CHALMERS

Utveckling och verifiering av scenarier för autonoma miniatyrbilar

JOHAN HASSEL
ROBERT KEMI
ANDERS NORDIN
FREDDIE NORDSTEDT
JOCKUM SVANBERG
CHRISTIAN ÅGREN

Institutionen för Data- och informationsteknik
Chalmers University of Technology
Göteborg, Sverige 2013
Kandidatarbete/rapport nr 2013:4

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Utveckling och verifiering av scenarier för autonoma miniatyrbilar

JOHAN HASSEL
ROBERT KEMI
ANDERS NORDIN
FREDDIE NORDSTEDT
JOCKUM SVANBERG
CHRISTIAN ÅGREN

© JOHAN HASSEL, June 2013
© ROBERT KEMI, June 2013
© ANDERS NORDIN, June 2013
© FREDDIE NORDSTEDT, June 2013
© JOCKUM SVANBERG, June 2013
© CHRISTIAN ÅGREN, June 2013

Examiner: Arne Linde

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Abstract

This project aims to cover the development of multiple scenarios for controlling autonomous vehicles, with the goal of providing autonomous cars that will be able to handle complex situations in traffic. The scenarios are developed on the Gulliver-platform, a platform of autonomous miniature vehicles currently in development at Chalmers University of Technology. Two scenarios are realized, cruise control and autonomous driving through an intersection without traffic lights. The scenarios utilizes both sensors and car-to-car communication. Furthermore, in order to evaluate the developed scenarios before implementing them on the physical testbed, a simulator capable of simulating sensors, communication and vehicular movement is required and developed.

Sammanfattning

Detta projekt behandlar utvecklingen av ett flertal styrscenarier för att styra autonoma bilar, med målet att ge autonoma bilar möjligheter att köras i komplexa trafiksituationer. Styrscenarierna utvecklas till Gulliver-plattformen, en plattform för autonoma miniatyrfordon som utvecklas på Chalmers Tekniska Högskola. Två styrscenarier implementeras, farthållning till framförvarande fordon samt autonom körning i korsning utan trafikljus. Scenarierna baseras på sensorer och kommunikation med andra fordon. För att evaluera de utvecklade scenarierna innan dessa körs på den fysiska plattformen krävs och utvecklas en simulator med stöd för sensor-, kommunikations- och mobilitetssimulering.

Innehåll

Termförklaringar

Arduino En mikroprocessor på ett litet kretskort med lättåtkomliga i/o anslutningar.

Broadcast En term inom datorkommunikation som innebär att paket som skickas mottages av alla på nätverket.

Charmeleon-mac Ett kommunikationsprotokoll för TinyOS system utvecklat av Mohamed Mustafa på Chalmers.

Distanskort Ett kretskort på Gulliver-plattformen vars funktion är att styra upp till 6 st ultraljudssensorer, samla mätdata och propagera datan till Huvudkortet. Kortet innehåller en I2C-buss med 6 kontakter för sensorerna samt en M-UART-buss med kontakt för sammankoppling med Huvudkortet.

Huvudkort Ett kretskort på Gulliver-plattformen som är kopplat till alla andra moduler i fordonen. Huvudkortet tar emot information från olika källor och skickar vidare det till rätt ställe. Huvudkortet sköter även de autonoma funktionerna hos Gulliver-plattformen som t.ex förmågan att följa en karta.

I2C buss En enkel synkron seriell multimasterbuss (tillåter att flera moduler med unika adresser ligger på samma buss) för låghastighetsöverförning av information mellan ett flertal enheter.

MICAz En liten strömsnål processor med tillhörande radiomodul. den är från början utvecklad för sensornätverk men den används i gulliver för kommunikaktion mellan bilarna.

nesC En variant av programmeringsspråket C som används för att programmera TinyOS enheter (Gay et al., 2003)

Python Välanvänt skriptspråk. Python är ett programmeringsspråk med stöd för flera olika programmeringsparadigmer, bland annat objektorienterad och funktionell programmering.

QT Ramverk för att utveckla grafiska applikationer i C++

QT kontroller Även kallat Map kontroller. Programvara som körs på ITX-datorerna på Gulliver-plattformen samt en extern dator.

RCM *Ranging and Communications Module*, mäter avstånd mellan två moduler, kan genom av triangulering användas för att beräkna en position.

Sonar En typ av sensor som mäter avstånd med hjälp av ultraljudsvågor.

Strålkast (eng. Ray cast) Teknik som används flitigt inom grafikrendering, metoden innebär att man beräknar hur ljusstrålar breder ut sig.

TinyOS Händelsetbaserat ramverk utvecklat för trådlösa sensornätverk (Gay and Levis, 2009).

TOSSIM Simuleringsmiljö för TinyOS-applikationer på MICAz-processorn.

UART Hårdvara som omvandlar data mellan parallell och seriell representation.

Vågkast En version av strålkast där strålen divergerar under utbredningen i rummet och skapar en vågfront.

1

Inledning

Detta kapitel syftar till att ge en övergripande bild över projektets omfång och syfte. Först diskuteras möjliga områden då autonoma fordon kan anses fördelaktiga över konventionella bilar. Därefter presenteras problemformuleringen utifrån både utvecklings- och verifieringssynpunkt. Slutligen definieras de två scenarierna som behandlas och refereras till under resterande del av rapporten.

1.1 Bakgrund

Många av dagens städer präglas av stora framkomlighetsproblem i trafiken. För att motverka bildandet av köer, reducera miljöpåverkan och potentiellt öka säkerheten finns det en teknisk vision av autonoma bilar anpassade för vardagliga körscenarier. Dessa bilar kan genom trådlös kommunikation och sensorer samverka med varandra för att skapa ett bättre och mer kontrollerat trafikflöde. Vid längre resor kan dessa bilar bilda kolonner där bilarna ligger tätt intill varandra för att minska luftmotståndet och minska vägytan som bilarna tar upp. Vid korsningar kan bilarna bestämma vem som bör köra först utan att orsaka flaskhalsar i trafikflödet genom att behöva vänta på ett trafikljus och därmed göra bättre användning av vägytan i städer.

I nuläget är det dyrt och svårt att utveckla styrsystem på befintliga, fullskaliga fordon, dessutom tillkommer en risk för kostsamma skador på hårdvara. Därför finns ett behov av en tillgänglig testbädd för utveckling av sådana algoritmer, en sådan plattform är Gulliver (Pahlavan et al., 2011) som utvecklas på Chalmers. Gulliver-plattformen består av miniatyrbilar i skala 1:8 vilka är utrustade med hårdvara som tillåter kommunikation och avståndsmätningar.

1.2 Problemformulering

För detta projekt finns det ett intresse av att utveckla ett antal på scenarion till Gulliver-plattformen. Dessa scenarier refereras till i projektet som *automatisk farthållare* samt *trafikkorsning*. De utvecklade scenarierna behöver även utvärderas innan överföring till hårdvara, och därmed utvecklas en simulatormjukvara. Ett antal krav specificerades för varje projektdel, och dessa formuleras i denna rapportsektion.

1.2.1 Scenarier

Scenarierna är formulerade för att falla tillbaka på målet med utvecklingen av autonoma fordon, att dessa genom trådlös kommunikation och sensorer ska kunna samverka med varandra för ett bättre och mer kontrollerat trafikflöde. Att dela upp styrningen av ett autonomt fordon i specifika scenarion gör att det är enklare att bryta ner ett större problem i mindre små och därmed ge en större möjlighet att utvärdera om målen för varje scenario uppfyllts eller inte. Övergripande krav på alla styrscenarier:

1. Ett fordon ska aldrig krocka med ett annat fordon
2. Beslutsfattandet om att öka eller sänka hastigheten ska implementeras på MICAz
3. Algoritmerna ska få plats på det begränsade minnet i MICAz-modulen

Farthållare

I detta scenariot undersöktes hur ett fordon ska kunna följa ett annat fordon helt autonomt med hjälp av sensorer. Det vill säga att fordonet ska anpassa sin hastighet och sitt avstånd till fordonet framför.

Specifika krav för farthållaren

1. Ett fordon ska kunna hålla sig inom säkerhetsavståndet till fordonet framför sig.
2. Ett fordon ska kunna anpassa sin hastighet till fordonet framför sig.

Korsningar

I detta scenariot undersöks hur fordon ska hantera en oövervakad korsning. Fordonen skall kunna ta in information från sensorer och avgöra om ett annat fordon befinner sig i närheten av korsningen. Om så är, så ska dessa fordon utbyta avstånden till korsningen och därefter anpassa sin hastighet efter detta för att skapa en smidig passage genom denna.

1. En bil ska inte vänta om korsningen är ledig
2. En bil ska inte stanna i korsningen

1.2.2 Simulator

För att känslig utrustning på Gulliver-plattformen inte ska komma till skada under utvecklingen finns det även behov av en simulator som kan simulera hela plattformen där scenariernas funktionalitet kan utvärderas innan ett fysiskt test på själva plattformen påbörjas. Denna simulator behöver endast kunna köra ett litet antal bilar för att en tillfredställande utvärdering av de två scenarierna ska kunna utföras. Eftersom Gulliver-plattformen fortfarande utvecklas kan det i framtiden finnas ett intresse av att kunna simulera betydligt fler fordon samtidigt, för att utvärdera scenarier utifrån ett trafikflödes-perspektiv, alternativt kunna köra simulationen parallellt med fysiska tester av plattformen. Därmed bör simulatorn kunna köras i realtid och utvecklas för att vara skalbar, så att en eventuell vidareutveckling kan ske i senare projekt.

Kraven på simulatorn kan delas in i följande lista:

1. Simulera TinyOS-kod
2. Simulera Gulliver-plattformens mobilitet

3. Simulera sensormätningar
4. Simulera bilarnas kommunikation
5. Synkronisera delsimuleringar
6. Köras i realtid
7. Utvecklas för att vara skalbar

1.3 Avgränsningar

Avgränsningarna i projektet är gjorda med avseende på att göra målen mer mätbara, minska komplexiteten och för att begränsa tidsåtgången.

1.3.1 Farthållare

- Endast ett följande fordon, samt ett ledande fordon på banan samtidigt
- Inga hinder som kan påverka sensormätningar

1.3.2 Korsningar

- Inga trafikljus eller centraliserade kontrollenheter
- Inga hinder som kan påverka sensormätningar
- Maximalt två fordon samtidigt
- Samtliga fordon befinner sig alltid inom kommunikationsavstånd från varandra

1.3.3 Simulator

- Maximalt två simulerade fordon samtidigt
- Sensormodell utan interferenseffekter

2

Material

Projektet baseras på Gulliverplattformen, vilket är en testplattform som används för att utveckla, demonstrera och skapa prototyper av autonoma fordonssystem (Vedder, 2012). Plattformen bygger på en radiostyrd miniatyrbil i skala 1:8, vilken ett flertal tidigare projekt på Chalmers därefter utvecklat hårdvara och mjukvara till för att möjliggöra en funktionalitet liknande en autonom robot.

Utöver bilgrunden innehåller plattformen bland annat hårvara som tillåter kommunikation till andra fordon, positionering genom triangulering samt ett flertal sensorer. Hårdvaran beskrivs i detalj del för del nedan:

Huvudkortet Den del i bilen som tar hand om största delen av logiken och styrningen. De andra komponenterna är anslutna till Huvudkortet. Den får en karta med positioner från QT-kontrollern och räknar sedan ut riktningen för att ta sig till positionen.

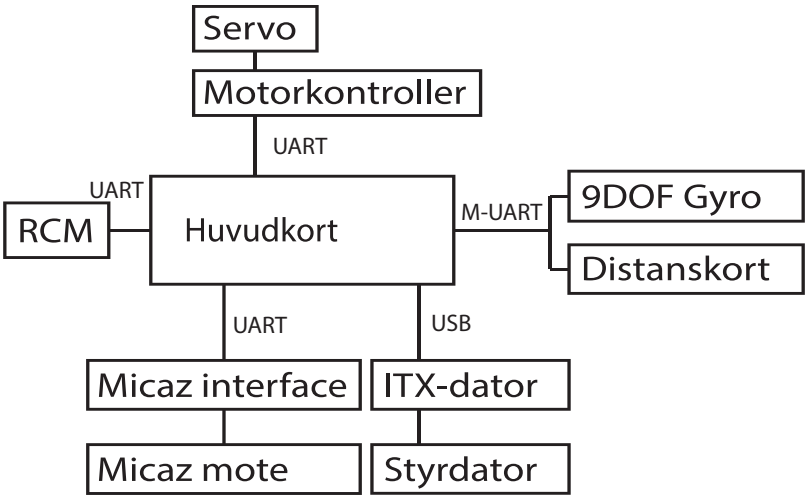
Sensorkortet Har ett 9DOF Gyroskop/Magnetometer för att hålla reda på hur bilen rör sig. Det finns även anslutningar för att koppla in sex ultraljudssensorer av typen SRF08 samt anslutningar för att koppla in ett antal IR-sensorer. Kortet tar emot data från sensorerna och lagrar dem, sen kan Huvudkortet be om data som sedan skickas. Det utvecklades också en möjlighet att stänga av de sensorer som inte var önskvärda att använda för tillfället.

Motorkontrollern Ansvarar för att styra motorn och servot som styr bilen. Den får kommandon från Huvudkortet om hastighet och önskad riktning, vilket den sedan använder för att styra motor och styrservo.

MICAz-modulen Den beslutsfattande delen av plattformen som styr över Huvudkortet genom ett Interfacekort. MICAz-modulen kan även kommunicera med andra MICAz-moduler på andra bilar via RF-radio.

RCM-modulen Möjliggör positionering av bilen. RCM-modulen kräver att ett flertal andra RCM-moduler används för att en position ska kunna bestämmas genom triangulering, genom att mäta upp tiden det tar för radiosignaler att färdas mellan de olika modulerna.

ITX-datorn Används för att via trådlöst nätverk ge bilen kartan att följa. Tillåter även att en server kan skicka kommandon till plattformen samt styra den med en handkontroll.



Figur 2.1: *Bild över hur de olika plattformsdelarna är ihopkopplade.*

2.1 Sensorer

För att uppnå högre säkerhet och precision i fordonens beslutsfattande krävs det mer informationskällor i kalkyleringen i form av sensorer. Att ta emot information från ett flertal olika källor innebär även en högre säkerhet ifall någon informationskälla skulle vara missvisande, felaktig eller avstängd.

Ett flertal informationskällor är att föredra då ett av önskemålen är att kunna simulera fel i olika sensorer. För att fordonen skall kunna hantera de nya scenarierna behövs nya sensorer för att kunna ta in mer information om omgivningen. Framförallt i korsnings scenariot där fordonen inte har någon sensor-information om sidorna på fordonet, eftersom den ursprungliga Gulliver-plattformen inte har stöd för detta.

Nya sensorer innebär även nya problem i form av störningar från varandra, sammanfogning av information och montering samt placering av sensorerna.

Placeringen av nya sensorer kräver tester och planering för att få fram rätt information från rätt omgivning. En felaktigt placerad sensor kan ge missvisande information eller störa ut andra sensorer. Varje fordon pulserar sina nuvarande ultraljudssensorer sekvensiellt med 65 ms mellanrum. Varje ultraljudssensor skickar iväg en ljudvåg och väntar på ekon i 65 ms. Om en ljudvåg från ett annat fordons ultraljudssensor skulle träffa dennes sensor under detta tidsfönster kommer det registreras som ett eko och ge felaktig data. Fordonen är oberoende av varandra och har sina egna starttider samt klockfrekvenser. Risken för att ett fordon skall plocka upp felaktiga ljudvågor är därmed relativt stor. I värsta fall har två fordon samma klockfrekvens och en liten skillnad i starttider, vilket medför att om ultraljudssensorerna träffar varandra kommer en utav dem alltid få felaktiga ekon.

2.2 RCM

Bilarna är utrustade med ett positioneringssystem RCM, Ranging and Communications Module (Altby et al., 2012). RCM mäter avståndet mellan två enheter genom radiokom-

munikation, och tillåter alltså positionering genom triangulering. Radiokommunikationen mellan RCM-enheter sker mellan frekvenserna 3.1 GHz och 5.3 GHz med en standardhastighet på 9.86 kbps, med möjlig hastighet upp till 158 kbps. Maxavstånd vid lägsta hastighet är 358 m utan hinder.

Vid triangulering krävs 3 basstationer som varje plattform beräknar positionen från genom den mottagande RCM-enheten som sitter på fordonen. Positionsdata från mätningarna skickas vidare till Huvudkortet och används sedan för att kontrollera att bilarna följer den väg de är instruerade att följa.

RCM erbjuder ett programmeringsgränssnitt över Ethernet och serieport. För att låta flera bilar kommunicera utan störningar har ett enkelt MAC-protokoll programmerats. MAC-protokollet ser till att bilarna synkroniserar sina tidsintervall och därmed får tillgång till triangulering mot basstationerna.

Intern kommunikation

Ett av kraven är att låta MICAz-modulen sköta allt beslutsfattande. Detta innebär att ny intern kommunikation måste införas mellan Huvudkortet och MICAz-modulen då Huvudkortet styr fordonen och innehar all sensor-information. MICAz-modulen måste kunna justera hastigheten på fordonet samt kunna få in information från sensorerna för att kunna fatta beslut.

2.3 Simulatormjukvara

Under projektets planeringsfas schemalades utvecklingen av en simulator för att kontrollera den utvecklade mjukvaran innan denna överförs till fordonsplatformen. Det behöver finnas tillgång till tidiga mjukvarubaserade tester för att försäkra att utrustningen inte kommer till skada under fysiska tester. Det definierades ett antal punkter med moment som simulatorn skulle kunna köra och evaluera, vilka kan delas in i två huvudmoment: Mobilitetssimulering och TinyOS-simulering. Redan i början av projektet användes TOSSIM för att simulera koden som utvecklas i TinyOS. Däremot kunde ett bra alternativ för mobilitetssimuleringen kunde inte väljas lika tidigt utan istället sammanställdes ett fleral potentiellt användbara mjukvaror eller verktyg i följande lista:

- Open source Development Architecture for Virtualization of Networked Cyber-physical system Infrastructures (OpenDaVINCI)
- Simulation of Urban MObility (SUMO)
- Egenutvecklad mjukvara

En grov sammanställning av funktionaliteten hos de olika simulatorerna finns i tabell 2.1.

OpenDaVINCI

OpenDaVINCI är ett mjukvarupaket bestående av ett flertal mjukvarudelar vilka synkroniseras via nätverk, vilket utvecklades för simulering av autonoma bilar på kartor i en 3D-värld. Paketet innehåller bland annat stöd för simulering av sensorer modellerade efter kameror och programvara för att utveckla nya kartor (Berger, 2010). Det finns i nuläget inget ramverk för att styra simulationen direkt via skriptspråk, istället måste detta utvecklas så att ett skript skickar paket via nätverk som styr simulationen.

Tabell 2.1: *Övergripande funktionalitet hos utvärderade simulatorer.*

Simulator	OpenDaVINCI	SUMO	Egenutvecklad
Typ av simulering	3D, Fordonssimulator	2D, Trafiksimulator	2D, Fordonssimulator
Vägbaserad styrning	Ja	Ja	Nej
Sensorsimulering	Ja	Nej	Ja
Skriptanslutning	Behöver utvecklas	Python via nätverk	Python direkt

SUMO

SUMO beskrivs enligt Behrisch et al. (2011) som "SUMO is an open source traffic simulation package [...] SUMO is not only a traffic simulation, but rather a suite of applications which help to prepare and to perform the simulation of traffic." Programvaran är utvecklad för simulering av trafikflöden på fordonsnivå över stora ytor (Behrisch et al., 2011), men tillåter kontroll över enskilda fordons egenskaper och körstil genom anslutning till Python-kod via gränssnittsmodulen TraCI. Detta gör att styrningen av bilen kan kontrolleras av ett annat skript som körs parallellt med mobilitetssimuleringen. SUMO har tidigare använts för att simulera trafikflöden parallellt med fysiska tester på Gulliver-platformen (Berger et al., 2013). Detta visar att både simuleringar och fysiska tester kan utföras på identiska kartor. Detta kan vara fördelaktigt vid utvärdering av simuleringens pålitlighet eftersom det tillåter att samma scenario körs som fysiskt test och simulering samtidigt, och alltså kan precisionen i mobilitetssimuleringen utvärderas.

Egenutvecklad mjukvara

Vi utvärderade även möjligheten att utveckla egen mjukvara för simulering av plattformen och implementerade en prototyp till hur denna skulle kunna fungera. För att se till att koden skulle kunna återanvändas vid behov delades denna prototyp tidigt i utvecklingsfasen upp i två moduler, en mobilitetsmodul och en sensormodul.

Mobilitetsmodulen består av en enkel rörelsemodell för fyrhjuliga fordon helt utan slirande hjul samt möjligheten att kontrollera varje enskilt fordons hastighet och framhjulsvinkel. Detta gör att simuleringen inte tar hänsyn till fördefinierade vägar och filer till skillnad från Gulliver-plattformen. Mobilitetsmodulen är istället tänkt att användas för utvärdering av scenarier där hastighetskontroll för fordonssynkronisering ligger i fokus, vilket i vårt fall innefattas av farthållar- och korsnings-scenariot.

Sensormodulen används för att simulera avståndsbedömningen hos de ultraljudssensorer som sitter på fordonen. Simuleringen modellerar varje enskilt fordon som en rektangel på vilken ett antal ultraljudssensorer kan placeras och roteras för att efterlikna placeringen på Gulliver-plattformen. Alla ultraljudssensorer har en konstant synfältsvinkel (field of view) och maxavstånd vilket definierar ett område där objekt kan ge utslag på avståndsmätningen. För varje ultraljudssensor behandlas samtliga fordon som helt eller delvis befinner sig inom utslagsområdet, med undantag för det fordon ultraljudssensorn sitter på. Den närmsta punkten inom utslagsområdet som ligger på kanten av fordonets rektangel för varje fordon beräknas, och till det lägsta värdet adderas brus. Resultatet blir en approximation av det sökta avståndet som efterliknar värderna som avläses på den fysiska plattformen.

3

Metod

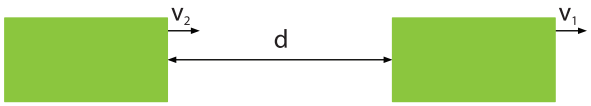
Under utvecklingsfasen delades projektgruppen upp i flera undergrupper vilka behandlade olika delar av projektutvecklingen. I följande kapitel diskuteras hur de olika delarna utformades och implementerades.

3.1 Styrscenarier

Styralgoritmerna för Gulliver körs på MICAz-moten eller i den simulerade miljön i TOSSIM. Båda scenarierna baseras på att algoritmen har tillgång till antingen samtliga eller vissa sensorer på plattformen, vilka beskrivs i kapitel 2. I korsningsscenariot beskrivs olika algoritmer som använder olika sensorer, eftersom ett av målen med projektet är att utveckla reservtyper av scenarier då en viss typ av sensorer inte är tillgänglig. Vidare måste scenariokoden få plats i det lediga programminnet på MICAz-modulen. I nuläget har scenariekoden tillgång till max 40 kilobyte av totalt 128 kilobyte, då resterande minne är upptaget av plattforms-specifika metoder för kommunikation och styrning.

3.1.1 Fartkontroll

Implementationen av Farthållaren baseras på en algoritm som endast tar emot data från Gulliver-plattformens ultraljudssensorer. Det fordon som är längst fram i detta scenario bestämmer farten för bakomvarande fordon. Fordonet bakom använder sig av sina sensorer för att anpassa sin fart så att ett fördefinierat säkerhetsavstånd behålls.



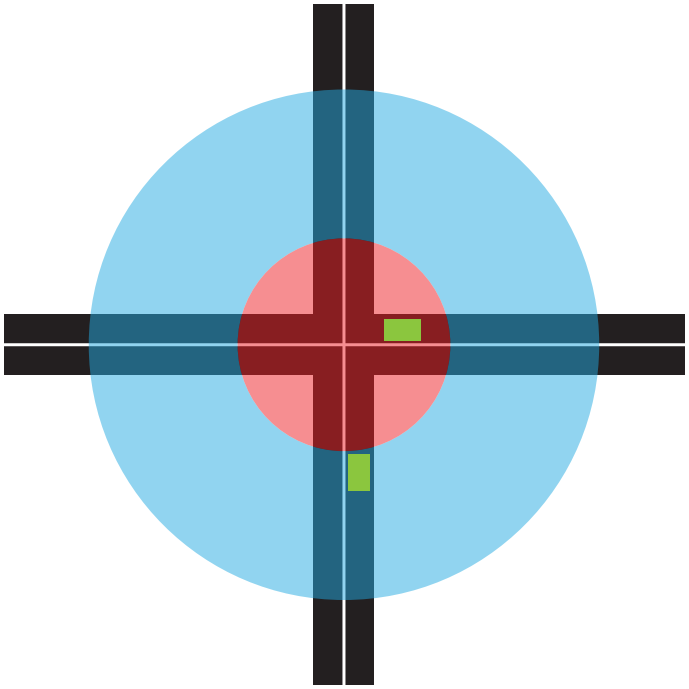
Figur 3.1: *Bild av Fartkontroll där v_1 är ledarfordonets hastighet, v_2 är efterföljande fordons hastighet samt som d är säkerhetsavståndet*

Algoritmen kan delas in i följande flöde:

1. Det bakre fordonet aktiverar de två främre sonar-sensorerna.
2. Mätdata mottages
3. Genom avståndsmätningen bestäms nuvarande avstånd till, samt hastighet för framförvarande fordon.
4. Det bakre fordonet anpassar sin hastighet för att behålla säkerhetsavståndet.

3.1.2 Korsning

Till skillnad mot fartkontrolls-scenariot behöver fordonen i korsnings-scenariot även kunna kommunicera med varandra. En korsning består av en area (den blå cirkeln i fig 3.2) som signalerar att ett fordon kommer till korsningen. Det är även definierat ett mindre område (röda området i fig 3.2 i korsningen som är det kritiska området där ett fordon måste stanna innan för att inte stanna i korsningen och blockera den. Detta mindre område räknas ut dynamiskt med hjälp av radien på korsningsarean och fordonets hastighet. En area kan definieras i Gullivers kart-klient, detta är bara för den blåa arean. Från information från kartklienten tillsammans med bilens hastighet kan man sedan enkelt räkna ut den röda arean.



Figur 3.2: *Bild av korsningen, det blåa området är korsningsarean där algoritmen startar och det röda är minsta avståndet från korsningen ett fordon behöver för att stanna.*

När fordonet anländer till korsningen och får en händelse (Event) så räknar den ut det kritiska minsta avståndet från korsningen (var den röda arean börjar), sen startas en subrutin som i regelbundna intervaller beräknar fordonets ankomsttid till det kritiska minsta avståndet från korsningen. Detta värde tillsammans med korsningens id sänds genom broadcast till alla andra fordon via Chameleon-protokollet (Leone et al., 2010). Fordonet fortsätter sen med att lyssna efter andra fordon som anländer till samma korsning. Om inget annat fordon närmar sig korsningen innan fordonet har kommit till det kritiska området så fortsätter fordonet med samma hastighet. Om fordonet tar emot ett

meddelande om att ett annat fordon närmar sig samma korsning så jämför den sin egen ankomsttid med det andra fordonets. I de fall där fordonets ankomsttid är mindre än det andra fordonets ankomsttid adderat med tiden det tar att passera korsningen samt en liten tidsmarginal fortsätter fordonet som vanligt. Om detta inte uppfylls stannar fordonet innan det kritiska området i korsningen. När fordonet lämnar korsningsarean avslutas subrutinen som styr broadcasten av anländningstid.

Om det inte kan bestämmas vilket fordon som ankommer först stannar båda fordonen vid korsningen och aktiverar sina sonar-sensorer. När fordonet upptäcker att det inte är ett fordon till höger kör det. Detta kan potentiellt låsa systemet om det står ett fordon vid varje infart till korsningen. Enligt projektets avgränsningar begränsas dock antalet fordon till två, och därmed kan detta ej uppkomma.

3.2 Utvärdering av befintliga sensorer

För att kunna effektivisera implementationen av styrscenarierna behövde vissa effekter på sensormätningar kartläggas. Dessutom krävde vissa implementationsidéer att Gulliverplattformen utökas med ett ytterligare antal sensorer. Till vilken gräns detta var utförbart eller inte behövde också analyseras. I följande sektion diskuteras hur, och på vilken hårdvara denna analys utförs.

3.2.1 Arduino och SRF08

På fordonen fanns sedan tidigare två ultraljudssensorer av typen Devantech SRF08 [1]. Dessa är högpresterande ultraljudssensorer (sonar) som klarar av att mäta avstånd från 3 cm till 600 cm. I samma krets finns även en fotodiod för att mäta ljusnivån i omgivningen, denna används dock inte i projektet. Sensorkretsarna ansluts via en standardiserad I2C buss till fordonens distanskort som lagrar informationen från sensorerna tills dess att huvudkortet vill åtkomst till dem.

Två av dessa ultraljudssensorer användes tillsammans med en Arduino Uno mikroprocessor (Arduino) för att kunna utföra interferenstester med sensorerna. Arduinon är en enkel mikroprocessor som snabbt kunde kopplas ihop med ultraljudssensorerna via dess I/O portar med hjälp av ett fåtal externa komponenter. Med hjälp av Arduinos community hittades en färdigprogrammerad klass för att hantera en av dessa ultraljudssensorer tillsammans med Arduinos egna bibliotek Wire. Wire tillhandahåller enkla funktioner för att kunna kommunicera via en I2C buss. Den färdigprogrammerade klassen tillhandahåller simpla funktioner för att kunna ge ultraljudssensorerna kommandon samt plocka in dess data. Tillsammans med dessa två hjälpmedel programmerades en applikation för att kunna hantera två ultraljudssensorer på samma I2C buss samtidigt.

Varje ultraljudssensor på samma I2C-buss måste ha en unik adress, och den programmerade applikationen möjliggör en snabb och enkel omprogrammering av ultraljudssensorernas adresser. Varje ultraljudssensor av typen SRF08 levereras med en och samma I2C-adress från återförsäljare. Vid tidigare tillfällen har omprogrammering skett via fordonen genom att programmera om Huvudkortet och kopplat in en sensor i taget vilket var tidskrävande. Detta kan nu göras mycket snabbare med hjälp av Arduinon och vår applikation, som enbart tar ett par sekunder.

[1] http://www.robot-electronics.co.uk/htm/srf08tech.html

3.2.2 QT-Kontroller

Fordonens Linux-datorer innehåller ett program kallat QT-kontroller vars uppgift är att kommunicera med huvudkortet samt med en extern dator via wifi för att ge användaren diverse information. QT-kontrollerns kod har tillsammans med ny kod i huvudkortet modifierats för att kunna ge användaren värdena från ultraljudssensorerna. Dock avfärdades denna idén mer eller mindre efter att Arduino-lösningen togs fram då denna var mycket enklare, snabbare och effektivare att genomföra för att få fram sensor-data.

3.2.3 Tester

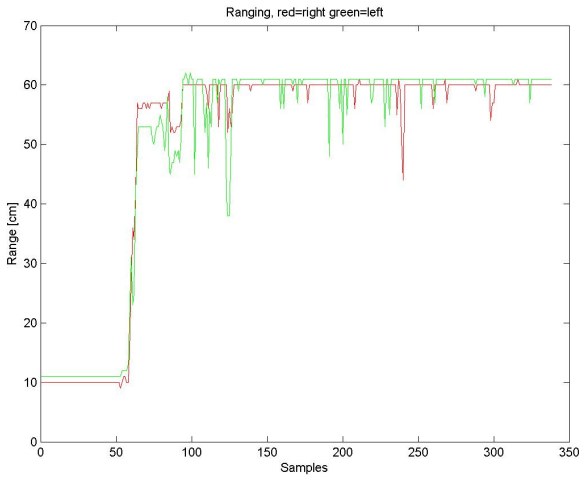
För att de nya scenarierna skall kunna få tillräckligt med information om omgivningen studerades en rad olika lösningar på nya sensorer. Preliminärt undersöktes möjligheten att expandera plattformen med flera av de redan installerade ultraljudssensorerna av typen SRF08. Till en början planerades det att utöka antalet ultraljudssensorer på fordonen till sex för att kunna ha bra uppsikt på all omgivning åt alla håll.

För få fram olika möjliga lösningar på problemet och undersöka om detta var möjligt gjordes ett test på ultraljudssensorerna. Två ultraljudssensorer kopplades ihop via en I2C-buss till en Arudino mikroprocessor. Arduinon programmerades med hjälp av vår applikation till att matcha fordonens sekvensiella pulsering av ultraljudssensorerna med samma väntetider mellan pulserna. Den programmerades även till att sända ultraljudssensorernas första eko-värde via en USB-kabel till en bärbar dator efter varje pulsering. En Python applikation programmerades till datorn för att samla in all information och spara undan det organiserat i en fil för vidare behandling. Tillsammans med ett fordon fixerades dessa två lösa ultraljudssensorer på ett plant underlag. Fordonet flyttades därefter runt i olika positioner för att efterlika de olika scenarierna. Fordonet testas med och utan dess egna ultraljudssensorer igång för att se hur ultraljudssensorerna stör ut varandra. Varje testfall placerades fordonet ut noggrant med fasta mått från ultraljudssensorerna. Arduinon startades och kördes i ungefär en minut per testfall. Cirka 20.000 mätvärden togs ut tillsammans.

Resultatet var mycket bra trots att viss mätdata försvann i överföringen. Med väntetiden mellan pulserna på 65 millisekunder enligt fordonen förväntades det ungefär 15 mätvärden per sekund. Dock hade Python applikationen enbart samlat in runt 7,5 mätvärden per sekund. Felet tros ligga i överföringen mellan Arduino och dator vilket skulle kunna förbättrats om noggrannare tester önskas. Ur datan kunde störningarna lätt urskiljas. Testerna kördes i 3 kategorier: Farthållare, Korsning samt Annat. Och i varje kategori gjordes ett flertal tester med olika parametrar. Nedan visas 4 av de testfall som ansågs vara mest illustrativa.

Farthållare

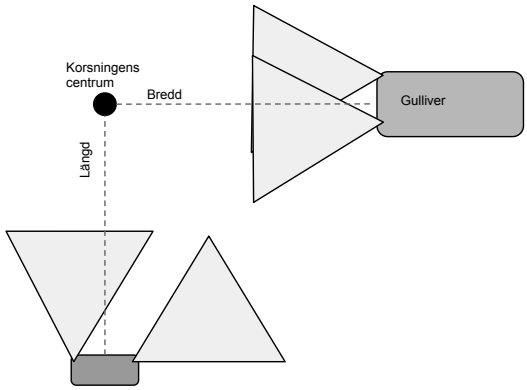
I kategorin Farthållare var målet att ta fram information om hur mycket sensorerna stör ut varandra i Farthållar-scenariot. Detta för att se hur pålitlig sensor-informationen är i detta scenario. Fordonet placerades rakt framför sensorerna på olika avstånd från 10 cm till 120 cm. I figur 3.3 syns ultraljudssensor-värdena från ett av testfallen då ett fordon stod på ett avstånd av 60 cm med sina egna ultraljudssensorer igång. I figuren synes de två ultraljudssensorernas returnerande värde i cm på y-axeln. Och antalet pulseringar på x-axeln. I detta fall hade fordonet en 4-5 meter fram till väggen framför sig. Störningarna som syns i grafen antages vara registrerade av ljudvågor som har studsat runt i rummet från fordonets ultraljudssensorer.



Figur 3.3: *Resultat från Farthållare.*

Korsning

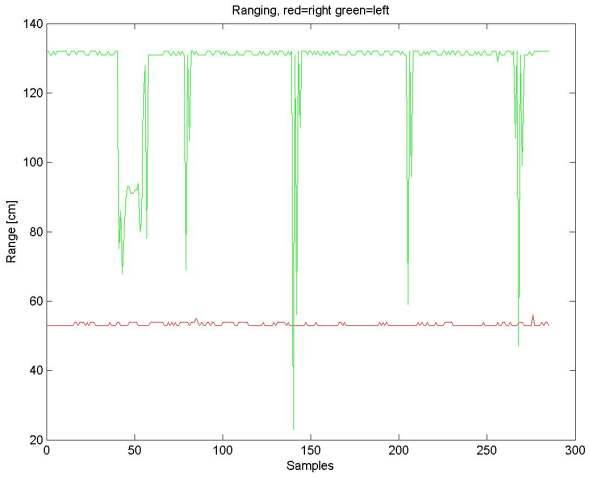
Målet i korsnings kategorin var att samla in information om störningarna i ett Korsningsscenario med en sensor vinklad 45 grader åt sidan. Detta för att undersöka om hur väl en vinklad ultraljudssensor kan registrera ett möjligt fordon i en korsning. I figur 3.4 syns en översikt på hur testerna utformades. Ett flertal olika värden på bredd och längd till korsningens centrum testades. I figur 3.5 syns ett testfall då fordonet stod på vänster sida från sensorerna med ett avstånd på 100 cm från korsnings centrum. Den högra ultraljudssensoren vinklades 45 grader till höger och pekade mot fordonet. Även i detta testfall fångades fordonets störningar upp. Dock är denna vinklade sensors funktion enbart att ta reda på om det finns något fordon där, distansen till fordonet är inte lika relevant i detta fall.



Figur 3.4: *Översikt över testernas utformning.*

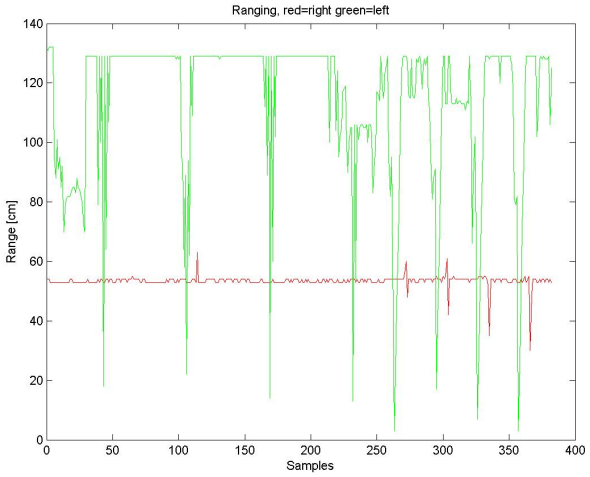
Annat

Målet i kategorin Annat var att testa olika sensor-upplägg. I figur 3.6 syns ett testfall då en utav ultraljudssensorna var pekade rätt emot fordonets front som hade sina ultraljuds-



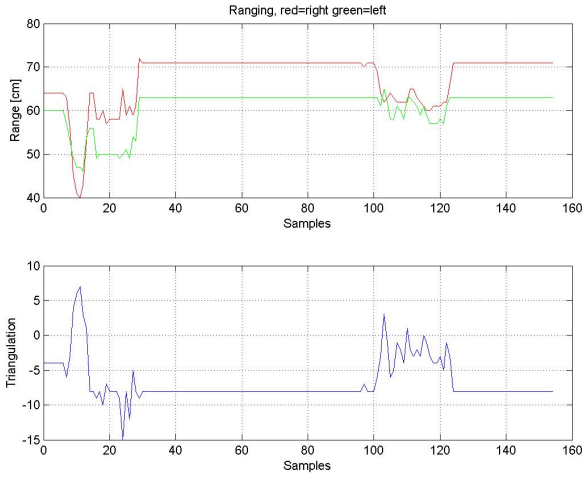
Figur 3.5: *Resultat från korsning.*

sensorer igång på ett avstånd av 130 cm riktandes rätt emot "testriggen". I denna graf syns tydligt vilken påverkan ultraljudssensorerna har på varandra.



Figur 3.6: *Resultat från ultraljudsssensorer.*

Tester i hur väl två framåtriktade ultraljudssensorer kan detektera framförvarande fordons förskjutning i sidled gjordes även. I flera tester likt Farthållar-testerna försköts fordonets position i sidled en bit. Från sensorvärderna triangulerades sedan fordonets sidoleds förskjutning med algoritmer i Matlab. Resultatet blev förvånansvärt bra, bortsett från vissa okända störningar. Resultatet antyder även att Farthållar-programmet skulle till viss del kunna kompensera för sidoleds-förskjutningar med hjälp av triangulering av ultraljudssensorerna i fronten. Figur 3.7 visar värdena från höger och vänster sensor i översta grafen. Samt trianguleringens värde i undre grafen där ett negativt värde innebär en förskjutning till vänster. Värdet visas i centimeters förskjutning. I detta testfall stod fordonet 10 cm till vänster samt 60 cm framför sensorerna.



Figur 3.7: *Resultat från triangulering.*

3.3 Plattformsmjukvara

För att kunna utföra de nya scenarierna behövdes mer utförlig kommunikation melllan MICAz modulen och Gullivers huvudkort. Kommunikation mellan de två modulerna fanns redan på plattformen, men behövde utökas med mer information som sensordata samt förmågan att kunna välja vilken sensordata som skall skickas. Till en början var tanken att Huvudkortet skulle sköta beräkningarna och MICAz modulen skulle agera radio med de andra fordonen, så ett gränssnitt för detta utvecklades och programmerades. Dock ändrades detta vid ett senare tillfälle, då det enligt målen är specificerat att det är MICAz modulen som skall sköta beräkningarna och därmed att ett nytt gränssnitt måste utvecklas.

3.4 Implementation av simulatormjukvara

Vi såg ett behov av att kunna simulera koden som utvecklades för varje scenario tidigt i projektet, dessutom behöver den simulerade koden kopplas samman med en simulation av hela Gulliver-plattformen. Resterande del av detta kapitel behandlar utformandet av hela mjukvarulösningen för simulering av plattformen. Mjukvaran kan delas upp i tre delar:

- Mobilitetssimulator
- Styrkodssimulator
- Sonarmätning-simulator

Först behandlas val av mobilitetssimulator utifrån de som presenteras i sektion 2.3, och sedan beskrivs hur samtliga delsimulationer ansluts till varandra. Sist i kapitlet analyseras olika matematiska modeller för sonarmätnings-simulatorn.

3.4.1 Val av mobilitetssimulator

För att kunna välja bland de tre alternativen av mobilitetssimulatorer som presenterades tidigare skapades en specifikatons-lista över de huvudkrav som fanns på simulatorerna,

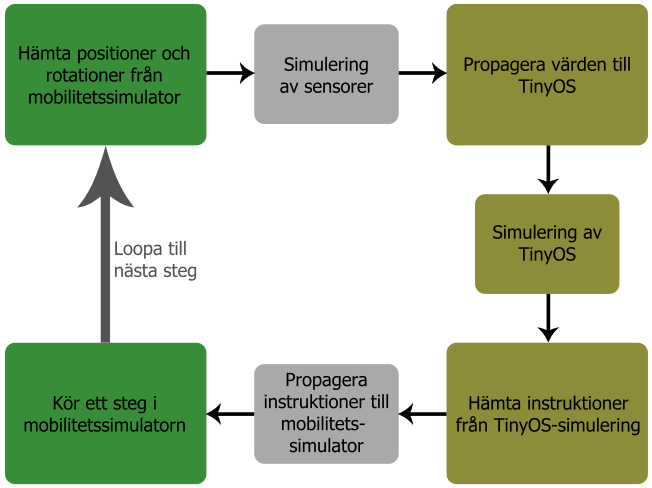
samt en uppskattning på den arbetsinsats som implementationen av varje alternativt sannolikt skulle resultera i. Kraven som definierades var:

- Två-dimensionell modellering av fordonspositioner och hastigheter.
- Kontroll över varje fordons hastighet.
- Kartdefinition och automatisk följning av väg.

Det första alternativet som förkastades var att utveckla egen mobilitetssimulation, eftersom detta skulle kräva mer arbete än vad som rymdes under projektets tidsram. Istället utvärderades de två kvarstående alternativen utefter dessa riktlinjer, det kom fram till att SUMO var det bästa valet för denna implementering. Detta baserades på svårigheterna som upplevdes med att försöka installera och exekvera OpenDaVINCI samt att ett stort arbete skulle behöva gå åt till att implementera ett nätverksgränssnitt mellan mjukvaran och skriptspråk. Med SUMO fanns istället tillgång till ett färdigutvecklat gränssnitt) mellan simulatorn och Python.

3.4.2 Sammankopplande av simulatorer

Då både vår valda mobilitetssimulator, SUMO och styrkodssimulator TOSSIM kan styras från Python valdes det att även implementera en simulationskontroller i Python. Kontrollerns uppgift är att synkronisera simuleringstiderna och överföra data mellan de två delsimuleringarna. Denna kontroller kör således simuleringssteg i en viss förbestämd frekvens och håller reda på förfluten tid för varje steg för att synkronisera både mobilitets- och styrkodsimulationen. Flödesschemat för ett helt simuleringssteg finns i figur 3.8.

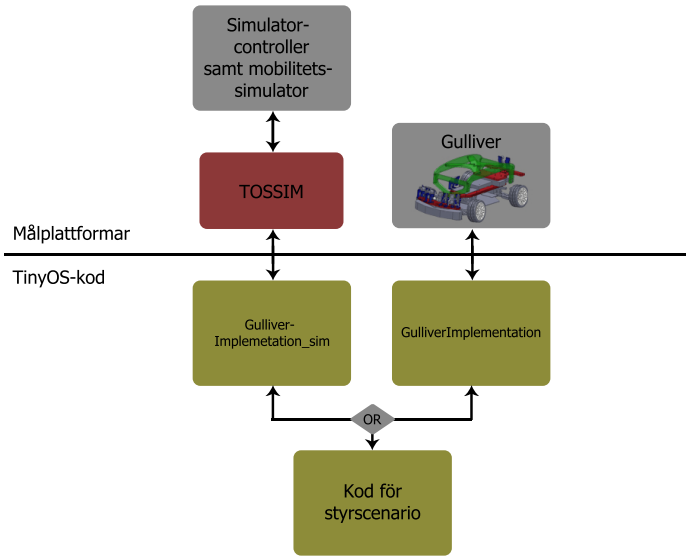


Figur 3.8: *Flödesschema för ett helt simulatorsteg i controller-skriptet.*

I början av varje steg beräknas värdena för de sensorer som förfrågades under senaste TOSSIM-steget genom sensor-simulationen, vilka sedan skickas tillbaka till TinyOS-programmet. Sedan körs stegvis TOSSIM tills dess att simulationstiden synkroniseras med den totalt förflutna tiden i kontrollern. Därefter utförs SUMO-simuleringen så att även dess tid synkroniseras med kontrollerns tid. Efter att båda simuleringarna slutförts inkrementeras kontrollerns tid med en variabel Δt. Som standard kör kontrollern simuleringsstegen i frekvensen 20 Hz, med en Δt på 50 ms, vilket gör att hela simulationen körs i realtid.

3.4.3 Kod för simulering i TinyOS

Implementationen av gränssnittet mellan TinyOS och TOSSIM kräver att en viss extrakod behöver läggas till i TinyOS-programmet som simuleras. För att spara plats bör denna extrakod endast refereras av koden hos det utvecklade scenariot vid simulering, eftersom den innehåller specifika detaljer som inte används när hela programmet laddas in i MICAz-modulen på Gulliver-plattformen vid ett fysiskt test. För att förenkla denna kodadministration valdes att skapa ett utbytbart abstraktionslager som tar hand om de högsta händelserna och kommandona i nesC-koden som används för att direkt styra fordonet eller skicka och ta emot data via trådlös kommunikation. nesC har stöd för utveckling av flera olika komponenter med samma händelser och kommandon via definitionen av ett gränssnitt (Gay et al., 2003), där alla funktioner grupperas samman. Detta gör det trivialt att skapa två versioner av abstraktionslagret som man enkelt kan byta mellan, en för simulering och en för överladdning till Gulliver-plattformen enligt figur 3.9.



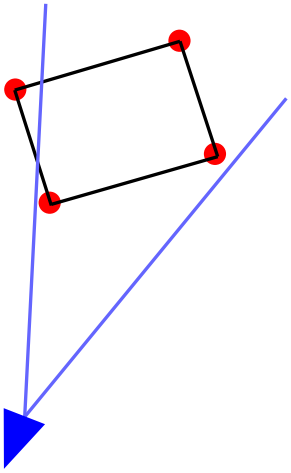
Figur 3.9: *Kodstruktur för båda möjliga målplattformar, simulering och Gulliver-plattformen.*

3.5 Modellering av sonarmätningar

För att få en god tillförlitlighet till simuleringen av Gulliverplattformen lades en stor del av den totala utvecklingstiden vid simulering av sonarmätdata. Dessa mätningar ansvarar till mycket stor del för hur bra styrscenariernas implementering kan anses, eftersom de utgör den direkta dataingången till MICAz-modulerna på fordonen. Enligt projektmålen skall modellen även kunna vidareutvecklas, så ett ytterligare mål för simulationen är att göra den så skalbar som möjligt. Eftersom värdena behöver skickas till den simulerade MICAz-modulen via TOSSIM ansåg vi att Python skulle göra implementationen enklast.

Ultraljudssensorerna begränsas delvis av ett synfält vars utbredning anges i grader, samt en maxdistans för mätningar. För sensorerna på Gulliver-plattformen approximeras synfältet till ca 45 grader, och maxdistansen till 6 meter. För att förenkla modelleringen av mätningen representeras ett Gulliverfordon som en rektangel med bredden 0.24 [m] och höjden 0.55 [m]. En bild över ett vanligt fall visas i figur Figur 3.10.

För att hitta en välfungerande modellering utifrån målen med en skalbar och robust simulator formulerades tre implementationsidéer:



Figur 3.10: *Ett fall för sensormätning där alla utom ett hörn av fordonet ligger i sensorns synfält, vars kanter visas av de blå linjerna.*

- Fysikbibliotek med stöd för strålkastning
- Egen grövre modell med avståndsmätning till fordonets hörn
- Egen generaliserad strålkastning med utbredande vågor, vågkastning

3.5.1 Strålkast

Genom fysikbibliotek får man tillgång till ett flertal användbara funktioner, bland annat strålkast (eng. ray cast) vilket kan användas för att upptäcka objekt längs med en linje definierad med en startpunkt, utbredningsriktning och längd. Biblioteket som utvärderades var Box2D, specifikt dess Python-versionen av samma bibliotek, Pybox2d vilket har just denna funktionalitet (Catto, 2011). För att modellera vågens utbredning med strålkastning kan flera strålar svepa över hela sensorns synfält genom att iterativt förändra utbredningsriktning, motsvarat av att kasta ett antal jämnt fördelade strålar mellan synfältets kanter i Figur 3.10. Antalet strålar som kastas ger en viss upplösning, där avståndet mellan två samplade punkter vid ett visst avstånd r för en sensor med synfältsvinkeln α och antalet strålkast N definieras enligt ekvation (3.1).

$$d_{samples}(\alpha, r, N) = \frac{\alpha r}{N - 1} \tag{3.1}$$

För att utvärdera strålkastningsmodellen kan man jämföra avståndet mellan samplingspunkter vid sensorns maximala avstånd, vilket för de ultraljudssensorer som är monterade på Gulliverfordonen är ca 6 meter. Man kan sätta ett maxvärde som samplingspunktsavståndet maximalt får anta vid maxavståndet, och därur beräkna hur många strålkast som måste göra för att uppfylla detta. Vi väljer det maximala punktavståndet till $d_{samples,max} = 0.2$ [m], något mindre än bredden hos ett Gulliverfordon vilket gör att vi alltid får en mätning på alla bilar inom synfältsområdet. Detta gör att vi kan beräkna det minsta antalet samplingar N_{min} genom ekvation (3.2). För värdena $\alpha = \pi/4, r = 6$ [m], $d_{samples,max} = 0.2$ [m] ges $N_{min} \approx 24$. Alltså behövs det nästan 25 samplingar per aktiv sensor för att få en godtagbar säkerhet i mätningarna. Med upp till fyra simultant aktiva sensorer per bil blir denna metod svårare att skala upp till ett större antal fordon

med godtagbar simulationshastighet jämfört med de andra implementationsidéerna, särskilt om simulatorn skall köras parallellt med riktiga fordon i framtida applikationer vilket kan ställa höga krav på skalbarheten.

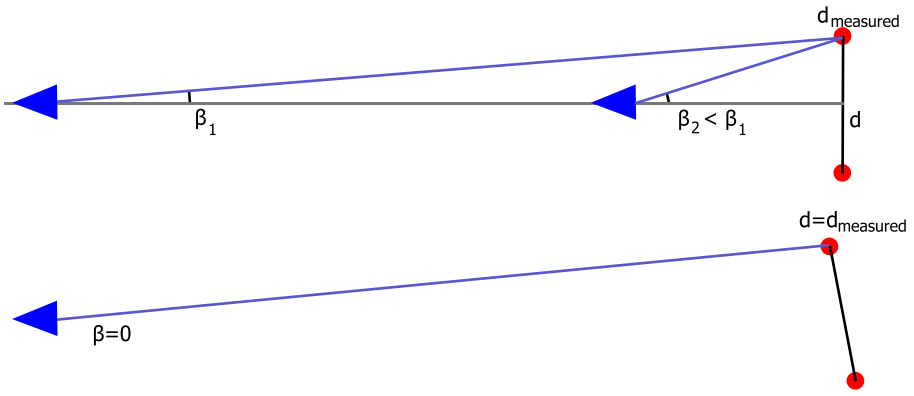
$$N_{min} = \frac{\alpha r}{d_{samples,max}} + 1 \tag{3.2}$$

3.5.2 Mätning av hörnavstånd

En grövre men betydligt snabbare modell är att endast beräkna avståndet till alla framförliggande fordons hörn, och sedan hitta det lägsta av dessa värden. På så sätt slipper man svepa ett stort antal mätningar, utan man kan istället direkt hitta mätvärden genom att endast testa om motsvarande punkt ligger innanför synfältet hos sensorn, och sedan beräkna punktens avstånd. Modellen ger en god approximation vid stora avstånd mellan sensor och mätpunkter då samtliga hörnen ligger i synfältet, enligt ekvationen för det relativa felet (3.4) då β närmar sig 0. Att notera är att felet endast uppkommer då den närmsta kantens normal approximativt pekar i motsatt riktning som vektorn mellan mätpunkten och sensorn. En liten vridning av fordonssidan skulle placera närmaste mätpunkt i ett av hörnen hos fordonet, och därmed skulle det uppskattade avståndet bli lika med det faktiska avståndet enligt ett av fallen i figur Figur 3.11. Vid mindre avstånd som vid scenariot för Farthållaren kommer dock modellen stämma sämre, eftersom vinkeln β då är betydligt större, och alltså ge upphov till ett större relativt fel.

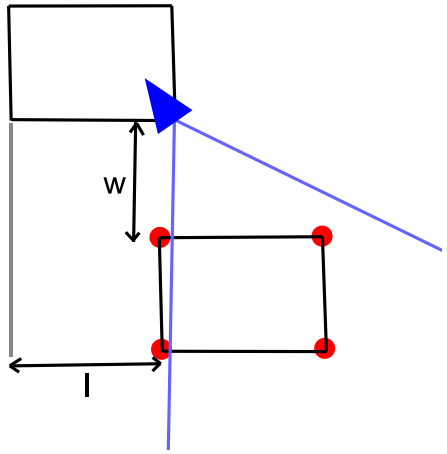
$$d = d_{measured} \cdot cos(\beta) \tag{3.3}$$

$$\left|\frac{\Delta d}{d}\right| = \frac{d_{measured} - d}{d} = \frac{d_{measured}}{d} - 1 = \frac{1}{cos(\beta)} - 1 \tag{3.4}$$



Figur 3.11: *Tre fall av hörnmätningar till de två närmsta hörnena på fordonet.*

Om ett eller flera av hörnen ligger utanför synfältet kan modellen ge betydligt större relativa fel, särskilt om bilen med mätpunkter befinner sig nära sensorn. Figur 3.12 visar ett worst-case-scenario för det relativa felet vilket skulle kunna uppkomma i ett flertal olika scenarier. Om fordonen som i figuren antas ligga en bilbredd i y-led samt en billängd i x-led från varandra kommer det relativa felet att ges av ekvation (3.5). Med insatta värden för fordonens dimensioner $w = 0.24, l = 0.55$ [m] blir det relativa felet $\Delta d/d \approx 1.5$. Det finns därmed stor risk att detta kan inverka på funktionaliteten hos det simulerade scenariot.



Figur 3.12: *Worst-case-scenario för hörnmätningsmodellen där det relativa felet närmar sig 1.5*

$$\frac{\Delta d}{d_{actual}} = (d_{actual} = w) = \frac{|d - w|}{w} = \sqrt{1 + \left(\frac{l}{w}\right)^2} - 1 \tag{3.5}$$

3.5.3 Mätning av kantavstånd

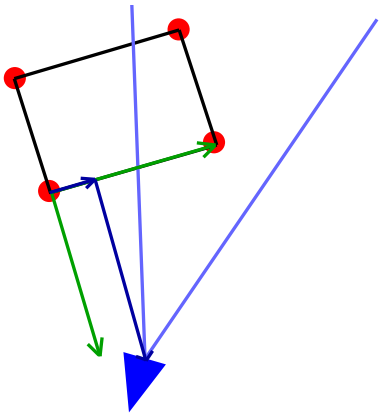
Istället för att mäta avstånden till fordonsrektangelns hörnen kan istället det kortaste avståndet till rektangelns kanter bestämmas. Detta gör att approximationen inte längre beror på mätalgoritmens säkerhet, utan istället på korrektheten hos fordonets rumsrepresentation.

Vid implementationen av denna modell antogs att fordonsrepresentationen alltid är en rektangel, detta gör att vi endast behöver betrakta en sida av rektangeln i taget. Om fordonsstukturen istället skulle modelleras som en konkav struktur skulle ett flertal kanter behöva utredas samtidigt. Algoritmen utformades så att de två närmsta hörnen för varje fordonsrektangel beräknas, och sedan testas om dessa ligger i sensorns synfält. Om åtminstone ett av hörnena ligger i synfältet, alternativt om båda kanterna av synfältet skär kanten mellan de två betraktade hörnpunkterna, betraktas den mellanliggande kanten som närmast till sensorn. Annars ersätts det näst närmaste hörnet med det tredje närmaste hörnet, och testet körs igen. Om testet återigen inte uppfylls med det nya hörnet betraktas fordonet som att det ligger utanför synfältet, och mätningen avslutas.

När den närmsta kanten beräknats skapas ett nytt koordinatsystem som beskriver positioner i kantrymden. Denna rymd definieras så att den ena basvektorn $\hat{E}$ följer kanten genom att gå från det närmsta hörnet till det näst närmsta hörnet, motsvarat av koordinat 0 och 1, respektive. Den andra basvektorn $\hat{P}$ definieras via ekvation (3.6), alltså genom att rotera $\hat{E}$ 90 grader. Då båda basvektorerna för kantrymden är definierade kan vi även definiera en matris A enligt ekvation (3.7) vilken mappar koordinater i simulationsrymden till kantrymden. Dessutom definieras ett origo C för det nya koordinatsystemet i simulationsrymden, vilket ges av positionen för det närmsta hörnet. Genom dessa två definitioner kan vi beräkna sensorns position i kantrymden $P_{edge} = A \cdot (P - C)$, resultatet visas i Figur 3.13. Punkten M för den närmaste möjliga mätpunkten på en förlängd linje av kantvektorn i kantrymden kan då bestämmas som $M_{edge} = \hat{E} \cdot P_{edge,x}$.

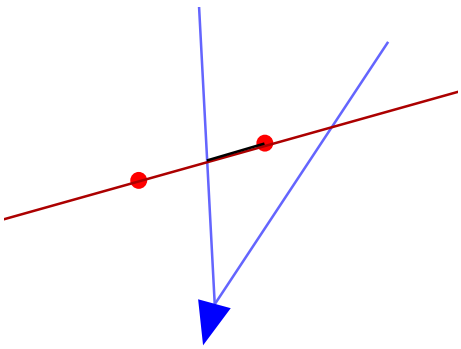
$$\hat{P} = \begin{bmatrix} \hat{E}_y \\ -\hat{E}_x \end{bmatrix} \tag{3.6}$$

$$A = \begin{bmatrix} \hat{E}_x & \hat{E}_y \\ \hat{P}_x & \hat{P}_y \end{bmatrix}^{-1} \tag{3.7}$$



Figur 3.13: *Basvektorerna i kantrymden i grönt, koordinaterna för sensorns position i kantrymden i blått.*

Eftersom den kortaste punkten på kanten kan ligga utanför sensorns synfält behöver punktens x-koordinat i kantrymden jämföras med ett intervall på möjliga värden $View$, vilket motsvarar de möjliga koordinaterna som ligger i sensorns möjliga vy. Detta intervall definieras som unionen mellan två ytterligare intervall $View = FOV \cap Edge$ där FOV är synfältets möjliga vy på fordonskantens förlänga linje, samt $Edge$ är det intervall på kantlinjen där kanten existerar. På grund av definitionen av kantrymden följer att $Edge = [0, 1]$. Gränserna för FOV hittas genom att beräkna x-koordinaten för skärningspunkterna mellan fordonets förlängda kantlinje och respektive synfältslinje. Vid vissa fall kan en av synfältslinjerna vara riktade bort från kantlinjen, men eftersom den förlängda linjen används för att hitta skärningspunkterna måste detta fall enskillt behandlas. Vid sådana fall ersätts motsvarande gräns av $\pm\infty$ där tecknet beror på vilken sida av den kvarvarande gränsen den felaktiga gränsen hamnade vid. På detta sätt kan vi även behandla de fall där synfältet endast begränsar utbredningen av det möjliga intervallet åt ett håll.



Figur 3.14: *Det möjliga intervallet (svart) för den närmsta punkten är unionen av intervallet mellan synfältets skärningspunkter och intervallet mellan kantens hörnen.*

Nu kan den närmsta mätpunkten N_{edge} inom det möjliga intervallet $View$ beräknas, genom att använda operationen $N_{edge} = clamp(M_{edge}, View)$ definierad i (3.8). Sedan kan avståndet d beräknas genom att transformera mätpunktspositionen N till simulatorrymden, $N = A^{-1} \cdot N_{edge} + C$. Nu beräknas avståndet $d = Distance(P, N)$.

$$clamp(v, Set) = \begin{cases} v, & v \in Set \\ max(Set), & v > max(Set) \\ min(Set), & v < min(Set) \end{cases} \tag{3.8}$$

4

Diskussion och slutsatser

I följande kapitel utvärderas resultaten från de tester och implementationer som utförts under projektets gång. Diskussionen delas upp i tre delar, en scenarie-del, en sensor-del och en simulator-del.

4.1 Scenarier

När styrscenarierna utvecklades visste vi inte hur mycket jobb vi behövde lägga ner på att utveckla algoritmerna och hur mycket jobb det skulle vara att implementera dem. Vi valde därför två enkla scenarier som ändå löser vardagliga problem. Dessa visade sig dock inte vara helt problemfria att implementera dels på grund av gruppens begränsade kunskap om Gulliver och dels för att sensorerna visade sig att inte vara helt pålitliga.

4.1.1 Farthållare

När Farthållar-systemet introducerades så kändes den algoritm som skulle lösa problemet som ganska enkel att implementera i plattformen. Efter en stunds fundering så förstods även möjligheterna med vad detta system skulle kunna ge för effekter när det kommer till minskat luftmotstånd, mindre energiförbrukning och i förlängningen mindre utsläpp av skadliga partiklar. Enligt en rapport gjord av trafikverket så står luftmotståndet för cirka 20-25 % av bränsleförbrukningen. Därmed skulle miljöbesparingen kunna bli relativt stor (Trafikverket).

Att börja med en ganska enkel algoritm, som mäter avståndet till bilen framför och därefter justerar hastigheten baserat på det uppmätta avståndet, kändes som en väldigt naturlig början på en lösning till problemet. Just i detta fallet fanns det redan sensorer i form av ultraljudssensorer som klarade av att mäta avståndet till framförvarande bil och var givetvis en av de grundstenar som behövdes för att lösa uppgiften.

Vidare skulle denna kunna utvecklas med en kontroll av kvalitén på informationen, utifrån detta kunna anpassa sitt säkerhetsavstånd och i förlängningen mängden utrymme kolonnen skulle kräva. Detta är något som skulle ligga utanför implementeringen i just detta arbete, men en finess som gärna skulle ses implementeras i framtida versioner av Gulliver.

Detta i sin tur kommer återigen tillbaka på att minska luftmotståndet, då bilarna kan “gömma” sig bakom varandra för att minska detta, och då samtidigt minska sin egen

bränsleförbrukning, precis som en cyklist som använder sig av sina med- eller motåkare för att uppnå samma funktion. Fenomenet kallas Slipstream och innebär att ledarbilen skapar ett undertryck bakom sig som alltså innebär ett minskat luftmotstånd till de bakomvarande fordonen.

När det kommer till sensorer och andra alternativ än sonar så har IR-sensorer verkat som ett intressant alternativ eller kanske mest intressant i samspel med nuvarande sonar-sensorer då de skulle kunna hjälpa till att motverka varandras nackdelar. En IR-sensor skulle då kunna hantera de tillfällen när sonar går bet, exempelvis när en yta absorberar ljudet från sonar-sensorn. IR-sensorers svaghet ligger istället på möjligheten att använda den när solen skiner väldigt starkt eller när miljön är väldigt ljus, då det utsända IR-ljuset kan försvinna i bakgrundsljuset. Just därför anses det att en smart lösning vore att använda sensorer av båda slagen och kanske vore det ett intressant koncept för framtida projekt. (Flynn, 1988)

4.1.2 Korsning

Korsningar är, som beskrivits tidigare, ett ganska komplicerat problem. Man behöver ta väldigt många parametrar och information i beaktande då alla fordon har olika egenskaper, som acceleration och bromskraft.

För detta projektet valdes därför begränsningen "två olika bilar som kommer från olika håll och dessa bilar kör endast rakt fram i korsningen". Detta för att det insågs att det skulle ge tidsbrist att försöka sig på något mer komplext och det är mer givande för framtida projekt av liknande karaktär att ha en stabil grund att bygga vidare på.

Hela algoritmen blev tyvärr inte implementerad i Gulliver-plattformen på grund av tidsbrist, dock var implementeringen och sammankopplingen med resten av Gulliver-plattformen är relativt okomplicerat.

Några möjliga vidareutvecklingar är en korsning med en korsningskontrollant som kan göra ett mer integrerat och effektivt trafikflöde (Vasirani and Ossowski, 2011). En annan möjlighet är att låta datorerna kopplas samman i ett Ad hoc-nätverk och genom det förhandla fram i vilken ordning de ska köra. Det finns redan en del forskning inom detta området.

Att sedan koppla samman detta med scenariot ovan, Farthållare, där vi låter flertalet bilar fungera som en enda enhet skulle kunna ge väldigt intressanta synergier och ge minskad energi-förbrukning, samtidigt som trafikflödet skulle flyta på bättre.

4.2 Sensorer

Resultatet från testerna användes preliminärt för att få ett underlag på hur ultraljudssensorerna påverkar varandra i olika situationer. Detta underlag låg grund till planerna om att expandera plattformen med ett flertal sensorer. Den intagna datan ger även en informativ kunskap om hur utvecklandet av en realistisk ultraljudssensor modell kan gå till. Ur datan undersöktes störningarnas karaktär för att evaluera möjligheten att modelera den med en statistisk modell.

Utifrån datan insåg vi vilka stora problem det finns med dessa. Att använda sig utav 6 st på ett och samma fordon skulle bli alldeles för kaotiskt. Ur testerna syntes alltid att störningarna innebar en spik på ett lägre värde. Detta för att sensorerna arbetar genom att ta emot upp till 16 ekon under ett ping. Och våran applikation samt fordonen använder sig enbart utav det första ekot vilket är det lägsta. Då störningarna alltid är lägre än det egentliga värdet innebär det en mindre säkerhetsrisk då fordonen troligtvis skul-

le deaccelerera istället för accelerera, vilket är positivt. Samtidigt finns förmodligen det sanningsenliga sensorvärdet någonstans bland de 16 ekon som ligger lagrade i ultraljudssensorn. Vilket möjligtvis skulle kunna tas fram med avancerade algoritmer för att sortera bort störningarna.

Nedan förklarar vi kort ett par möjliga lösningar på att kunna hantera det olika scenarierna, samt möjliga lösningar för att kunna simulera sensorerna.

Möjliga lösningar

- Att försöka få sonarna att aldrig hamna i en sådan situation att ljudpulserna kan kollidera är en möjlig lösning. I Farthållar-fallet kan störningarna på de främre ultraljudssensorerna i ett stort rum antas vara så små att fordonen klarar av Farthållare till viss del. För att ta in information om ett fordon finns till höger i en korsning skulle en ny sensor kunna placeras i fronten riktad 45 grader till höger. Denna sensor hade med stor sannolikhet utsattes för mycket störningar. Men hade ändå kunnat känna av om ett fordon finns i den positionen. Problemet med denna lösning är att fordonet som kommer från höger i korsningen kan utsättas för störningar och misstro att ett annat fordon finns framför. Även i ett öppet rum kommer ljudpulserna från den högerriktade sensorn att spridas runt och orsaka störningar. Ytterligare tester skulle behöva göras för att kolla hur mycket störningar fordonet till höger får utav den nya ultraljudssensorn.

- Att programmera om sensorerna så de kan skicka flera pulser efter varandra för att med specifika algoritmer hålla reda på vems pulser som är vems är en möjlig lösning Kleeman (1999). Ett fordon skulle kunna skicka sina pulser med ett mellanrum unikt för fordonet. När ultraljudssesnorn registrerar ett eko kan fordonet ta reda på om ekot kommer från sig själv genom att mäta tiden mellan ekon. Problemet med denna lösning är att ultraljudssensorerna som används sitter på ett färdigt kretskort och är mycket svåra att programmera om. En ultraljudssensor av denna modell går inte att använda på detta sätt för närvarande.

- Att göra fordonens ultraljudspulsningar beroende av varann. Fordonen skulle med jämna intervall kunna synkronisera med varann och se till att de alltid har sitt egna tidsfönster för att använda sina ultraljudssensorer på. Med en pulstid på 65 ms skulle en tid på 1000 ms kunna delas upp på 15 fordon med var sin pulsering. Problemet med en sådan lösning hade vart att fordonen har för dålig tidsupplösning på sin ultraljudssensor data. Med 3 ultraljudssensorer var och 15 bilar tar det 3 sekunder emellan varje data steg.

- För tillfället pulserar Distanskortet automatiskt alla kopplade ultraljudssensorer. En lösnings som diskuterades var att implementera funktioner i Distanskortet som kan stänga av ultraljudssensorerna. Detta skulle möjliggöra Micaz-modulen att kunna stänga av och sätta på ultraljudssensorerna vid de olika scenarierna där det behövs. Och på så sätt minimera störningarna från ultraljudssensorer som egentligen inte används. Dessa funktioner implementerades i Distanskortet men testades aldrig i hårdvaran.

- Förlita sig på andra typer av sensorer.

Möjliga lösningar för simulatorn

Störningarna som syntes från testerna var väldigt oregelbundna och varierande. Vilket gör det svårt att simulera i en simulator. Nedan följer två möjliga lösningar för att kunna simulera dessa störningar.

- Implementera en modell för ljudets utbredning i rummet samt en modell över hur fordonen skickar iväg sina pulser. En verklighetstrogen modell över ljudets utbredning i rummet samt en modell över hur fordonen pulserar sina sensorer skulle möjligtvis kunna spegla verkligheten med hög noggrannhet beronde på modellerna. Problemet är att ljudets utbredning i rummet är mycket svårt och krävande att modellera verklighetstroget. Och att få en simulator att simulera detta i realtid kan bli mycket krävande.
- Göra diverse tester med riktig data och köra simuleringen utifrån verklig data. Testerna på ultraljudssensorerna som gjordes skulle gå att fortsätta på med fordonen i rörelse och ta in all sensordata. Mellan två fordon skulle 3 stycken liknande sensor grafer kunna tas fram för varje fordon när de accelererar, har konstant hastighet och deaccelererar. En simulator skulle då kunna använda sig utav dessa kartor för att generera verklighetstrogna sensorvärden med störningar.

4.3 Simulator

Beroende på vilket typ av scenario som ska simuleras kan det vara fördelaktigt att byta mellan olika typer av modeller för både mobilitets- samt sensormätningssimulationen. För tillfället körs Gulliver-plattformen helt bundet till vägar definierade i kartklienten på servern. Det kan i framtiden även finnas ett behov av att utveckla ett alternativ där utvecklade styralgoritmer kontrollerar styrningen av bilen på en lägre nivå, exempelvis skulle rattutslaget kunna kontrolleras av algoritmer. I ett sådant fall är det fördelaktigt om mobilitetsmodelleringen enkelt bytas kan ut i simuleringscontrollern. Ett liknande fall är simulering av hundratals fordon samtidigt där sensorsmodellen skulle kunna bytas ut mot en snabbare men grövre modellering, exempelvis till den förenklade hörnmätningsalgoritmen som presenterades i sektion 3.5.2.

För att förenkla processen med att byta ut delar av simulatorn finns en möjlighet till utveckling av simulatorkontrollern. En möjlighet är att skapa ett modulärt system, liknande det abstraktionslager som utvecklades i TinyOS-delen av simulatorn, för att lätt skapa olika implementationer med ett antal förbestämda metoder vilka ropas på av controllern. Detta gör att användaren lätt kan implementera alternativa simuleringsmodeller utifrån de bestämda metoderna, och därmed trivialt byta mellan olika modeller vilka lämpar sig för olika scenariotyper. I nuläget måste delar av controllern skrivas om för att tillåta något liknande, och dessutom används mobilitetssimulatorn SUMOs inbyggda GUI för att presentera resultatet för användaren. För att skapa en helt modulär controller, där endast algoritmen för den specificerade modellen representeras i en modul, skulle det därmed behöva utvecklas ett fristående sätt att rita upp simulationstillståndet alternativt spara simuleringsdata.

Simulatorn fungerar i nuläget mycket bra för att simulera mindre scenarier likt de som utvecklades under projektet. Framförallt är det mycket enkelt för användaren att byta från simulering till kompilering till plattform via abstraktionslagret beskrivet i sektion 3.4.3. Interface-delen i abstraktionslagret förenklar även för nya utvecklare som har en viss vana

med nesC, eftersom alla möjliga metoder samlas på samma ställe, vilket ger en god översikt över vilka möjligheter man har att styra plattformen.

Litteraturförteckning

Altby, A., Boström, T., Sibgatullin, T. and Stjärne, K. (2012). Robust och noggrant positioneringssystem baserat på avståndsmätningar mellan mobila noder. Bachelor's thesis. Göteborg: Chalmers Tekniska Högskola. (Kandidatarbete inom Data- och Informationsteknik).

Arduino *Arduino*. `http://www.http://arduino.cc/en/` (Hämtad 2013-06-07).

Behrisch, M., Bieker, L., Erdmann, J. and Krajzewicz, D. (2011). Sumo - simulation of urban mobility: An overview. *SIMUL 2011, The Third International Conference on Advances in System Simulation*. Barcelona, Spain. pp. 63–68.

Berger, C. (2010). Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles. PhD thesis. Aachen: Shaker Verlag. Aachener Informatik-Berichte, Software Engineering.

Berger, C., Dahlgren, E., Grunden, J., Gunnarsson, D., Holtryd, N., Khazal, A., Mustafa, M., Papatriantafilou, M., Schiller, E., Steup, C., Swantesson, V. and Tsigas, P. (2013). Bridging physical and digital traffic system simulations with the gulliver test-bed. *Communication Technologies for Vehicles*. pp. 169–184.

Catto, E. (2011). *Box2D v2.2.0 User Manual*. `http://box2d.org/manual.pdf` (Hämtad 2013-05-14).

Flynn, A. (1988). Combining sonar and infrared sensors for mobile robot navigation. *The International Journal of Robotics Research*. Vol. 7. pp. 5–14.

Gay, D. and Levis, P. (2009). *TinyOS Programming*. Cambridge University Press.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E. and Culler, D. (2003). The nesc language: A holistic approach to networked embedded systems. *PLDI '03 Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. New York, USA. pp. 1–11.

Kleeman, L. (1999). Fast and accurate sonar trackers using double pulse coding. *I Intelligent Robots and Systems*. pp. 1185–1190 vol. 2.

Leone, P., Papatriantafilou, M., Schiller, E. M. and Zhu, G. (2010). Chameleon-mac: Adaptive and self-∗ algorithms for media access control in mobile ad hoc networks. *Stabilization, Safety, and Security of Distributed Systems*. Springer. pp. 468–488.

Pahlavan, M., Papatriantafilou, M. and Schiller, E. (2011). Gulliver: a test-bed for developing, demonstrating and prototyping vehicular systems. *MobiWac '11 Proceedings of the 9th ACM international symposium on Mobility management and wireless access.* New York, USA. pp. 1–8.

Trafikverket *Bränsleförbrukning.* `http://www.trafikverket.se/PageFiles/46572/6_luftmotstand.pdf` (Hämtad 2013-05-20).

Vasirani, M. and Ossowski, S. (2011). Learning and coordination for autonomous intersection control. *Applied Artificial Intelligence.* pp. 193–216.

Vedder, B. (2012). Gulliver: Design and Implementation of a Miniature Vehicular System. Master's thesis. Göteborg: Chalmers Tekniska Högskola.