s, och varje hastighet körs minst tre gånger. Tidsavståndet är $h = 0,8$ s och stillastående säkerhetsavstånd är $d_0 = 0,35$ m.

Hypotesen är att avståndsfelet g_{err} konvergerar mot noll inom 15 s och håller sig därefter inom ± 10 cm, vilket motsvarar att framkomlighetskravet P2 uppfylls. Nödstoppet utlöses aldrig under körningen, vilket motsvarar att säkerhetskravet P1 uppfylls. Kriterierna är $g_{\text{err}}(t)$, konvergenstid, stationär avvikelse, d_{min} och nödstoppsflagga.

Experimentet genomförs genom att starta GulliView och robotarna enligt systemets dokumentation. Båda robotarna placeras på samma bana med ett initialt avstånd mellan ledarens bakre kant och följarens främre kant på $g_0 = 1,2$ m, och med bashastigheten v_{ledare} . Robotarna startas samtidigt och körs i 30 s efter att följaren visuellt konvergerat mot önskat avstånd. Under hela processen loggas position, hastighet, gap och nödstoppsstatus. Efter varje försök stoppas båda robotarna och loggdata exporteras. Stegen upprepas för respektive ledarhastighet v_{ledare} .

Experiment 3: Respons på hastighetsförändring hos ledare Detta experiment använder Testfall 2 för att undersöka forskningsfrågorna F-1 och F-2 för kolonnkörningsmanövern och evaluerar krav P1 och P2, se avsnitt 3.2.1. Bashastigheterna för robotarna är $v_{\text{bas}} = 0,5$ m/s och ledarens nya hastighet $v_{\text{ny}} \in \{0,1; 0,3; 0,7; 0,9\}$ m/s. Varje hastighetsändring körs minst tre gånger.

Hypotesen är att amplituden av följarens hastighetsavvikelse inte överstiger ledarens, det vill säga att kolonnkörningen är strängstabil och uppfyller framkomlighetskravet P2, samt att $d_{\min}$ aldrig understiger d_0 och nödstoppet aldrig triggas, vilket motsvarar att säkerhetskravet P1 uppfylls. Kriterierna är $v_{\text{ledare}}(t)$, $v_{\text{följare}}(t)$, $g_{\text{err}}(t)$, $d_{\min}$, nödstoppsflagga och återhämtningstid till stationärt tillstånd.

Experimentet genomförs genom att starta GulliView och robotarna enligt systemets dokumentation. Båda robotarna placeras på samma bana med ett initialt avstånd från fram till bak på $g_0 = 1,2$ m, och med bashastigheten $v_b = 0,5$ m/s. Robotarna startas samtidigt och får konvergera under 20 s. Därefter ändras ledarens hastighet till v_{ny} via ROS-parametern `/leader_cruise_speed` och körningen pågår i 20 s efter ändringen. Båda robotarna stoppas, loggdata exporteras, och stegen upprepas för respektive v_{ny} .

Experiment 4: Passering genom korsning 2 vid synkroniserad ankomst och varierade hastigheter Detta experiment använder Testfall 3 för att undersöka forskningsfrågor F-1 och F-2 för korsningsmanövern. Experimentet evaluerar den gruppbaseade VTL-logiken i korsning 2 enligt kraven K1 och K2, se avsnitt 3.3.1. Robot-1 kör från Ö mot N och Robot-2 kör från S mot N. Se figur 5.3a för illustration. Hastigheten för båda robotarna är $v \in \{0,3; 0,5; 0,6; 0,7\}$ m/s. Varje hastighet körs fem gånger.

Hypotesen är att både robotarna vid de lägre hastigheterna kan passera genom korsning 2 utan att en kollision sker. Robotarna ska även uppnå framkomlighet inom ändlig tid vid de lägre hastigheterna. Detta motsvarar att säkerhetskravet K1 och framkomlighetskravet K2 uppfylls. Vid de högre hastigheterna förväntas systemets marginaler minska, eftersom kortare reaktionstid, positionsbrus från GulliView och fördröjningar i hastighetsändringar kan göra att den gruppbaseade VTL-logiken inte hinner reglera robotarna tillräckligt tidigt. Om en kollision sker uppfylls inte säkerhetskravet K1. Om en robot stannar eller blockeras under försöket uppfylls inte framkomlighetskravet K2. Kriterierna för evalueringen är kollisionsstatus, minsta avstånd mellan robotarna $d_{\min}$ och robotarnas passagetid genom korsningen ΔT .

Experimentet genomförs genom att starta GulliView och robotarna enligt systemets dokumentation. Robot-1 placeras cirka 0,3 m före startsegmentet `INTERSECTION_PATH`, vilket motsvarar rörelsen från öster mot norr. Robot-2 placeras vid början av startsegmentet `TURN_FLOOR`, vilket motsvarar rörelse från söder mot norr, se bilaga A figur A.1. Robotarna konfigureras med fixerade rutter genom korsning 2 och startas samtidigt. Körningen pågår tills båda robotarna har lämnat korsningen. Därefter stoppas robotarna, loggdata exporteras och försöket upprepas fem gånger för varje hastighet.

Experiment 5: Passering genom korsning 3 vid synkroniserad ankomst och varierad hastighet Detta experiment använder Testfall 3 för att undersöka forskningsfrågorna F-1 och F-2 för korsningsmanövern. Experimentet evaluerar

reservationslogiken i korsning 3 enligt kraven K1 och K2, se avsnitt 3.3.1. Robot-1 kör från Ö mot N och Robot-2 kör från S mot Ö, se figur 5.3b för illustration. Hastigheten för båda robotarna är $v \in \{0,3; 0,5; 0,6; 0,7; 0,9; 1,0; 1,1\}$ m/s. Varje hastighet körs fem gånger.

Hypotesen är att både robotarna kan passera genom korsning 3 utan att en kollision sker inom det testade hastighetsintervallet. Robotarna ska även uppnå framkomlighet inom ändlig tid. Detta motsvarar att säkerhetskravet K1 och framkomlighetskravet K2 uppfylls. Eftersom korsning 3 använder reservationslogik förväntas endast en robot i taget få tillträde till konfliktszonen och den gemensamma östra vägarmen. Därför förväntas risken för kollision vara lägre än i korsning 2 vid höga hastigheter. Vid de höga hastigheterna kan positionsbrus från GulliView och fördröjningar i hastighetsändringar minskar säkerhetsmarginalen. Om en kollision sker uppfylls inte säkerhetskravet K1. Om en robot stannar eller blockeras under försöket uppfylls inte framkomlighetskravet K2. Kriterierna för evalueringen är kollisionsstatus, minsta avstånd mellan robotarna d_{min} och robotarnas passagetid genom korsningen ΔT .

Experimentet genomförs genom att starta GulliView och robotarna enligt systemets dokumentation. Robot-1 placeras cirka 0,3 m före startsegmentet WEST_EXIT_ROUNDABOUT, vilket motsvarar rörelsen från öster mot norr. Robot-2 placeras vid början av startsegmentet STRAIGHT_WHITEBOARD, vilket motsvarar rörelsen från söder till öster, se bilaga A, figur A.1. Robotarna konfigureras med fixerade rutter genom korsning 3 och startas samtidigt. Körningen pågår tills båda robotarna har lämnat korsningen. Därefter stoppas robotarna, loggdata exporteras och försöket upprepas fem gånger för varje hastighet.

Experiment 6: Rondellpassage vid synkroniserad ankomst och varierad hastighet Detta experiment använder Testfall 4 för att evaluera hur robotarnas hastighet påverkar säkerhet och framkomlighet under rondellmanövern. Detta experiment undersöker även forskningsfrågorna F-1 och F-2 för rondellmanövern och evaluerar kraven R1 och R2, se avsnitt 3.4.1. Robotarnas hastighet är $v \in \{0,4; 0,5; 0,6; 0,7\}$ m/s, och varje hastighet körs fem gånger.

Hypotesen är att båda robotarna vid de lägre hastigheterna kan passera genom rondellen utan att en kollision sker. Robotarna ska även ha en framkomlighet inom ändlig tid vid de lägre hastigheterna. Detta motsvarar att kravet för säkerhetsbedömning R1 och kravet för framkomlighet inom ändlig tid R2 uppfylls. Vid de högre hastigheterna förväntas GulliViews prestandabegränsningar medföra att en kollision kommer att ske. Om en kollision sker blir följderna att en framkomlighet inom ändlig tid inte uppnås. Kriterierna för evalueringen är kollisionsstatus, avståndet mellan robotarna och robotarnas passagetid genom rondellen. Passagetiden mäts från att roboten kör in i rondellzonen till att den kör ut ur rondellzonen.

Experimentet genomförs genom att systemet, inklusive robotarna och GulliView,

startas enligt dokumentationen. Respektive robot placeras sedan på sin startposition enligt figuren 5.4. Därefter startas båda robotarna samtidigt. Körningen pågår tills robotarna har passerat helt genom rondellen, varefter programmet stängs av och stegen upprepas för de angivna hastigheterna.

6

Resultat och diskussion

Detta kapitel redovisar att de implementerade manövrerna i huvudsak uppfyller säkerhets- och framkomlighetskraven inom de testade hastighetsintervallen. För de flesta manövrer ligger gränsen mellan 0,5 och 0,8 m/s, medan reservationslogiken förblir kollisionsfri upp till 1,1 m/s. Kapitlet är uppdelat efter manövertyp. För varje manöver bedöms resultat utifrån de krav, hypoteser och testfall som definierades i kapitel 3 och kapitel 5.

6.1 Nödstopp

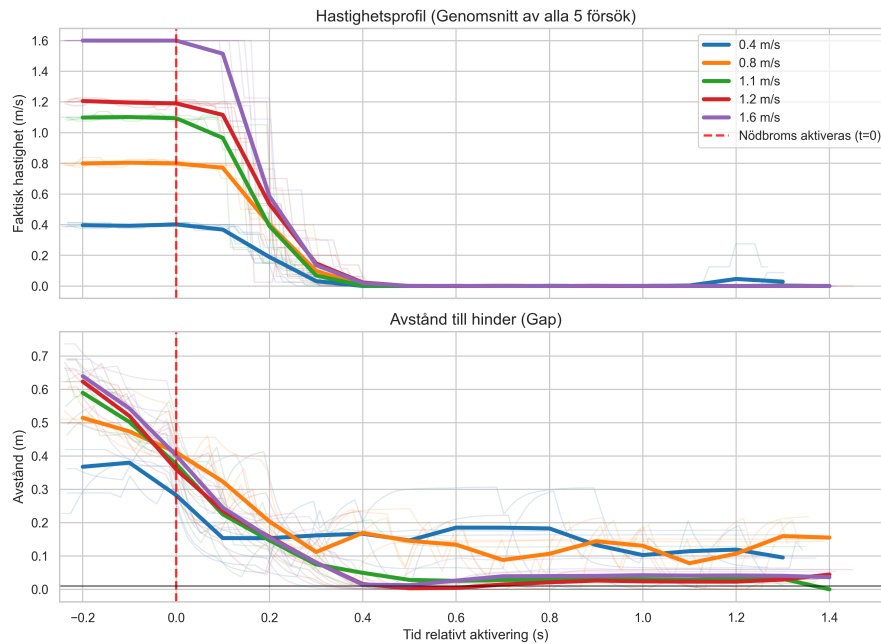
I avsnitt 5.3 beskrivs experiment 1. I tabell 6.1 och figur 6.1 presenteras resultat för experimentet. Experimentet testar ifall roboten lyckas stanna innan den kolliderar med kartongen som simulerar en annan robot i experimentet. Tabell 6.1 visar ifall den kolliderade med hindret vid de olika hastigheterna. Figur 6.1 visar hastigheten och avståndet till hindret när nödstoppet aktiverades.

Tabell 6.1: Kollisionsstatus vid nödstopp över fem försök per hastighet.

	0,4 m/s	0,8 m/s	1,1 m/s	1,2 m/s	1,6 m/s
Försök 1	S	S	S	K	K
Försök 2	S	S	S	S	K
Försök 3	S	S	K	K	K
Försök 4	S	S	K	S	S
Försök 5	S	S	S	K	K

Notera: 'S' indikerar fullt stopp innan hindret, 'K' indikerar kollision med hindret.

Figur 6.1 visar vad som skedde när nödbroms aktiverades vid de olika hastigheterna och avståndet till hindret från roboten medan den bromsade in.



Figur 6.1: Figuren visar hastigheten för robotarna och avståndet till hindret vid de olika försöken. De mörka linjerna är ett medelvärde av de fem olika försök som syns i bakgrunden.

Resultaten från tabell 6.1 visar att för hastigheterna 0,4 m/s och 0,8 m/s uppfylls kravet E2, då roboten stannade innan hindret vid samtliga försök. Figur 6.1 visar att detta stämmer där en säkerhetsmarginal på 0,1 till 0,2 m finns för hastigheterna 0,4 och 0,8 m/s.

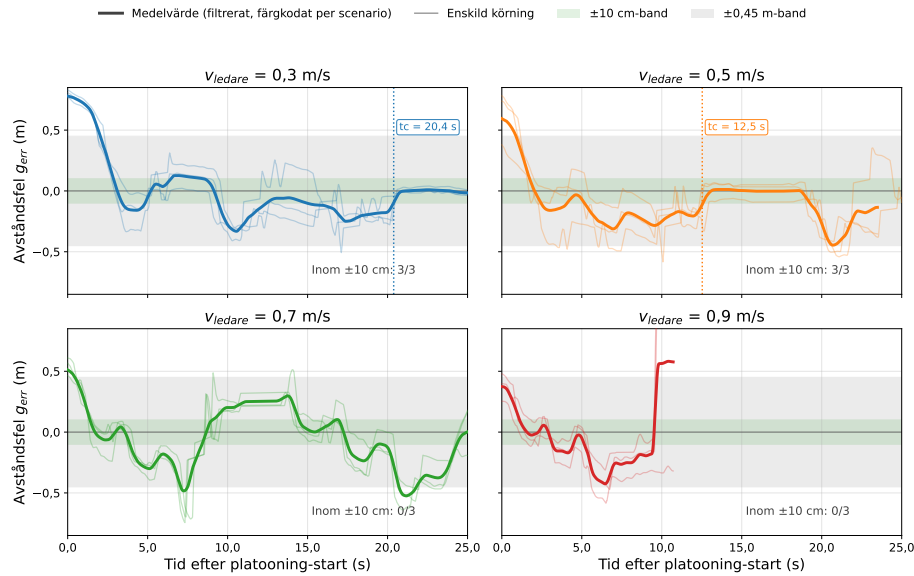
Vid hastigheter högre än 1,1 m/s resulterade majoriteten av försöken i kollision. Figur 6.1 indikerar att bromskommandot initieras vid ungefär samma avstånd från hindret oavsett hastighet (undantaget 0,4 m/s). Detta verifierar experimentets hypotes. Att systemlatensen och prestandabegränsningen i GulliView medför att bromskommandot inte hinner verkställas innan kollision inträffar när hastigheten ökar.

Gällande krav E3 på stabilitet syns variationer i figur 6.1 där det uppmätta avståndet varierar under inbromsningen. Detta är ett resultat av positionosäkerhet i GulliView. Loggdata visar emellertid att nödstoppet inte oscillerar mellan aktivt och inaktivt tillstånd till följd av bruset, vilket innebär att E3 uppfylls. Analys av `emergency_brake`-loggen bekräftar även att nödstoppet erhöll högst prioritet och omedelbart åsidosatte övrig logik vid detekterad fara.

6.2 Kolonnkörning

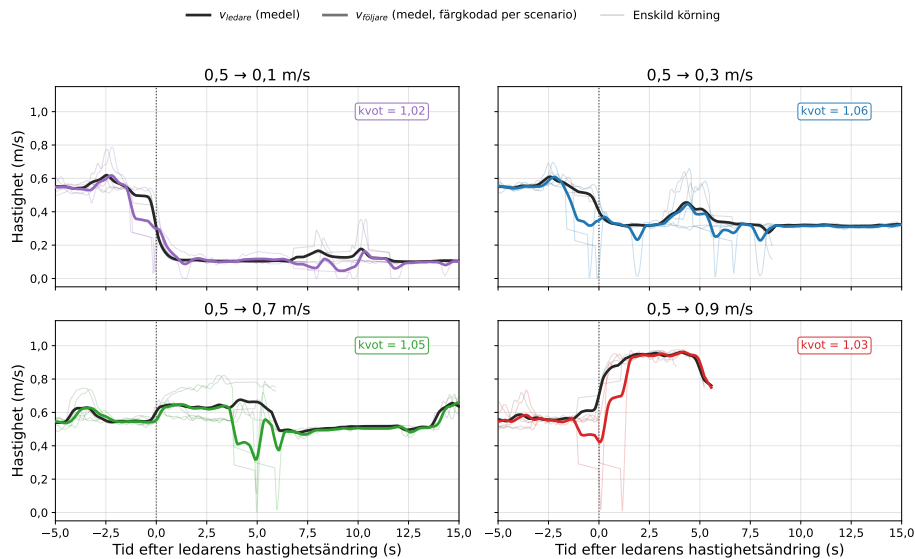
I avsnitt 5.5 beskrivs experimenten för kolonnkörning. Experiment 2 mäter om följaren konvergerar mot det önskade avståndet vid konstant ledarhastighet.

Figur 6.2 presenterar avståndsfelet $g_{err}(t)$ för fyra ledarhastigheter. Avståndsfelet beräknas som skillnaden mellan uppmätt avstånd och önskat avstånd. En körning räknas som konvergerad när $|g_{err}| \leq 0,10$ m under minst 2,5 s. Vid 0,3 och 0,5 m/s uppfyllde alla körningar konvergenstkriteriet. Medianvärdet för konvergenstid t_c var 20,4 s vid 0,3 m/s och 12,5 s vid 0,5 m/s. Vid 0,7 m/s och 0,9 m/s nådde ingen körning felbandet under minst 2,5 s. Resultaten visar därmed att regulatören kan stabilisera avståndet vid lägre hastigheter, men att avståndsregleringen blir mindre robust vid högre ledarhastigheter. De högre hastigheterna uppvisar större variationer i avståndsfelet men inget av försöken utlöste nödstoppet.



Figur 6.2: Avståndsfelet $g_{err}(t)$ från experiment 2 för fyra ledarhastigheter. Tjocka linjer är medelvärde, tunna linjer är enskilda körningar. Det gröna bandet markerar hypotesens ± 10 cm. Prickad vertikal linje är medianen av konvergenstiden t_c .

Experiment 3 undersöker följarens hastighetsrespons när ledaren ändrar hastighet. Figur 6.3 visar att följaren reagerar på de olika hastighetsförändringarna. För att minska påverkan från tillfälliga mätspikar används en 95-procentsbaserad filtrering. För strängstabilitetsknoten var medianvärdena 1,02 vid $0,5 \rightarrow 0,1$ m/s, 1,06 vid $0,5 \rightarrow 0,3$ m/s, 1,05 vid $0,5 \rightarrow 0,7$ m/s och 1,03 vid $0,5 \rightarrow 0,9$ m/s. Värdena nära 1 betyder att följarens hastighetsavvikelse är ungefär lika stor som ledarens. De tunna kurvorna visar dock att enskilda körningar fortfarande innehåller tillfälliga spikar. Även om resultaten visar på att regulatören fungerar i praktiken, så innebär det inte att strikt strängstabilitet kan garanteras för alla körningar.



Figur 6.3: Hastigheter från experiment 3. Svart linje är ledarens hastighet och färgad linje är följarens. Tjocka linjer är medelvärde, tunna linjer är enskilda körningar. Vertikal streckad linje markerar tidpunkten då ledarens hastighet ändrades. Rutan i övre högra hörnet visar medianen av strängstabilitetskvoten $\hat{v}_{\text{följare}}/\hat{v}_{\text{ledare}}$.

Resultaten visar att säkerhetskravet P1 uppfylls i samtliga körningar, eftersom g aldrig understeg 0,35 m och nödstoppet aldrig utlöstes. Framkomlighetskravet P2 uppfylls vid 0,3 och 0,5 m/s, där avståndsfelet konvergerar mot felbandet, men vid 0,7 och 0,9 m/s uppfylls inte kravet. Tillfälliga avvikelser i enskilda körningar gör att P2 inte kan anses verifierat för dessa hastigheter. Resultaten visar att kolonnkörning är stabil vid låga hastigheter men tappar robusthet när hastigheten ökar.

Detta beror på att CACC-regulatorns framkoppling bygger på att ledarens hastighet och acceleration kan mätas med god noggrannhet. De individuella kurvorna i figur 6.2 och 6.3 visar hopp i mätdata från GulliView, ett brus som regulatorn förstärker vid högre hastigheter och innebär att regulatorn reagerar på störningar till skillnad från ledarens faktiska rörelse. En orsak kan också vara att regulatorns parametrar inte är anpassade för nedskalade fordon eftersom Ploegs modell är utvecklad för fullskaliga fordon. Att strängstabilitetskvoten ligger strax över 1 betyder att hastigheten förstärks, men förstärkningen är så liten att mätbruset gör det osäkert om den verkliga kvoten ligger över eller under 1. Kolonnkörning fungerar därmed vid låga hastigheter upp till 0,5 m/s, men mätkvaliteten i GulliView begränsar hur väl manövern kan utföras.

6.3 Korsningskoordinering

I detta avsnitt presenteras resultaten från experiment 4 för korsning 2 med VTL-logik och experiment 5 för korsning 3 med reservationslogik. Resultaten kopplas till säkerhetskravet K1 och framkomlighetskravet K2, se kapitel 3.3.1.

Säkerheten utvärderas utifrån kollisionstatus och minsta avstånd mellan robotarna, d_{min} , medan framkomlighet utvärderas med hjälp av passagetiden ΔT .

I tabell 6.2 presenteras resultaten för kollision under experimenten. En kollision i detta avsnitt innebär inte en fysisk kollision mellan robotar utan att nödstoppet aktiverades. För korsning 2 genomfördes samtliga försök utan kollision vid hastigheterna $v \in \{0,3; 0,5\}$ m/s. Vid $v = 0,6$ m/s lyckades endast ett av fem försök, medan samtliga försök vid $v = 0,7$ m/s resulterade i kollision. Detta innebär att säkerhetskravet K1 uppfylls för VTL-logiken vid de lägre hastigheterna, men inte vid de högre hastigheterna.

För korsning 3 observerades däremot inga kollisioner i de redovisade försöken. Vid hastigheterna $v \in \{0,3; 0,5; 0,6; 0,7; 0,9\}$ m/s genomfördes samtliga fem försök utan kollision. För $v = 1,0$ m/s och $v = 1,1$ m/s genomfördes endast ett försök per hastighet, vilket innebär att försöket inte är lika robust. Resultaten indikerar att reservationslogiken i korsning 3 är mer robust än VTL-logiken i korsning 2 för de testade hastigheterna.

Tabell 6.2: Kollisionstatus för korsning 2 och korsning 3 över försöken per initialhastighet.

Korsning 2

	0,3 m/s	0,5 m/s	0,6 m/s	0,7 m/s
Försök 1	S	S	K	K
Försök 2	S	S	K	K
Försök 3	S	S	K	K
Försök 4	S	S	S	K
Försök 5	S	S	K	K

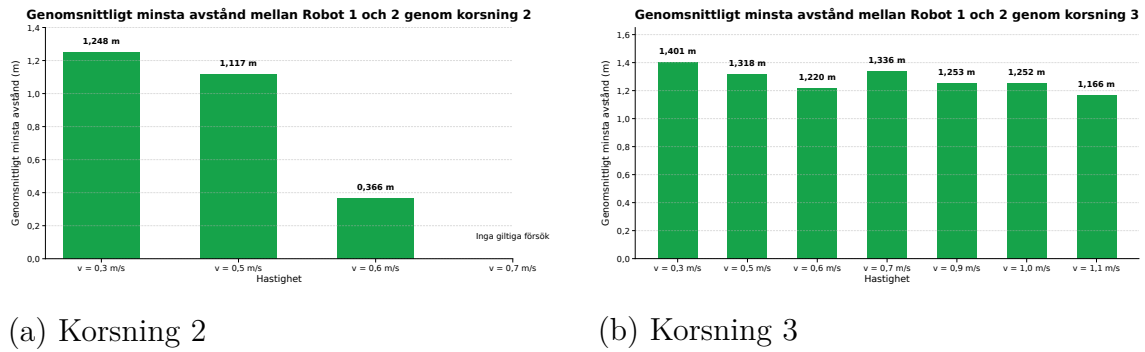
Korsning 3

	0,3 m/s	0,5 m/s	0,6 m/s	0,7 m/s	0,9 m/s	1,0 m/s	1,1 m/s
Försök 1	S	S	S	S	S	S	S
Försök 2	S	S	S	S	S	-	-
Försök 3	S	S	S	S	S	-	-
Försök 4	S	S	S	S	S	-	-
Försök 5	S	S	S	S	S	-	-

Notera: 'S' indikerar ett kollisionsfritt försök, 'K' indikerar kollision och '-' indikerar att inget försök genomfördes.

Figur 6.4 visar det genomsnittliga minsta avståndet mellan robotarna under korsningspassagen. För korsning 2 minskade d_{min} med ökande hastighet. Vid $v = 0,3$ m/s och $v = 0,5$ m/s uppvisas relativt stora säkerhetsmarginaler, medan avståndet minskade kraftigt vid $v = 0,6$ m/s där endast ett försök lyckades. Vid $v = 0,7$ m/s saknas giltiga kollisionsfria försök.

6. Resultat och diskussion

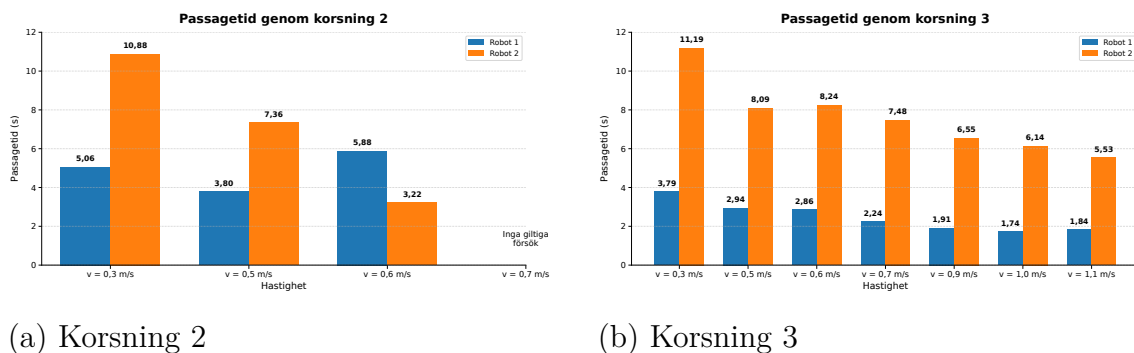


Figur 6.4: Genomsnittligt minsta avstånd mellan Robot 1 och Robot 2 för olika hastigheter i korsning 2 och korsning 3.

Den observerade minskningen i d_{min} för korsning 2 stämmer överens med hypotesen om att högre hastigheter leder till ökad kollisionsrisk. VTL-logiken bygger på att robotar som saknar företräde hinner bromsa eller vänta innan de når konfliktområdet. När hastigheten ökar minskar tiden mellan zoninträde och konfliktzon, vilket gör systemet mer känsligt för positionsfel, kommunikationsfördröjningar och robotarnas bromsförmåga. Systemet fungerar stabilt vid 0,3m/s och 0,5m/s, men säkerhet kan inte garanteras vid högre hastigheter.

För korsning 3 var variationen i d_{min} betydligt mindre. Detta kan förklaras av korsningens reservationslogik och dess utformning, se figur 5.3b. Robotarna tillåts inte befinna sig i samma konfliktområde samtidigt, utan den robot som saknar företräde väntar tills korsningen bedöms vara fri. Resultatet blir större säkerhetsmarginaler och en mer stabil kollisionsfri passage över samtliga redovisade hastigheter.

I figur 6.5 redovisas passagetiderna för Robot 1 och Robot 2 i samtliga experiment. Gemensamt för båda korsningarna är att passagetiden generellt minskar med ökande hastighet, vilket indikerar förbättrad framkomlighet vid högre v .



Figur 6.5: Genomsnittlig passagetid för Robot 1 och Robot 2, för olika hastigheter i korsning 2 respektive korsning 3.

För VTL-logiken bör passagetiderna främst analyseras för $v = 0,3$ m/s och $v = 0,5$ m/s, eftersom dessa hastigheter genomfördes helt utan kollisioner. Vid dessa hastigheter uppfylls framkomlighetskravet K2 då robotarna passerar korsningen inom ändlig tid utan att fastna i väntelägen. Vid $v = 0,6$ m/s och $v = 0,7$ m/s kan K2 däremot inte bedömas på samma sätt eftersom säkerhetskravet K1 inte längre uppfylls.

Variationerna i stapeldiagrammet i figur 6.5 för korsning 2 som representerar VTL-logiken beror på att fyra av fem försök misslyckades vid $v = 0,6$ m/s, medan samtliga försök lyckades vid lägre hastigheter. Resultatet indikerar att experimentet hade kunnat kompletteras med fler mellanliggande hastigheter, exempelvis mellan 0,5 och 0,6 m/s, för att tydligare identifiera gränsen där systemet övergår från stabilt till instabilt beteende.

För korsning 3 som representerar reservationslogiken är passagetiderna mer stabila över hastigheterna se figur 6.5. Robot 1 uppvisar genomgående kortare passagetid än Robot 2, vilket stämmer överens med implementationen där Robot 1 ges företräde i detta experiment. Robot 2 väntar tills korsningen och den gemensamma vägarmen bedöms vara fria innan passage tillåts. Reservationslogiken prioriterar därmed säkerhet framför symmetrisk framkomlighet, vilket förklarar skillnaden i passagetid mellan robotarna.

6.4 Rondellkoordinering

I avsnitt 5.5 beskrivs experiment 6, nedan presenteras resultaten för detta. En kollision i detta avsnitt innebär inte en fysisk kollision mellan robotar utan att nödstoppet aktiverades. Tabell 6.3 sammanfattar kollisionsstatusen för samtliga fem försök vid varje hastighet i experiment 6.

Tabell 6.3: Kollisionsstatus vid rondellmanövern över fem försök per initialhastighet.

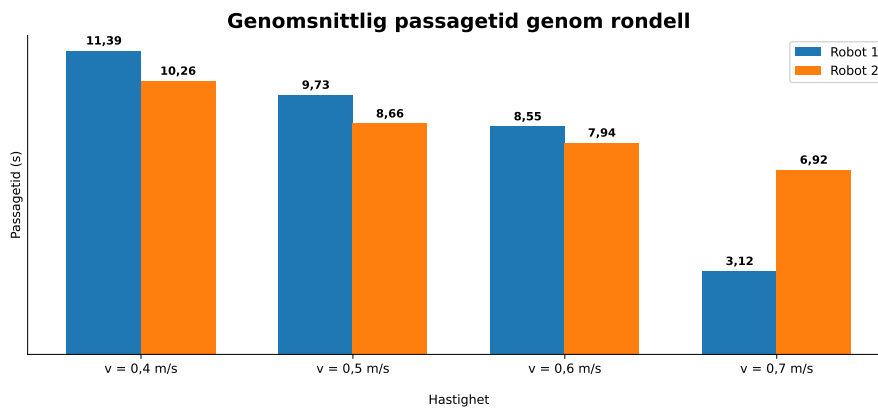
	0,4 m/s	0,5 m/s	0,6 m/s	0,7 m/s
Försök 1	S	S	S	K
Försök 2	S	S	S	-
Försök 3	S	S	S	-
Försök 4	S	S	S	-
Försök 5	S	S	S	-

Notera: 'S' indikerar kollisionsfri rondellmanöver, 'K' indikerar kollision, '-' indikerar att inget försök genomfördes.

Vid hastigheterna $v \in \{0,4; 0,5; 0,6\}$ m/s lyckades robotarna genomföra rondellmanövern utan kollision i samtliga fem försök, vilket indikerar att säkerhetslogiken fungerade konsekvent inom detta hastighetsintervall. Vid 0,7 m/s genomfördes endast ett försök, då detta test resulterade i en kollision. Ytterligare tester vid denna hastighet utfördes inte för att undvika risk för skador på

robotarna. Testet vid hastigheten 0,7 m/s tyder på att systemet inte klarar av att hantera rondellpassage vid denna högre hastighet.

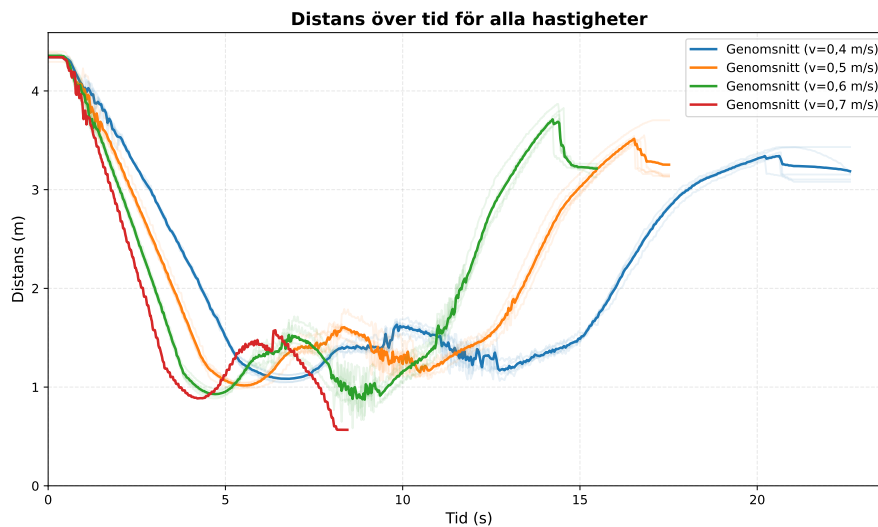
Figur 6.6 visar genomsnittlig passagetid för samtliga hastigheter, specificerat per robot.



Figur 6.6: Resultat av rondellprestanda för experiment 6. I stapeldiagrammet representerar blått Robot 1 och orange Robot 2.

Figur 6.6 visar att passagetiden generellt minskar med ökad hastighet för hastigheterna $v \in \{0,4; 0,5; 0,6\}$ m/s, vilket stämmer överens med hypotesen som gavs i avsnitt 5.5. Prestandaskillnaden mellan de två robotarna är relativt liten och konsekvent genom alla hastigheter. För hastigheten 0,7 m/s avviker resultaten dock markant. Robot 1 uppvisar en mycket kort passagetid på 3,12 s, medan Robot 2 visar 6,92 s. Denna stora skillnad beror på att testet vid 0,7 m/s resulterade i en kollision, vilket innebär att säkerhetskravet R1 och framkomlighetskravet R2 inte uppfylldes vid hastigheten 0,7 m/s.

Figur 6.7 visar avståndet mellan robotarna under hela passagen genom rondellen, från inträde till utträde. Kurvorna illustrerar hur säkerhetsmarginalen förändras under rondellpassagen för olika hastigheter i experiment 6. Det är viktigt att notera att avståndet mäts mellan robotarnas mittpunkter, vilket innebär att det faktiska avståndet mellan robotarnas ytterkanter är mindre. Med tanke på robotarnas storlek innebär ett uppmätt avstånd på 1,0 m mellan mittpunkterna att robotarna är cirka 0,3–0,4 m från att kollidera.



Figur 6.7: Resultat av rondellsäkerheten för experiment 6 vid olika hastigheter. Grafen illustrerar distansen över tid mellan de två robotarna. De skarpa linjerna visar medeldistansen av de fem testerna som gjordes i den hastigheten, medan de skuggade linjerna representerar varje enskilt test i den hastigheten.

Vid hastigheterna $v \in \{0,4; 0,5; 0,6\}$ m/s uppvisar kurvorna ett karakteristiskt mönster: avståndet börjar vid cirka 4,5 m när robotarna närmar sig rondellen, minskar till ett minimum när de befinner sig närmast varandra inuti rondellen, och ökar sedan när de lämnar rondellen. Minimiavståndet sjunker till ungefär 1,0 m mellan mittpunkterna, vilket innebär en faktisk säkerhetsmarginal på 0,3–0,4 m mellan robotarnas ytterkanter. Variationen mellan de fem upprepningarna, visualiserad genom de skuggade enskilda testlinjerna, är relativt liten, vilket indikerar god reproducerbarhet.

Vid hastigheten 0,7 m/s observeras ett avvikande beteende. Kurvan visar att avståndet mellan robotarnas mittpunkter minskar kontinuerligt och når cirka 0,6 m vid ungefär 8 sekunder, där en kollision inträffade. Trots att det uppmätta avståndet mellan mittpunkterna är 0,6 m innebär robotarnas fysiska dimensioner att deras ytterkanter överlappar vid detta avstånd, vilket resulterar i kollision. Detta visar att säkerhetslogiken inte fungerade tillräckligt effektivt vid denna högre hastighet för att förhindra kollisionen. Resultatet demonstrerar att systemet inte uppfyller säkerhetskravet R1 och framkomlighetskravet R2 vid hastigheten 0,7 m/s, men uppfyller båda kraven vid lägre hastigheter.

Resultaten från experiment 6 visar att de implementerade manövrerna uppfyller både kraven R1 och R2 vid hastigheterna 0,4, 0,5 och 0,6 m/s. Vid dessa hastigheter genomfördes samtliga fem försök kollisionsfritt, med reproducerbara passagetider som minskade vid ökad hastighet. Den minsta distansen mellan robotarnas mittpunkter stabiliserades vid cirka 1,0 m, vilket motsvarar en säkerhetsmarginal på 0,3–0,4 m mellan ytterkanter.

7

Slutsats

I detta projekt har ett enhetligt system för kooperativa manövrer utvecklats och implementerats i en laboratoriemiljö. Manövrer för tre trafikscenarier har utvecklats: kolonnkörning, rondellkoordinering och korsningskoordinering. Dessutom har en nödstoppsfunktion skapats för att säkerställa att robotarna inte kolliderar. En omfattande omstrukturering av mjukvaruarkitekturen har genomförts för att förenkla utvecklingen av systemet.

Utifrån de genomförda experimenten kan forskningsfrågorna F-1 och F-2 besvaras för respektive manöver. För nödstoppet fastställs att F-1 uppfylls för hastigheter upp till 0,8 m/s. Gällande kolonnkörning upprätthålls säkerheten inom hela det testade hastighetsintervallet vilket uppfyller F-1, men F-2 uppfylls endast upp till 0,5 m/s, varefter avståndsregleringen tappar sin robusthet och framkomligheten kan inte säkerställas. För korsningskoordinering kan båda korsningstyperna upprätthålla F-1 och F-2. För den VTL-baserade logiken försämrars dock säkerhetsmarginalen vid hastigheter över 0,5 m/s. Den reservationsbaserade logiken uppvisar en stabil kollisionsfri passage för samtliga testade hastigheter upp till 1,1 m/s och där ökad hastighet resulterar i en lägre passagetid. Vid rondellkoordinering säkerställer systemet både F-1 och F-2 upp till 0,6 m/s. Vid 0,7 m/s uppfylls inget av kraven för rondellen.

Den övergripande slutsatsen är att systemets prestanda och säkerhet begränsas av hastigheten. Dessa begränsningar beror till stor del på systemets beroende av den aktuella versionen av det externa lokaliseringssystemet `gulliview_matei`. Mätosäkerhet och latens i detta system medför att robotarna inte hinner exekvera nödvändiga kommandon snabbt nog vid högre hastigheter och har felaktigt positionsdata, vilket riskerar säkerheten.

7.1 Framtida arbete

Den framtagna mjukvaruarkitekturen har skapat en stabil grund för att vidareutveckla systemet. Ett viktigt nästa steg för systemet är att robotens egna sensorer integreras med GulliView för att skapa en bättre omgivningsuppfattning och tillföra både mer redundans och stabilitet. Parallellt med detta projektet har

en ny version av GulliView varit under utveckling. När denna är färdig kan den integreras i det nuvarande systemet för att potentiellt förbättra prestandan och möjliggöra manövrer i högre hastigheter.

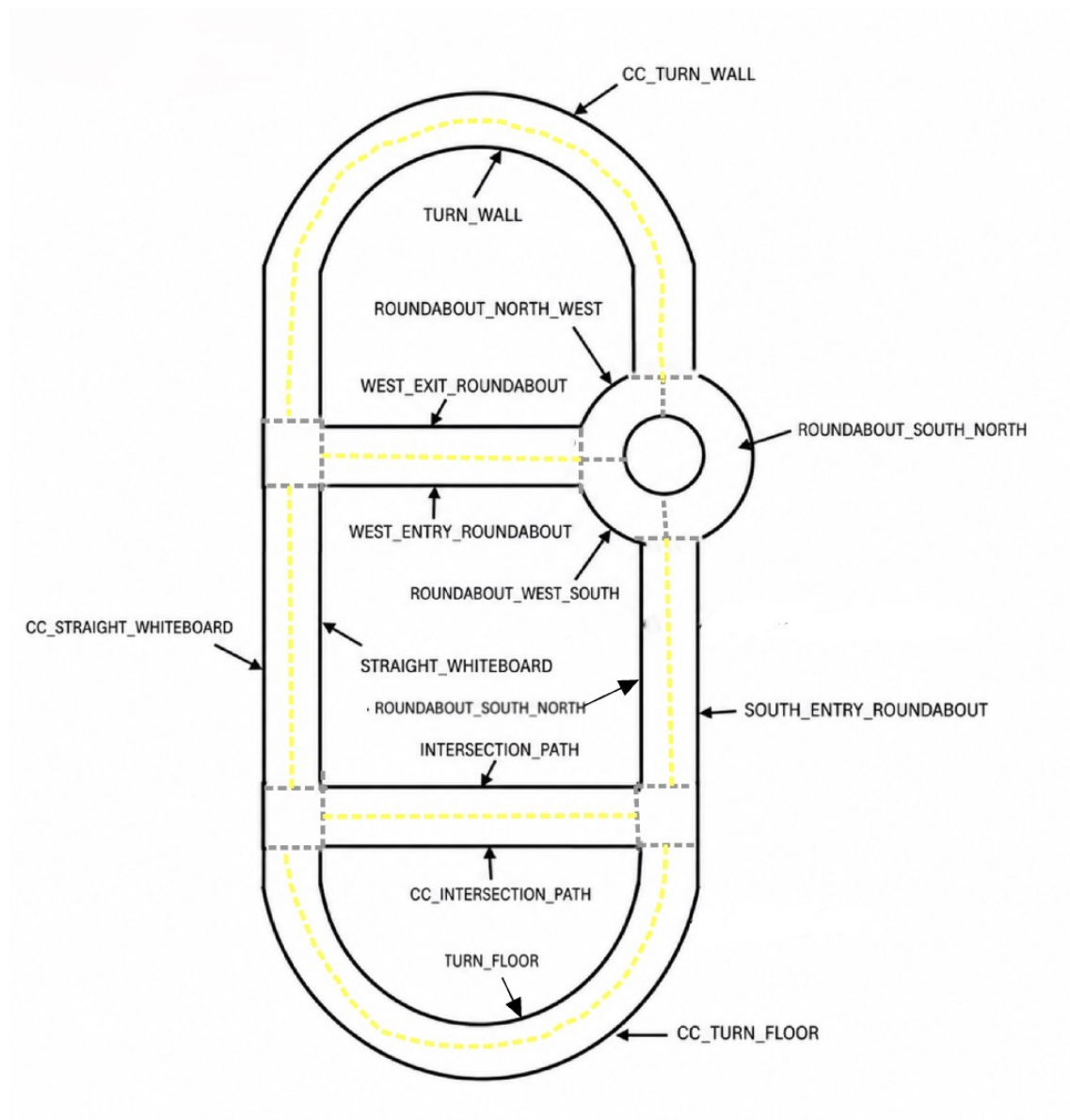
Litteratur

- [1] T. Malmentun, E. Nylander, T. Reinerson, E. Samuelsson, E. Svensson och J. Viberud, "Utveckling samt evaluering av lokalisering och kooperativa styrsystem," Kandidatarbete, Chalmers tekniska högskola, 2025.
- [2] A. Asp, R. Hashemi, O. Johansson, L. Ohlsson, C. Ridderstolpe och S. Täng, "Styrning av nedskalade fordon," Kandidatarbete, Chalmers tekniska högskola, 2022.
- [3] J. Ploeg, B. T. M. Scheepers, E. van Nunen, N. van de Wouw och H. Nijmeijer, "Design and experimental evaluation of cooperative adaptive cruise control," i *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2011, s. 260–265. DOI: 10.1109/ITSC.2011.6082981.
- [4] Eclipse SUMO, *MSCFModel_CC.cpp: bilföljningsmodeller inklusive Ploeg CACC*, 2026. hämtad 17 maj 2026. URL: https://github.com/eclipse-sumo/sumo/blob/main/src/microsim/cfmodels/MSCFModel_CC.cpp.
- [5] A. Casimiro, E. Ekenstedt och E. M. Schiller, "Self-Stabilizing Manoeuvre Negotiation: The Case of Virtual Traffic Lights," i *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, 2019, s. 354–356. DOI: 10.1109/SRDS47363.2019.00048.
- [6] V. Savic, E. M. Schiller och M. Papatriantafilou, "Distributed algorithm for collision avoidance at road intersections in the presence of communication failures," i *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, s. 1005–1012. DOI: 10.1109/IVS.2017.7995846.
- [7] M. Fakirah, S. Leng, X. Chen m. fl., "Visible light communication-based traffic control of autonomous vehicles at multi-lane roundabouts," *J Wireless Com Network*, årg. 2020, s. 125, 2020. DOI: 10.1186/s13638-020-01737-x. URL: <https://doi.org/10.1186/s13638-020-01737-x>.
- [8] WiFiBot. "Wifibot Lab: An Open Platform for R&D and Education," hämtad 16 april 2026. URL: <https://www.wifibot.com/>.
- [9] E. Olson, "AprilTag: A robust and flexible visual fiducial system," i *2011 IEEE International Conference on Robotics and Automation*, 2011, s. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.
- [10] J. Postel, "User Datagram Protocol," Internet Engineering Task Force (IETF), tekn. rapport RFC 768, aug. 1980. URL: <https://www.rfc-editor.org/rfc/rfc768.txt>.

- [11] M. Quigley m.fl., "ROS: an open-source Robot Operating System," i *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, 2009.
- [12] Trafikanalys. "Automatiserad kolonnkörning - en lösning för framtiden?" Hämtad 6 maj 2026. URL: https://www.trafa.se/globalassets/rapporter/2016/rapport-2016_22-automatiserad-kolonnkorning---en-losning-for-framtiden.pdf.

A

Segmentkarta för testbädden



Figur A.1: Översikt över testbäddens segment. Figuren visar segmentnamn, körbanor och kopplingar mellan olika delar av testbädden.

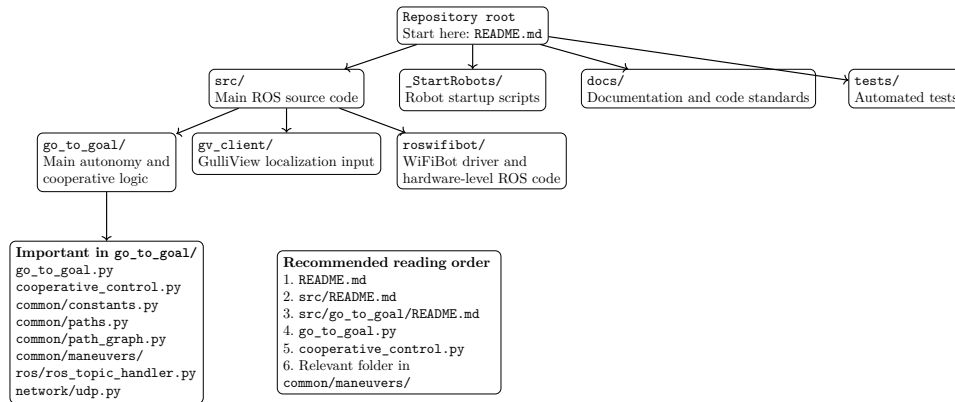
B

Källkod - Bitbucket

Projektets kod finns i Bitbucket-repositoriet **Distributed Cooperative Maneuvers**. Repositoriet innehåller implementationen av de kooperativa manövrarna, ROS-noderna, kommunikationslogiken och hjälpskript för att starta robotarna i evalueringsmiljön. Källkoden nås via projektets Bitbucket-sida:

<https://bitbucket.org/automationarticulatedvehicles>

Syftet med denna bilaga är att ge en kort översikt över var de viktigaste delarna av projektet finns och hur framtida projektgrupper rekommenderas att börja navigera i kodbasen.



Figur B.1: Översikt över Bitbucket-repositoriets olika delar samt läsrekommendationer.