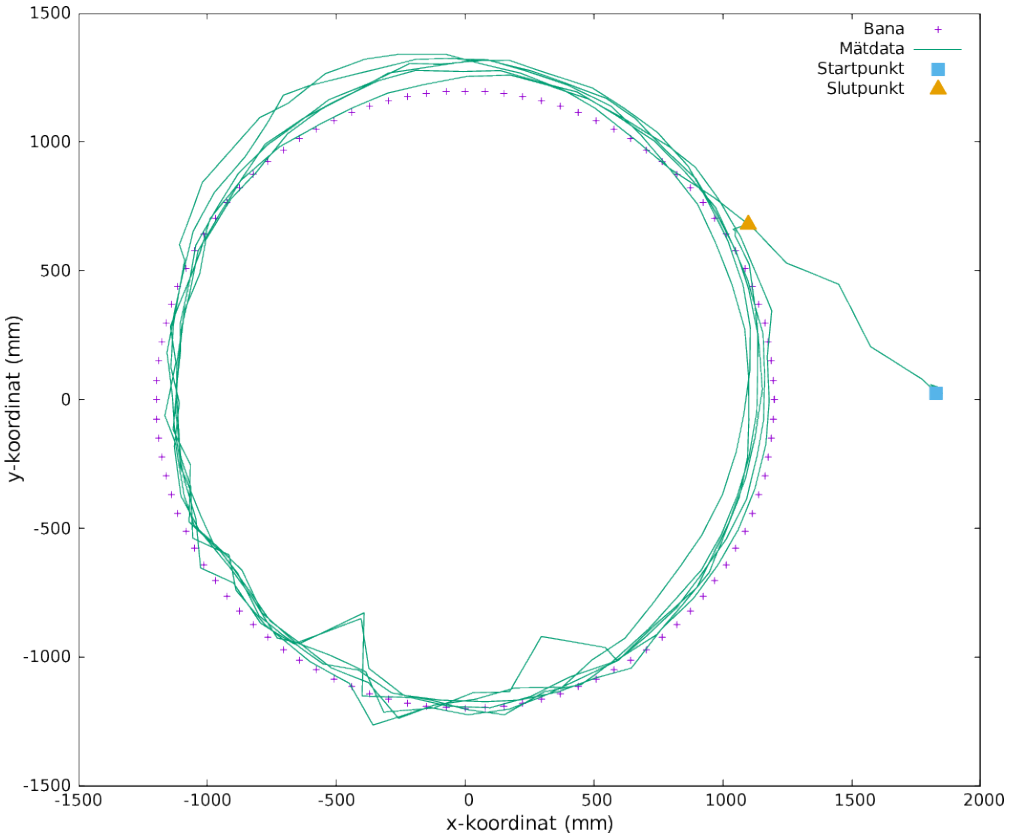
CHALMERS



Ett prisvärt alternativ för global visuell lokalisering och styrning av autonoma fordon

Kandidatarbete, våren 2016

Axel Arkheden

Robert Gustafsson

Andreas Lindhé

Romi Zaragatzky

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborgs universitet
Göteborg, Sverige, juni 2016

Kandidatarbete DATX02-16-09

Ett prisvärt alternativ för global visuell lokalisering och styrning av autonoma fordon

Kandidatarbete, våren 2016

AXEL ARKHEDEN
ROBERT GUSTAFSSON
ANDREAS LINDHÉ
ROMI ZARAGATZKY

CHALMERS

Institutionen för data- och informationsteknik
Division of Networks and Systems
Chalmers tekniska högskola
Göteborgs universitet
Göteborg, Sverige, juni 2016

Ett prisvärt alternativ för global visuell lokalisering och styrning av autonoma fordon

Kandidatarbete, våren 2016
AXEL ARKHEDEN
ROBERT GUSTAFSSON
ANDREAS LINDHÉ
ROMI ZARAGATZKY

Handledare: Elad Schiller, Institutionen för data- och informationsteknik,
Thomas Petig, Institutionen för data- och informationsteknik
Examinator: Arne Linde, Institutionen för data- och informationsteknik

Kandidatarbete DATX02-16-09
Institutionen för data- och informationsteknik
Division of Networks and Systems
Chalmers tekniska högskola & Göteborgs universitet
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: mätdata från lokaliseringssystemet under en testkörning

Ett prisvärt alternativ för global visuell lokalisering och styrning av autonoma fordon
Kandidatarbete, våren 2016
Axel Arkheden
Robert Gustafsson
Andreas Lindhé
Romi Zaragatzky
Institutionen för data- och informationsteknik
Chalmers tekniska högskola & Göteborgs universitet

Sammandrag

En prisvärd arkitektur för global visuell lokalisering och autonom styrning av fordon har utvecklats. Arkitekturens syfte att vara ett lättillgängligt alternativ för testning av autonoma fordon på liten skala eller som grund för billigare implementation av autonoma system i kommersiella lösningar. Rapporten behandlar huvudsakligen två områden: lokalisering och styrning.

Lokaliseringssystemet använder en uppsättning kameror, som övervakar ett visst område i ett rum. Kamerorna är kopplade till ett datorsystem som detekterar utplacerade visuella markörer inom området och beräknar deras position i rummet. Positioneringsdatan strömmas över ett lokalt nätverk, så att fordonet (eller andra datorer på nätverket) får del av informationen. Systemet är alltså inte begränsat till att enbart lokalisera fordon, utan kan med lätthet anpassas för positionsbestämning av andra objekt.

Delsystemet som sköter den autonoma styrningen av fordonet jämför positionsdatan från lokaliseringssystemet med en förplanerad bana, som finns lagrad i form av koordinater i en databas på fordonet. Med hjälp av den aktuella positionsdatan och banans koordinater kan fordonet i realtid beräkna aktuell avvikelse från banan samt vilka styrsignaler som krävs för att minimera avvikelsen och sätta bilen på rätt kurs. Tack vare att systemen för styrning och lokalisering är separerade kan andra system implementeras och testas med arkitekturen som grund.

För positionsbestämning av stationära objekt uppvisar lokaliseringssystemet en genomsnittlig mätavvikelse på 15 mm, jämfört med reell position. Mätresultat från upprepade testkörningar på en cirkulär bana (2,4 m i diameter) visar att fordonet hade en genomsnittlig medelavvikelse från banan på 57 mm (inklusive mätavvikelsen på 15 mm).

Nyckelord: GulliView, visuell lokalisering, autonoma fordon, global lokalisering inomhus, AprilTags, Pure pursuit, positionsbestämning.

An affordable vision-based alternative for global localization and steering of autonomous vehicles
BSc Thesis, spring 2016
Axel Arkheden
Robert Gustafsson
Andreas Lindhé
Romi Zaragatzky
Department of Computer Sience and Engineering
Chalmers University of Technology & Göteborgs universitet

Abstract

An affordable architecture for global visual localization and autonomous steering of vehicles has been developed. It is tailored to be an easily accessible alternative for testing of autonomous vehicles on a small scale or as a basis for a cost effective solution in commercial applications. The report is mainly centered around two topics: localization and steering.

The localization system utilizes an array of cameras, monitoring a certain area of a room. The cameras are connected to a computer, that calculates the position of tags it detects in the monitored area. This positioning data is streamed over a local network, so that the vehicle (or other computers on the network) can utilize the information. Thus, the system is not limited to localization of vehicles, but can with ease be modified for positioning of other objects.

The system for autonomous steering of the vehicle compares the positioning data from the localization system with a preprogrammed route, stored as coordinates in a database on the vehicle. With the current positioning data, and the coordinates of the path, the system can in real time calculate the vehicles' deviation from the path, as well as the proper control signals to minimize the deviation and set the car back on track. Because the system for steering is separate from the localization system, other systems can be implemented and tested with this architecture as a basis.

When measuring the position of a stationary object, the localization system reports the position with a mean deviation of 15 mm compared to the actual position of the object. Measurements from repeated test drives on a circular track (with a diameter of 2.4 m) show the vehicle having a mean deviation of 57 mm from the track (including the measuring divergence of 15 mm).

Keywords: GulliView, visual localization, autonomous vehicle, indoor localization, AprilTags, Pure pursuit, positioning.

Förord

Vi vill tacka Thomas Petig och Elad Schiller; våra handledare som hjälpt oss genom detta projektet och bidragit med expertis och erfarenhet för att göra vårt arbete så bra som möjligt.

Innehåll

1 Inledning **1**
- 1.1 Syfte & mål . . . 2
- 1.2 Vårt bidrag . . . 3
- 1.3 Arbetsmetod . . . 3

2 Uppgiftsbeskrivning & systemöversikt **5**
- 2.1 Kamerasystem . . . 5
- 2.2 Bana & styrning . . . 6
- 2.3 Kommunikation . . . 6
- 2.4 Utvecklingsmiljö & modellbil . . . 7

3 Teoretisk bakgrund för arkitekturens lokalisering och styrning **9**
- 3.1 GulliView – en programvara för visuell lokalisering . . . 9
- 3.2 Metod för avvikelseberäkning och styrning . . . 10
- 3.3 PID-regulator . . . 11
- 3.4 ROS – ett ramverk för automatisering . . . 11

4 Utförande & implementation **13**
- 4.1 Kamerasystem . . . 13
  - 4.1.1 Videoström & beräkning av position . . . 14
  - 4.1.2 Positionering av kameror . . . 16
- 4.2 Bana & felhantering . . . 19
  - 4.2.1 Design av testbana . . . 19
  - 4.2.2 Säkerhetsområde . . . 19
  - 4.2.3 Övervakning av position . . . 19
- 4.3 Styrning . . . 20
  - 4.3.1 Styrmetoder . . . 20
  - 4.3.2 Servostyrenhet . . . 23
  - 4.3.3 Regulator . . . 25
- 4.4 Informationsflöde & kommunikation . . . 26

5 Resultat **29**
- 5.1 Arkitektur . . . 29
- 5.2 Prestanda . . . 30

6 Diskussion & slutsats **35**
- 6.1 Val av metod för avvikelseberäkning . . . 35

6.2 Lokaliseringssystemets prestanda 37
6.3 Metoddiskussion 38
6.4 Relaterade arbeten 38
6.5 En grund för framtida forskning 39
6.6 Vår arkitektur i samhället 40
6.7 Slutsats 41

7 Litteraturförteckning **43**

Bilaga A Grafer **III**
A.1 Pure Pursuit III

Bilaga B Polulu styrenhet **V**
B.1 Styrservo V
B.2 Motorservo V

Bilaga C uEye konfigurationsfil **VII**

Bilaga D `cameraAngle.m` **XIII**

Bilaga E Byte av koordinatsystem **XXI**

1
Inledning

Autonoma förarsystem är idag ett hett ämne och många företag arbetar med automatisering av bilar. Tesla har nyligen utvecklat ett autopilotsystem som bland annat använder sig av sensorer på bilen för att undvika kollisioner [3]. Volvo ska å sin sida snart utföra ett storskaligt test kallat *Drive Me*, då 100 autonoma bilar ska köra samtidigt på Göteborgs vägar [4]. Google och BMW tillhör också de ledande företagen på området [5], och fler lär följa efter med egna liknande system.

Det finns många anledningar till varför det satsas så storskaligt på projekt om automatisering av bilar. Förutom att det är en stor lyx och bekvämlighet när det kommer till t.ex. personbilar, så är autonoma förarsystem också trafiksäkrare än mänskliga förare [6]. En annan nämnvärd anledning är att autonoma förarsystem kör mer miljövänligt än mänskliga förare [7]. Inom industrin används tekniken även för automatisering av arbetsfordon, så kallade Automated Guided Vehicles (AGV). Noggrann global lokalisering av fordon är nödvändigt bland annat i stora varuhus eller lager, då många sådana AGV:er ska manövrera samtidigt på en begränsad yta [8]. GPS är inte ett alternativ här då precisionen inomhus är dålig [9], vilket gör behovet för andra lokaliseringslösningar inomhus stort.

Testning och evaluering av nya system och funktioner inom autonom styrning är nödvändigt, och något som kan leda till långsammare utveckling är den dyra teknik som behövs för systemen. Testning av nya koncept och funktioner kräver oftast inte i första hand fordon och miljöer i full skala, utan snarare modeller i inomhusmiljö. Lokaliseringstekniker inomhus idag är ofta dyra eller oprecisa vilket är ett problem då det kommer till att hitta prisvärda system för testning eller till användning av AGV:er.

Det har redan utförts många projekt inom området för global lokalisering med hög precision. I [10] användes förpositionerade fysiska objekt som markörer. Med hjälp av dessa gjordes en karta av omgivningen och arbetsfordon kunde sedan lokaliseras med denna i kombination med sensordata. I ett annat arbete användes Wi-Fi, där signalstyrkan mättes och tillsammans med avståndsmätning avgjorde robotens position [11]. En viktig teknik under utveckling är visuell lokalisering med hjälp av kameror och visuella landmärken, vilket är både prisvärt och flexibelt. Ett projekt inom området använde en kamera på testfordonet och visuella markörer utsatta på bestämda positioner. Dessa, tillsammans med data från en Inertial Measurement Unit (IMU) användes för att lokalisera fordonet [12]. Ett annat exempel på en metod för visuell lokalisering är att använda lampor i taket för lokalisering av fordonen [13].

I Gulliver-projektet på Chalmers används speciella modellbilar som en testbädd och simuleringsmiljö för autonoma förarsystem [14]. Detta forskningsprojekt påbörjades redan år 2011 i form av en konceptidé, som lade grund för vissa kandi-

datarbeten [15] [16]. Efter detta vidareutvecklades projektet i flera kandidat- och mastersarbeten [17] [18] [19]. Vissa av dessa behandlade olika virtuella simuleringar av Gulliver-bilen och olika testscenarion.

Ett annat kandidatarbete handlade bland annat om att integrera programvaran GulliView i Gulliver-projektet [20], vilket även användes för den visuella lokaliseringen i vårt projekt. I vårt arbete används en bil som liknar Gulliver-bilen, tillsammans med lokaliseringsprogrammet GulliView, för att utveckla en arkitektur för global visuell lokalisering, och styrning av autonoma fordon.

I takt med att autonom styrning blir allt vanligare så ökar behovet av prisvärda och lättillgängliga lösningar för testning av styrning och lokalisering. Såväl forskningsprojekt som kommersiella produkter med autonoma system har användning för lättillängliga system för global visuell lokalisering, och färdiga lösningar för autonom styrning har många potentiella användningsområden.

1.1 Syfte & mål

Det är inte alltid ekonomiskt eller praktiskt möjligt att testa styralgoritmer på en bil i full skala, utan ofta finns det ett behov av att testa systemen inomhus och på mindre skala. För att positionsbestämma fordon inomhus är GPS inte en lämplig lokaliseringsmetod, då den på grund av reflektioner får dålig noggrannhet inomhus [9]. Därför behövs en annan metod för global lokalisering inomhus, som dessutom passar väl för småskaliga modeller.

En lösning som är väl anpassad för inomhusbruk är visuell lokalisering. Det är dessutom en teknik som är både prisvärd och lättillgänglig, eftersom den kan implementeras med vanliga konsumentkameror. En till fördel med tekniken är att den med lätthet kan förstärka andra lokaliseringslösningar, eftersom den är billig och inte stör andra lösningar. Styralgoritmer är inte det enda som behöver kunna testas i en laborationsmiljö, utan även andra delsystem hos autonoma fordon. Exempelvis skulle funktionalitet så som platooning (då ett fordon autonomt följer ett annat) eller hinderdetektion kunna testas, och då behövs en redan fungerande implementation av ett autonomt styrsystem. Dessutom behöver lokaliseringssystem för autonoma fordon kunna följa rörliga objekt; att ha en fungerande lösning för styrning på plats underlättar vid testning av lokaliseringssystem.

Även på den kommersiella marknaden är prisvärda system för global lokalisering intressant, eftersom det skulle öppna upp för fler användningsområden för automatiserade robotar, och fler som har kan implementera tekniken. Inom transportindustrin är den minskade arbetskostnaden jämfört med mänskliga förare, och besparingar i form av färre olyckor, en så stor vinst att det snabbt blir lönsamt med autonoma förarsystem. Men för att automatisera andra branscher är tröskeln högre, exempelvis är det en ganska liten besparing att ha robotservitörer [21] istället för mänskliga servitörer på ett café, och då blir det viktigt att systemet inte är dyrt att implementera.

Målet med vårt projekt är att utveckla en arkitektur med global visuell lokalisering för styrning och testning av autonoma fordon. Arkitekturen ska med hög precision både kunna lokalisera och styra ett fordon längs en bana. Genom att använda konsumentelektronik, som persondatorer och kameror, skall lösningen erbjuda

ett prisvärt och lättillgängligt alternativ jämfört med befintliga tekniker.

1.2 Vårt bidrag

Vi har utvecklat en arkitektur för global visuell lokalisering och autonom styrning av fordon, som genom att bygga på fri och öppen mjukvara (*FOSS*) och konsumentelektronik erbjuder ett prisvärt alternativ jämfört med befintliga lösningar.

Vår arkitektur består av dels ett lokaliseringssystem och dels av ett system för styrning av fordonet. Lokaliseringssystemet består av kameror och ett givet lokaliseringsprogram, som behövde modifieras för kompatibilitet med vår kameramodell och för en noggrann positionsbestämning. Detta program använder videoströmmen från kamerorna för att detektera markörer på marken och på fordonet, för att bestämma fordonets position. Positionen, i form av en koordinat och en riktning, publiceras sedan över ett lokalt nätverk.

Styrmodulen på fordonet lyssnar på nätverket efter positionen som skickas trådlöst från kamerasystemet. Fordonets koordinater jämförs sedan med koordinaterna i en bana, som lagras på bilen. Koordinaterna för punkter på banan genereras av ett program som skapades i detta projekt. Ett antal styrmetoder för minimering av bilens avvikelse från banan implementerades och utvärderades. Detta eftersom arkitekturen är generell och ska kunna användas för olika system med olika köregenskaper: för olika system kan olika styrmetoder vara mer passande. Kommunikationen mellan alla processer som pågår både på fordonet och på kamerasystemen uppfördes med hjälp av automatiseringsramverket ROS.

Arkitekturen har testats i ett laboratorium, med hjälp av en modellbil och en kamera på stativ, på ett sätt som ger kameran fritt synfält till bilen. Beräkningar gjordes även för utplaceringen av takmonterade kameror, för att täcka in en större yta och kunna testa med större banor. Under testningen körde bilen med en konstant hastighet i en cirkulär bana och med hjälp av dessa tester har precisionen uppmätts och antecknats. Testerna övervakades även i ett visualiseringsvektyg som utvecklades i projektet. Bilens medelavvikelse från banan uppmättes till 5,7 cm. Lokaliseringsystemets detektering av bilen då denna stod stilla hade en medelavvikelse på 1,5 cm.

1.3 Arbetsmetod

Under arbetsprocessen användes en agil arbetsmetod, då mycket av kravspecifikationerna behövde falla på plats i takt med att den teoretiska grunden lades.

Testning och verifiering av systemet skedde iterativt. Först utvecklades och testades olika program parallellt, oberoende av varandra. Sedan uppfördes och testades kommunikationen mellan två program i taget. Så småningom byggdes hela kommunikationskedjan upp och först då kunde systemet testas i sin helhet.

Testningen av hela systemet utfördes med en kamera på stativ, då kamerorna inte hann monteras i taket. Senare ställdes stativet upp på ett bord, för att simulera takhöjd, vilket gjorde att precisionen märkbart förbättrades. PID-regulatorns parametrar behövde kalibreras (som beskrivs i avsnitt 4.3.3), vilket skedde genom

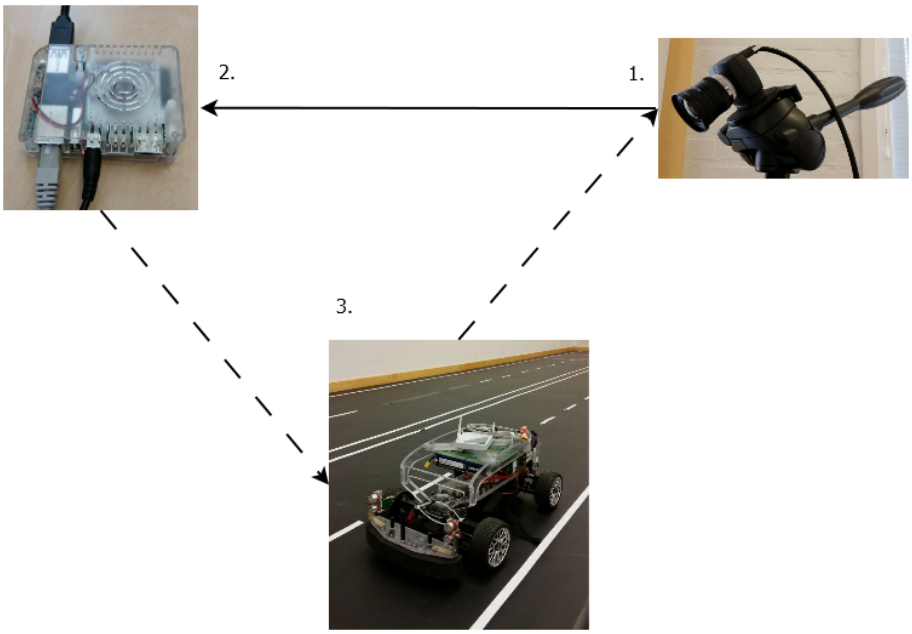
upprepade tester. Olika banor testades, men på grund av att testningen begränsades av det område som täcktes in av en kameras synfält, användes framför allt cirkulära banor. En annan begränsning var bilens svängradie, vilket gjorde att banans radie utökades så mycket som möjligt. För att inte ha en för snäv kameravinkel för lokaliseringssystemet att kunna hantera så behövde detta testas, för att minimera mätosäkerheten som med nödvändighet införs av större vinklar.

Under arbetets gång diskuterades och utvärderades olika styrmetoder för bilen. Inledningsvis användes en enkel metod för avståndsreglering, men då denna inte gav tillfredsställande resultat vid testning började andra, mer avancerade, metoder undersökas. Vissa utvärderades endast i teorin, medan andra implementerades och testades även i praktiken. Slutligen implementerades Pure Pursuit, som hade utvärderats såväl i teorin som i praktiken och resulterade i god precision och stabilitet (mer om detta i avsnitt 6.1).

2

Uppgiftsbeskrivning & systemöversikt

Uppgiften som löses i detta projekt består av många mindre steg, som exempelvis att beräkna kameramonteringen och uppföra kommunikationen mellan kamerasystemet och bilen. För att strukturera arbetet på ett passande sätt delades därför uppgiften upp i fyra deluppgifter – kamerasystem, bana, styrning och kommunikation – som beskrivs i följande avsnitt. Avsnittet beskriver även utvecklingsmiljön och de olika tekniska resurser, både hårdvara och mjukvara, som använts. All programmering av delsystemen kommer att ske i programmeringsspråket C++.



Figur 2.1: En överblick av de olika komponenterna: (1) Kameran, (2) Odroid-kortet med programmet GulliView, (3) Bilen.

2.1 Kamerasystem

En av deluppgifterna som måste lösas är att kunna lokalisera bilen. Projektet inriktar sig på visuell lokalisering med hjälp av kameror och programvara som lokaliserar bilen för att ge information om dess position. För att lösa denna deluppgift har gruppen tillgång till en iDS uEye-kamera [22] (Figur 2.1, nr 1), vilken styrs av en

Odroid XU4 [23] (Figur 2.1, nr 2). Dessutom behöver kamerornas placering beräknas för att dess synfält skall täcka in hela köområdet.

Kameran filmar bilen och analys av videoströmmen hanteras av programvaran *GulliView* [24]. Lokaliseringen av bilen sker med hjälp av *AprilTags* [25] som markörer. GulliView och AprilTags beskrivs mer detaljerat i avsnitt 3.1. GulliView behöver utökas och anpassas för att uppfylla funktionaliteten som detta projekt kräver. Det behövs stöd för kameramodellen uEye, beräkning av bilens riktning, hantering av data från flera kameranoder och en 3D-projektion av bilens koordinater.

2.2 Bana & styrning

För att skapa en bana som bilen ska kunna följa implementeras en databasmodul, där banan lagras och sedan hämtas och läses in vid körning (Figur 2.1, nr 3). Databasen ska innehålla koordinaterna till den tänkta banan som bilen ska följa. Informationen som lagras i databasen är ganska enkel och av samma datatyp, bestående av x- och y-koordinater. Det bestämdes att använda en vanlig textfil för att lagra koordinaterna, eftersom en mer avancerad databas inte har mycket att tillföra för att lagra så få olika datatyper.

Koordinaterna som finns lagrade i databasen och bilens position och vinkel, som bilen får från kameran, ska sedan användas för att beräkna bilens avvikelse från banan. Avvikelse- eller styrvinkelberäkningen ska ske i samma modul som läser in banan, det vill säga databasmodulen. Beräkningen som gjordes för att lösa denna uppgift beskrivs i kapitel 4. Avvikelsen från banan eller styrvinkel ska sedan skickas vidare till en regulator. I detta arbete används även en färdig modul för en PID-regulator som ska modifieras för att passa systemet.

Det beslutades att bilens hastighet ska hållas konstant låg under testning, för att förenkla upptäckandet av fel i systemet. Därför skall endast styrlogik implementeras, och inte någon form av dynamisk hastighetsreglering. Efter att styrinformationen passerat regulatorn skickas styrsignalen vidare till styrnings- och motorstyrenheten, som i detta arbete utgörs av en *Pololu Micro Maestro* servostyrenhet [26]. Denna ställer sedan in motsvarande vinkel på hjulen. Det finns redan en modul för hanteringen av servostyrenheten, men denna behöver modifieras eftersom en annan servostyrenhet används (detta förklaras närmare i avsnitt 4.3.2.

2.3 Kommunikation

De olika delsystemen behöver kunna kommunicera med varandra och för detta används *ROS* [27], som är ett ramverk för mjukvara till robotar. ROS arbetssätt beskrivs noggrannare i avsnitt 3.4. ROS kommer att köras på bilen, där databasmodulen, UDP-servern, PID-regulatorn och servostyrenheten befinner sig.

Kommunikationen mellan kamerasystemet och bilen sker trådlöst via nätverksprotokollet UDP. GulliView, som finns på varje Odroid-kort, agerar som en UDP-klient för att kommunicera med bilen. På bilen ska en UDP-server som tillhör GulliView implementeras. UDP-servern behöver anpassas till ROS kommunikationssystem (beskrivs i avsnitt 3.4), så att den kan kommunicera med övriga program på

bilen.

2.4 Utvecklingsmiljö & modellbil

Det rum där systemet testats är ett 5 × 10 m stort rum, med en takhöjd på 3 m. Kamerasystemet ska så småningom monteras i taket i rummet, för att maximera kamerornas synfält (se avsnitt 4.1.2). Under testningen ska ett kamerastativ användas (se Figur 2.2), som också ställs på lådor för att simulera takhöjd. I rummet finns det tillgång till ett lokalt nätverk med en trådlös åtkomstpunkt, över vilket lokaliseringsdatan från kamerasystemet strömmas.

Testbilens hårdvara består av en x86-plattform från Intel, en SSD för lagring av data och operativsystem, samt ett trådlöst nätverkskort.

Figur 2.2: Utvecklingsmiljön, med kamerastativ som är placerat på golvet. På labbrummets golv finns en bana utmarkerad, men denna användes inte i detta projekt, då enklare banor användes för testning.

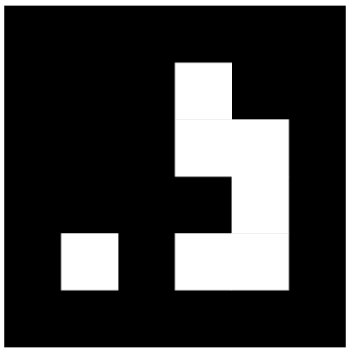
3

Teoretisk bakgrund för arkitekturens lokalisering och styrning

För att utveckla systemet krävdes kunskaper inom flera områden som var nya för gruppen. I detta avsnitt förklaras GulliView som är en programvara för visuell lokalisering, ramverket ROS som användes för att förenkla kommunikationen mellan delsystemen, metoder för att beräkna avvikelsen från banan och styrning med en enkel regler-loop.

3.1 GulliView – en programvara för visuell lokalisering

I projektet används flera komponenter som redan finns färdigutvecklade inom Gulliver-projektet, framför allt programvaran GulliView. GulliView är en programvara för visuell lokalisering, som utvecklats i ett tidigare arbete [24]. GulliView använder sig av AprilTags som markörer för att identifiera objekt i bildprocessningen [25]. AprilTags är fyrkantiga, svartvita markörer som liknar QR-koder, specialutvecklade för visuell lokalisering av robotar (se Figur 3.1). AprilTags är utvecklat av APRIL Robotics på institutionen för datateknik på University of Michigan, och är mycket väl anpassat för positionsbestämning [25].



Figur 3.1: Ett exempel på en AprilTag. I GulliView markerar denna origo i koordinatsystemet

För att bygga upp ett koordinatsystem och markera en referens i lokaliseringen så används fyra stycken markörer. Tre av dem används, varav en för att markera origo och två som referens-axlar till koordinatsystemet. Den fjärde utgör perspektivet och bestämmer lutningen på planet som referens-axlarna spänner upp. GulliView lokaliserar de övriga markörerna som objekt och räknar ut deras relativa koordinater gentemot planet.

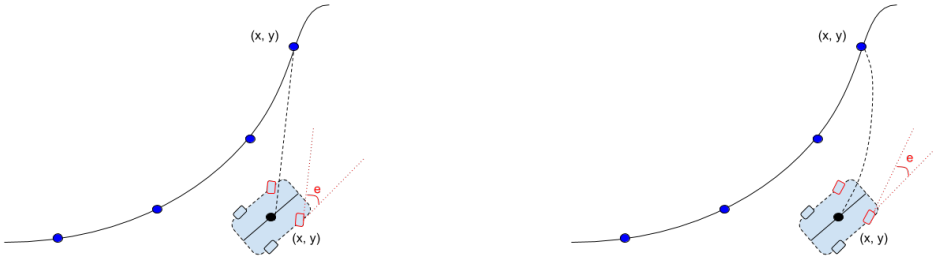
Koordinatparet paketeras och transmitteras vidare trådlöst genom nätverket. Det finns redan en en enkel UDP-server implementerad som tar emot och packar upp dessa paket.

3.2 Metod för avvikelseberäkning och styrning

För att kunna styra bilen rätt, behövs en väl genomtänkt metod för att minimera bilens avvikelse från banan. Det finns flera metoder för avvikelseberäkningen och -minimeringen. Den huvudsakliga skillnaden mellan dessa är om felet som skickas till PID-regulatorn är ett avstånd till banan eller ett vinkelfel på hjulen.

För avståndsreglering kan två olika metoder användas: avståndet som regleras kan antingen vara beräknat vinkelrätt från banan eller från bilen. Då avståndet beräknas vinkelrätt från banan, beräknas först två vektorer. Den ena går från en passerad punkt på banan till en punkt på banan framför bilen, den andra går från samma utgångspunkt till bilen. Genom att sedan jämföra vektorn till bilen med dess projektion på den förstnämnda vektorn, beräknas bilens avstånd till banan. Regulatorparametrarna testas då fram experimentellt. Den andra metoden mäter avståndet vinkelrätt från bilen till banan. För att beräkna regulatorparametrarna används här en matematisk metod som tar hänsyn till systemets fördröjning och bilens rörelsebana.

Två exempel på metoder för beräkning av vinkelfelet är Follow-the-carrot och Pure Pursuit [28]. Båda metoderna använder sig av en punkt på banan som bilen anses kunna nå vid korrekt styrning. Follow-the-carrot är en väldigt enkel metod, som innebär att bilens hjul ställs in direkt efter bilens avstånd och vinkelfel relativt punkten (se Figur 3.2a). En mer komplicerad men också mer exakt metod är Pure Pursuit. Denna metod tar även hänsyn till bilens rörelsebana då vinkelfelet på hjulen beräknas. Då bilens hjul ställs in till en viss vinkel, kommer rörelsebanan se ut som en cirkelbåge med en motsvarande radie. På grund av detta beräknas radien på cirkelbågen som krävs för att nå den valda punkten på banan och utifrån radien beräknas även den motsvarande vinkeln på hjulen (se Figur 3.2b). Då bilen närmar sig punkten, väljs en ny punkt ut som bilen ska sikta mot.

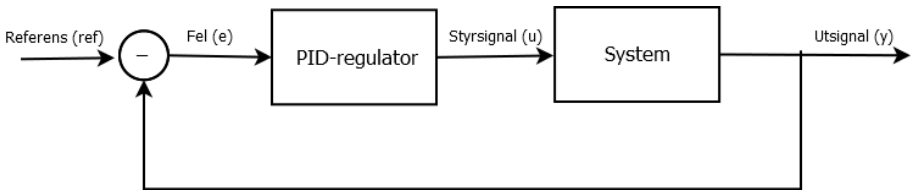


(a) Follow-the-carrot. **(b)** Pure Pursuit.

Figur 3.2: Med styrmetoden Follow-the-carrot ställs hjulen så att deras vinkel är lika stor som bilens vinkel till den valda punkten på banan. Med metoden Pure Pursuit ställs hjulvinkeln efter den cirkelbåge som bilen skall färdas i för att nå punkten på banan. I båda fallen skickas vinkelfelet e sedan vidare till styrregulatorn.

3.3 PID-regulator

För autonom styrning av bilen behövs en process som tar emot bilens avvikelse från banan och uppdaterar bilens styrning i realtid utifrån detta. Denna deluppgift ska i detta projekt utföras av en så kallad PID-regulator, som diskuterades tidigare i kapitel 2. Denna process, tillsammans med resten av systemet, ingår i en reglerloop (se Figur 3.3) som gör att styrningen uppdateras efter varje iteration. En PID-regulator tar emot ett fel, dvs. det verkliga värdets avvikelse från referensvärdet, och beräknar en motsvarande styrsignal till systemet för att minska felet [29].



Figur 3.3: Figuren visar en modell av en enkel reglerloop. En PID-regulator reglerar systemet beroende på differensen mellan referenssignalen och systemets utsignal.

PID-regulatorn består av tre delar: en proportionell del (P), en integrerande del (I) och en deriverande del (D). Var och en av dessa multipliceras med en skalär (K_p, K_i och K_d), som tillsammans kallas PID-regulatorns parametrar. En större proportionell del resulterar å ena sidan i ett snabbare system, vilket innebär att felet minimeras snabbare. Å andra sidan blir systemet mer instabilt, då större P-del ger större svängningar. P-delen eliminerar inte felet helt, utan för detta ändamål behövs en integrerande del. En stor integrerande del kan dock också resultera i ett instabilare system. För att dämpa oönskade svängningar kan en deriverande del användas, som dock gör systemet långsammare.

Parametrarna beror på metoden för avvikelseberäkningen och de kan antingen bestämmas experimentellt eller via matematisk modellering. En nackdel med den senare metoden är att matematiska modeller försummar vissa fysikaliska faktorer och de beräknade parametervärdena kan vara något oprecisa. En mer experimentell metod för parameterkalibrering är Ziegler-Nichols metod [29]. Denna metod kräver att systemet sätts i självsvängning genom att gradvis öka K_p, medan K_i och K_d är nollställda. När det korrekta K_p-värdet för självsvängning har hittats, används detta värde för att beräkna de bäst anpassade parametervärdena för systemet. Vilken metod som valdes diskuteras i kapitel 4 och anledningen till att just denna valts beskrivs i kapitel 6.

3.4 ROS – ett ramverk för automatisering

ROS står för Robot Operating System, och är ett open source ramverk och utvecklingsmiljö för mjukvara till robotsystem [27]. ROS sköter i huvudsak kommunikation mellan delsystem i större robotsystem, detta i form av meddelandehantering och -leverering som sköts av en huvudprocess kallad roscore.

Program i systemet körs som noder i ROS, där de kan kommunicera genom att publicera eller prenumerera på olika kanaler, så kallade *ROS-topics*. Varje ROS-topic arbetar med en specifik meddelandetyp, vilket kan bestå av flera primitiva datatyper. Kommunikationen över ROS-topic sker anonymt och asynkront, vilket betyder att de olika noderna inte vet när meddelandet kommer eller vilken nod som publicerat det. När en nod publicerar ett meddelande av en viss typ ser roscore till att notifiera och leverera det till alla prenumererande noder.

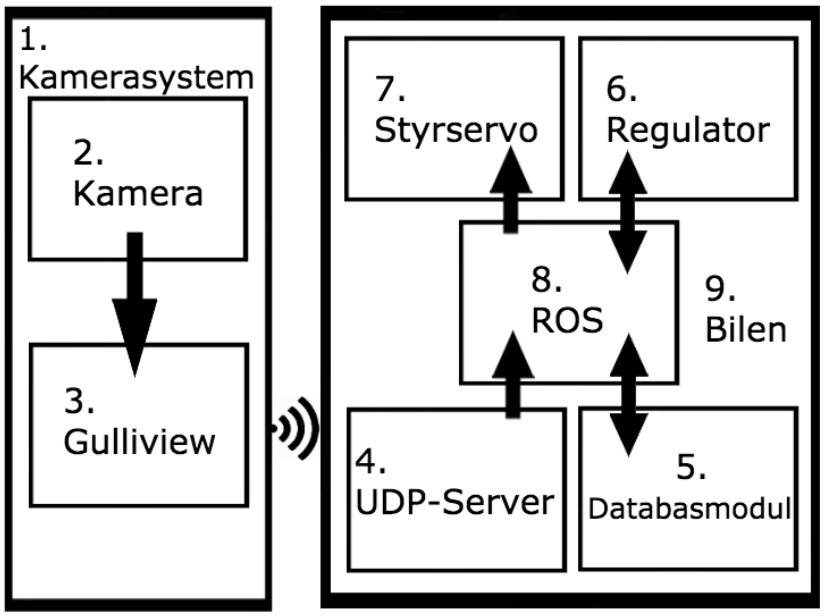
Detta leder till att delsystem med skilda funktioner kan ha sina egna program dedikerade till delsystemets specifika funktion, men fortfarande lätt och snabbt kommunicera med andra delsystem. ROS innehåller även programpaket för visualisering och testning.

Ett av de programpaket är *RViz* som används för att visualisera robotar i 3D [30]. RViz fungerar som en vanlig ROS-nod och använder sig av ROS-topics för att hämta information. Denna information kan vara position och/eller orientering för att förflytta ett objekt i det tredimensionella rummet. Detta är speciellt intressant i detta projekt då det går att representera bilen som en figur och som sedan förflyttas allt eftersom ny information om nuvarande position kommer. Genom att också rita upp en bana som är statiskt utmarkerad går det att övervaka hur noggrant bilen följer den tänkta banan.

4

Utförande & implementation

I detta kapitel tas implementationen av olika delsystem upp, och detaljer kring hur dessa är utformade diskuteras. Varje delsystem består av en till flera noder och nodernas kommunikationskanaler. Figur 4.1 visar en överblick över arkitekturen, som innehåller alla noder och informationsflödet mellan dem.



Figur 4.1: En detaljerad beskrivning av systemet. (1) Kamerasystemet. (2) Kamera. (3) Lokaliseringsprogrammet GulliView, som körs på enkortsdatorn beräknar bilens position och skickar ut positionsdatan på nätverket som en UDP-ström. (4) UDP-servern som tar emot datapaketen på bilen. Paketen skickas sedan till databasmodulen. (5) Databasmodulen. Skickar avvikelse eller styrvinkel till regulatorn. (6) Regulator för styrningen. Skickar styrsignal till styrservon. (7) Styrservon. Behandlar styrsignal och skickar denna till motorn. (8) ROS Master. (9) Bilen.

4.1 Kamerasystem

Kamerasystemet är en kombination mellan kamerornas placering i rummet och lokaliseringssystemets funktioner för att bestämma positionen på bilen. Nedan beskrivs

tillvägagångssättet för att få uEye-kameran kompatibel med GulliView (se Figur 4.1, nr 2 resp. 3), bestämma position och vinkel på bilen, samt uträkningen på antal kameror och deras placering. Detta krävs för att på ett effektivt sätt täcka arean för banan, utan att kompromissa med kamerornas förmåga att detektera markörerna.

4.1.1 Videoström & beräkning av position

Tack vare Linux enastående hårdvarustöd kan GulliView använda många olika typer av kamera, bara genom att detektera de ansluta videoenheterna under `/dev/video`. Tyvärr använder iDS uEye-kamera inte Linux-kärnans video-API (*Video4Linux*; V4L, V4L2) [31], utan stöd för dess drivrutiner behövde implementeras direkt i GulliView.

Med hjälp av kamerans dokumentation och API kunde kameran initialiseras till att hämta bilder med en upplösning på 1280×1024 pixlar med 8 bitars färgdjup (se Bilaga C). Kameran är kapabel att hämta bilder med 12 bitars färgdjup, men OpenCV-biblioteket som GulliView använder sig av endast hanterar matriser med ett djup på 8, 16, 32 eller 64 bitar [32]. Då alla matrisoperationer är beräkningstunga gör att valet att använda sig av 8 medför att det blir en mindre mängd data att hantera.

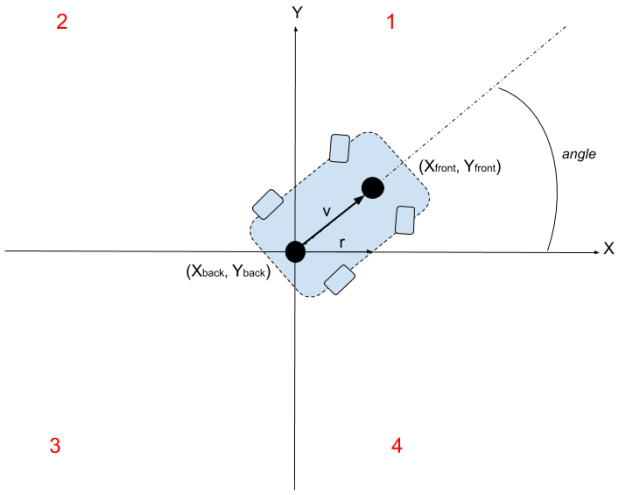
För beräkningen av bilens riktning används två stycken markörer, en främre och en bakre. Markörerna placeras på bilens tak bredvid varandra i bilens färdriktning. De två koordinaterna från detekteringen av dessa markörer används för att att skapa en vektor mellan dem. Denna vektor normaliseras för att sedan beräkna vinkeln i positiv riktning i förhållande till x-axeln. Följande beräkning görs:

$$\vec{v} = \begin{bmatrix} x_{front} - x_{back} \\ y_{front} - y_{back} \end{bmatrix}, \vec{r} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, angle = \arccos\left(\frac{\vec{v} \cdot \vec{r}}{||\vec{v}|| \times ||\vec{r}||}\right)$$

Denna beräkning ger vinkeln i första och andra kvadranten och därför behövs en jämförelse mellan bilens främre- respektive bakre markör för att avgöra om den befinner sig i tredje- eller fjärde kvadranten (se Figur 4.2). Om jämförelsen $y_{front} < y_{back}$ är sann, betyder det att bilen är riktad mot tredje eller fjärde kvadranten och då behövs följande korrigering av vinkeln: $angle = 360 - angle$.

Bilens två markeringar är monterade på taket på bilen och befinner 230 mm från marken. Men en kamerahöjd på 2617 mm och en kameravinkel på 20 grader, så kommer bilen i vissa positioner uppfattas som längre bort än vad den egentligen är. För att detta inte ska uppstå krävs en 3D-projektion av bilens koordinater, ned på samma nivå som referensmarkeringarna (se Figur 4.3) befinner sig på.

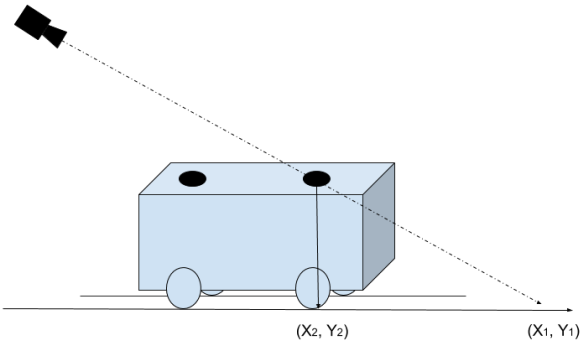
Denna projektion består av en normalvektor($s\vec{r}c$), med samma längd som bilens höjd, och går från bilens markörer på taket till marken där referensmarkeringarna är placerade. För att ta reda på vinkeln på projektionsvektorn så krävs information om hur mycket bilden är roterad i förhållande till koordinatsystemet. GulliView approximerar denna rotation genom att använda de fyra markörerna på marken för att göra en homografimatris och polärfaktorisering, vilket resulterar i en korresponderande rotations- och translationsvektor. Utöver rotationsvektorn och translationsvektorn krävs en kameramatris ($Kmat$) som innehåller kamerans brännvidd (f_x,f_y) och optiska centrum (c_x,c_y). Slutligen beräknas projektionsvektorn och dess koordinat kan avläsas, vilket även är bilens verkliga position.



Figur 4.2: Figuren visar bilens vinkel i förhållande till x-axeln. De två punkterna på bilens ovansida representerar två markörer. Vektorn $\vec{v}$ är vektorn mellan dessa markörer och vektorn $\vec{r}$ är normalvektorn till x-axeln. Siffrorna ett till fyra beskriver kvadranterna för koordinatsystemet.

$$Kmat = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \; \vec{src} = \begin{bmatrix} 0 \\ 0 \\ -height \end{bmatrix}$$

Detta sker genom OpenCV-funktionen ProjectPoints [33]. Det färdiga resultatet med detekteringen samt projektionsvektorn i GulliView kan ses i Figur 4.4.



Figur 4.3: Figuren visar en modell av bilen med två punkter som representerar markörerna på ovansidan. Kameran befinner sig uppe i det vänstra hörnet och är riktad mot bilen med en viss vinkel. Koordinaterna (X_1,Y_1) visar den detekterade positionen från kameran och (X_2,Y_2) visar bilens verkliga position.

Figur 4.4: Detta är en bild från programmet GulliView tagen i testrummet. Det finns fyra stycken markörer på golvet som markerar origo, X-axel, Y-axel samt den fjärde kallad ”Quad Axis”. På bilen finns två stycken markörer, den främre och den bakre, där den främre bestämmer position och den bakre vinkeln. Från den främre går en projektionsvektor från mitten av markören till golvytan under bilen.

4.1.2 Positionering av kameror

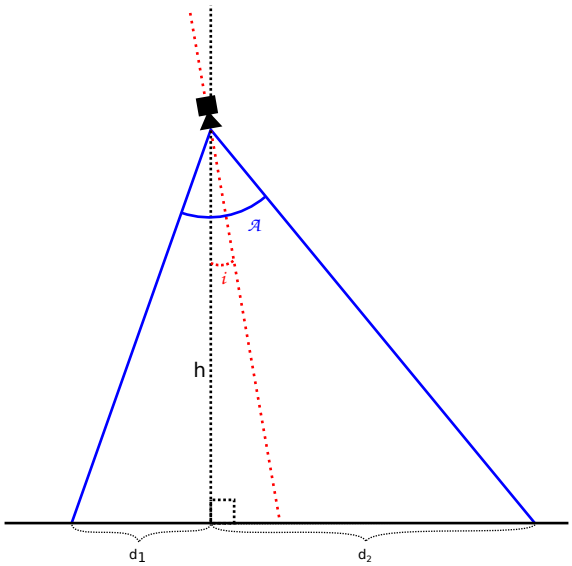
För att optimera kamerasystemets förutsättningar att korrekt kunna detektera markörerna valdes kameraplaceringen noggrant. Praktiska test visade att vid en kameravinkel (markerad i Figur 4.5 som i) nämnvärt över 20° kunde GulliView inte tillförlitligt detektera markörerna över hela kamerans synfält (se Figur 4.6). Därför valdes 20° som den största vinkel som kamerorna kunde tillåtas ha.

För att underlätta beräkningen av kamerornas placering utvecklades ett MATLAB-skript (se Bilaga D). Givet kamerans olika parametrar (sensorstorlek och brännvidd) samt vinkel och relativa höjd mot markörerna så ger skriptet det parallelltrapets som är kamerans synfält. Specialfallet 0° vinkel ger en rektangel, med samma proportioner som kamerans sensor.

Skriptet passar väl för den utrustning som projektet använder sig av, eftersom kamerans objektiv projicerar en rätlinjig avbildning. För kameror vars optik projicerar en icke rätlinjig avbildning (som exempelvis vissa billiga webbkameror, eller kameror med fisheye-objektiv) så ger skriptet inte en representativ bild av verkligheten.

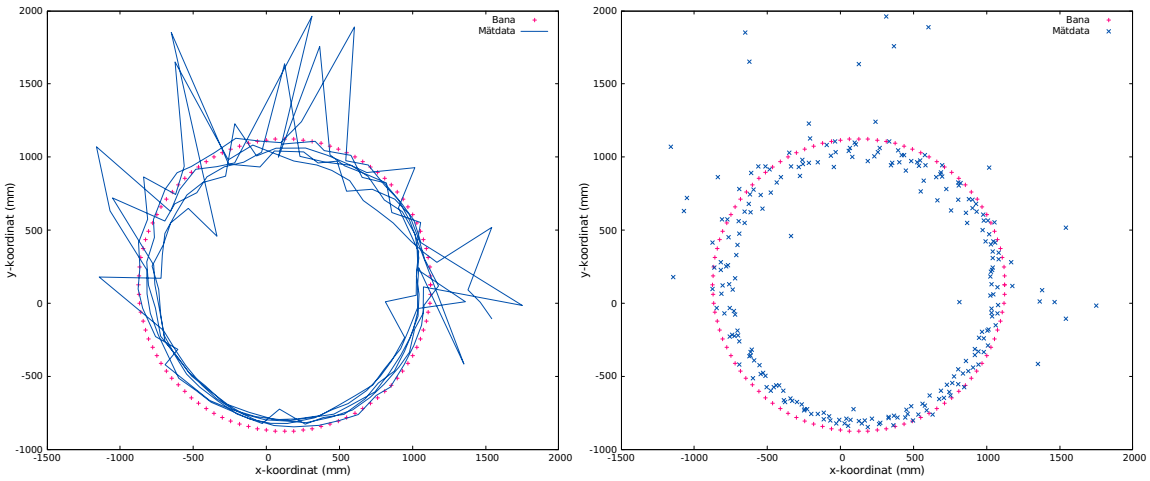
Takhöjden i rummet var 3000 mm, och höjden från bilens tak till rummets tak var 2780 mm. Kamerafästet och kameran självt skulle ta ytterligare ca 163 mm, vilket skulle lämna 2617 mm som höjd från kamerans lins till markörerna på bilens tak.

Placerade 2617 mm över betraktningsytan med 20° vinkel blir synfältet 3500 mm långt. För att täcka upp hela rummets yta blev kameraplaceringen enligt diagrammet i Figur 4.7.



Figur 4.5: Då vinkeln i är större än 0 blir kamerans synfält längre och skevat. MATLAB-skriptet `cameraAngle.m` användes för att beräkna hur mycket längre och hur mycket skevare synfältet blev vid en viss vinkel.

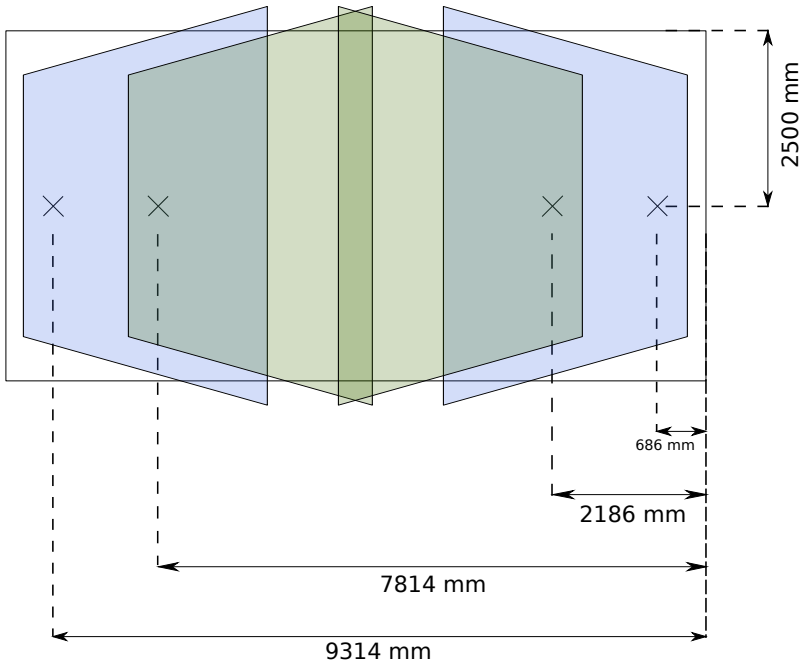
För att undvika områden där bilen inte kan detekteras av någon kamera valdes kameraplaceringen så att det aldrig skulle finnas mindre än en billängds överlappning vid övergångar från en kamera till en annan. I överlappande områden kommer kamerasystemet rapportera koordinaterna med dubbelt så hög frekvens, då samma position kommer detekteras av två olika kameror.



(a) Bilens detekterade körbana.

(b) Alla positionsdetekteringar för körningen.

Figur 4.6: Mätdata vid flera varvs körning. Vid tidiga testkörningar av systemet var kamerasystemet ej monterat i taket utan på en tripod, varför kameran behövde vinklas mycket mer än om den suttit i taket. Detta gav upphov till sämre detektering av markörerna, manifesterat som brus i övre delen av diagrammet (den del av banan som var längst bort från kameran).



Figur 4.7: Figuren visar kameraplaceringarna i testrummet. Kryssen beskriver var kamerorna är placerade och längderna är kamerornas placering i förhållande till den ena kortsidan. Varje kameras synfält visas i form av en trapetsoid och deras respektive överlappning med varandra.

4.2 Bana & felhantering

Nedan beskrivs hur banan ska skapas och lagras, vad som händer om avvikelsen från banan blir för stor och hur övervakningen av positionen sker.

4.2.1 Design av testbana

För att på ett enkelt och korrekt sätt mäta hur noggrant bilen följer den planerade banan så användes en cirkulär bana. På detta sätt kunde avvikelsen efter ett helt varv jämföras med den initiala positionen. För att generera en banan så skrevs ett mindre program som kan generera en bestämd mängd punkter, jämnt fördelade längs en randen på cirkel med en viss radie. Programmet utgår ifrån en generell ellips, där cirkel är ett specialfall. För att testa bilens egenskaper provades både ellipser, cirklar och räta linjer, men på den begränsade köryta som medgavs under omständigheterna för testen kunde endast den cirkulära banan användas på ett lämpligt sätt.

Koordinaterna för punkterna längs banan lagras i en textfil, som läses in av bilens styrmodul. Styrmodulen använder sig av positionsdata från lokaliseringssystemet för att passa vidare aktuella koordinater från banan till vald strymetod. Mer om styrningen hittas i avsnitt 4.3.

4.2.2 Säkerhetsområde

Alla välgjorda och robusta system behöver vara feltoleranta, vilket är en aspekt vid såväl testning som vid färdig produkt. Den största farhåga som kan ske vid testning är en eventuell kollision med omkringliggande väggar. För att se till att detta inte sker så ska bilen använda sig av en virtuell gräns, vilken den ej får passera. Denna gräns ska övervakas vid varje rapportering av ny position, för att förhindra en för stor avvikelse från den planerade banan. Gränsen är en rektangel som är placerad utanför banan. Det som sker om bilen skulle hamna utanför detta område är att motorn slutar driva framåt och därmed stannar. Skulle det vara en feldetektering som har inträffat så kommer motorn att börja driva igen så fort den får information om att den är tillbaka inom det tillåtna området.

4.2.3 Övervakning av position

För att övervaka var bilen befinner sig i förhållande till den planerade banan så skapades en nod som sköter kommunikationen mellan GulliView-servern och visualiseringsverktyget RViz. RViz har flera fördefinierade former på de markörer som kan användas för visualiseringen. Av tydlighetsskäl så används en pil för att visa bilens position och riktning. Banans utformning består av markörer som är punkter och de placeras statiskt på de genererade koordinaterna som utgör banan. ROS-noden får sina koordinater och vinkeln på bilen från GulliView-servern. Pilens placering startar i den koordinat som tagits emot men kräver en konvertering av Euler vinkeln för att få rätt orientering. Vinkeln som tagits emot är i grader men RViz använder sig av det fyrdimensionella talsystemet kvaternion för att beskriva orientering i rummet, därför krävs en konvertering. En kvaternion beskrivs i fyra dimensioner uttryckt

$q = w + x \times i + y \times j + z \times k$ [34]. För en rotation i x, y - planet, vilket är runt enhetsvektorn $\vec{u} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, ger en konvertering där $q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2} \times \vec{u}$. Detta ger att $w = \cos\frac{\theta}{2}$, $z = \sin\frac{\theta}{2}$ *och* $x = y = 0$.

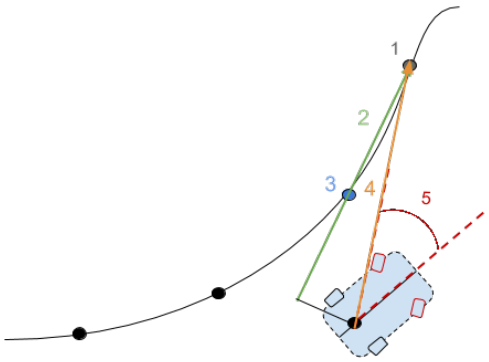
4.3 Styrning

För att få ett autonomt fordon att köra till en önskad destination eller följa en önskad bana behövs ett sätt att ge bilen riktlinjer till vart den ska och hur den ska sköta styrningen. Nedan tas det upp tre av de styrmetoder som valdes att implementeras. För att få rätt styrsignaler till motorn för styrningen används en servostyrenhet, som tar emot styrsignaler från en regulator. Servostyrenheten sköter även motorn, men som tidigare nämnt i avsnitt 2.2, skall hastigheten hållas konstant i detta projekt. Under testningen användes en låg hastighet på 0,43 $\mathrm{m\,s^{-1}}$, för att underlätta observationer av bilens beteende i olika situationer.

4.3.1 Styrmetoder

För att kunna följa en önskad bana behövs en metod för att beräkna bilens avvikelse från denna. Avvikelsen behandlas sedan av styrenheten som minskar felet och därmed styr bilen mot rätt väg. Här fanns ett flertal metoder för att beräkna bilens avvikelse och sedan styra den, och nedan beskrivs tre alternativ som övervägdes.

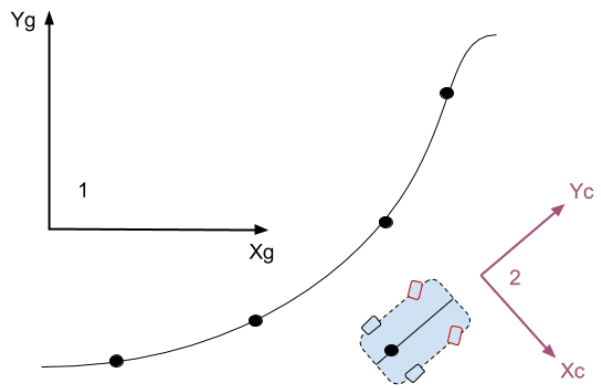
Follow-the-carrot



Figur 4.8: Bilen då den ska styra till banan genom styrningsmetoden Follow-the-carrot. (1) Bilens utvalda destinationspunkt. (2) Avstånd till destinationspunkten. (3) Punkt 2 på banan. (4) Vektor från bilen till den valda punkten. (5) Styrvinkeln för hjulen. vinkelfel och styrvinkel på hjulen.

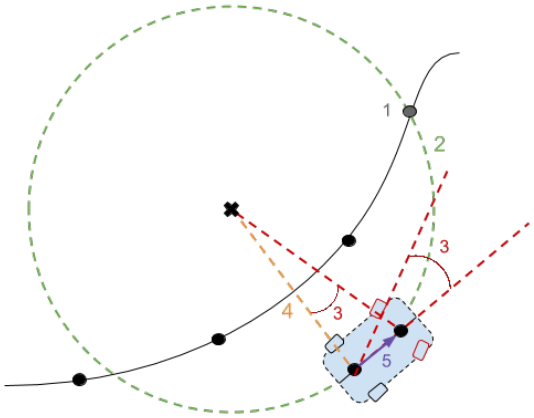
I denna metod väljs en destinationspunkt på banan som ligger framför bilen och inom dess svängradie (se Figur 4.8, nr 1), punkten bakom denna (se Figur 4.8, nr 3). En vektor ritas upp från bilen till punkten (se Figur 4.8, nr 4), och dess vinkel beräknas sedan. Vinkeln på vektorn jämförs med vinkeln på bilen, och en vinkelskillnad räknas sedan ut genom att dra bort bilens vinkel från vektorns (se Figur 4.8, nr 5). Vinkelskillnaden skickas sedan till styrenheten som skalar om värdet så att det motsvarar styrsignalen för samma vinkel på hjulen. Värdet skickas därefter vidare till styrservon. När avståndet till den valda punkten (se Figur 4.8, nr 2) är mindre än ett förvalt planeringsavstånd så väljs nästa punkt ut på banan. Avståndet mäts genom att rita upp en tangent mellan punkt 1 och 2 på banan, projicera bilens position på denna tangent och ta fram avståndet mellan destinationspunkten och bilens projicerade position.

Pure Pursuit



Figur 4.9: Bilen då den ska styra till banan genom styrningsmetoden Pure Pursuit. (1) Det globala koordinatsystemet för körytan. (2) Bilens lokala koordinatsystem.

Pure Pursuit är snarlik till Follow-the-carrot, men mer utvecklad och tar hänsyn till egenskaperna hos Ackermann-styrda fordon, såsom bilar. En punkt på banan väljs ut precis som i Follow-the-carrot (se Figur 4.10, nr 1), och en vektor ritas upp från bakaxeln på bilen till dess framaxel (se Figur 4.10, nr 5). Vektorn mellan bilens hjulaxlar går nu utmed y-axeln i bilens lokala koordinatsystem (se Figur 4.9, nr 2) och x-axeln är vinkelrät mot denna ut från bilens högra sida. Origo ligger placerad på bilens bakre hjulaxel. Koordinaterna till den valda punkten på banan räknas om till koordinater i bilens lokala koordinatsystem och en tänkt cirkel ritas upp som går igenom både origo och den valda punkten (se Figur 4.10, nr 2). Nedan hittas formeln för att gå från globala till lokala koordinater. Härledning till följande formel återfinns i bilaga E.



Figur 4.10: Bilen då den ska styra till banan genom styrningsmetoden Pure Pursuit. (1) Bilens utvalda destinationspunkt. (2) Bilens beräknade färdbana till punkten. (3) Styrvinkel för att följa beräknad färdbana. (4) Avstånd från bilens bakaxel till cirkelbanans mitt. (5) Avstånd från bilens bakaxel till dess framaxel.

$$x_{local} = (x_{goal} - x_{car})\cos(\theta) + (y_{goal} - y_{car})\sin(\theta) \tag{4.1}$$
$$y_{local} = -(x_{goal} - x_{car})\sin(\theta) + (y_{goal} - y_{car})\cos(\theta) \tag{4.2}$$

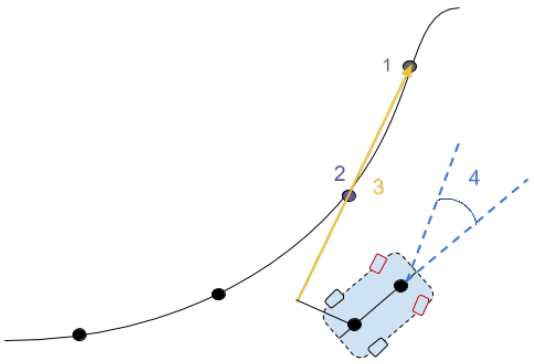
Hur radien på cirkeln beräknas härleds i ekvationerna nedan. Symbolerna motsvarar desamma i Figur 4.12.

$$d_{radius} = r_{circle} - x_{local} \tag{4.3}$$
$$(r_{circle} - x_{local})^2 + y_{local}^2 = r_{circle}^2 \tag{4.4}$$
$$r_{circle}^2 - 2r_{circle}x_{local} + x_{local}^2 + y_{local}^2 = r_{circle}^2 \tag{4.5}$$
$$2r_{circle}x_{local} = D^2 \tag{4.6}$$
$$r_{circle} = \frac{D^2}{2x_{local}} \tag{4.7}$$

En rätvinklig triangel kan sedan ritas upp som går mellan cirkelns mitt, origo och bilens framaxel. Vinkeln på triangeln vid cirkelns mitt (se Figur 4.10, nr 3) beräknas sedan med följande formel:

$$\alpha_3 = \tan(\frac{|\vec{v_5}|}{|\vec{v_4}|})$$

där siffrorna på vektorer och vinklar motsvarar desamma som i Figur 4.10. Detta är samma vinkel som bilens hjul behöver ha för att bilens bakaxel ska följa den beräknade cirkelbågen fram till den valda punkten. Vinkeln skickas till regulatorn, som skalar om den till motsvarande styrsignal för hjulen och skickar sedan vidare



Figur 4.11: Bilen då den ska styra till banan genom styrningsmetoden Pure Pursuit. (1) Bilens utvalda destinationspunkt. (2) Avstånd till destinationspunkt. (3) Punkt 2 på banan. (4) Bilens maximala styrvinkel.

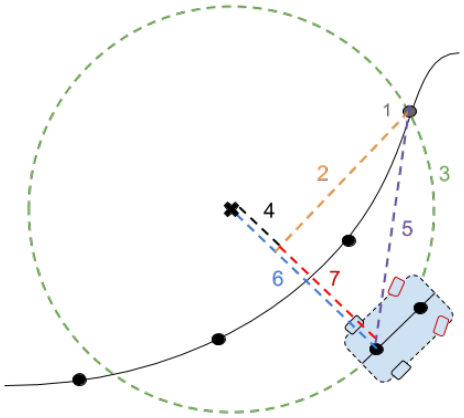
signalen till styrservon. När bilen kommit inom ett valt planeringsavstånd till punkten (se Figur 4.11, nr 2), eller om den beräknade styrvinkeln är större än hjulens maximala styrvinkel (se Figur 4.11, nr 4) väljs nästa punkt på banan och samma process upprepas. Planeringsavståndet räknas ut genom att rita upp en vektor mellan destinationspunkten (se Figur 4.11 , nr 1) och punkt 2 (se Figur 4.11, nr 3) och sedan projicera bilens position på denna vektor. Planeringsavståndet blir då avståndet mellan destinationspunkten och bilens projicerade position.

Distance Control

I Distance Control är avvikelsen istället bilens avstånd vinkelrät mot banan (se Figur 4.13, nr 5). Här väljs två punkter på banan, den närmaste som ej är passerad (se Figur 4.13, nr 1), och den föregående (se Figur 4.13, nr 4). En vektor ritas upp från den förstvalda till den efterföljande (se Figur 4.13, nr 2), och en vektor från den förstvalda punkten till bilen (se Figur 4.13, nr 6). Bilens vektor projiceras sedan på banans vektor (se Figur 4.13, nr 3), och en ny vektor mellan bilens vektor och dess projektion på banan ritas upp (se Figur 4.13, nr 5). Punktens koordinater i det globala koordinatsystemet räknas om till lokala koordinater för bilen för att bestämma vilken sida om banan bilen befinner sig på. Längden på vektor nummer 5 i Figur 4.13 är felet som skickas vidare till styrenheten, med rätt tecken beroende på vilken sida om banan bilen befann sig på.

4.3.2 Servostyrenhet

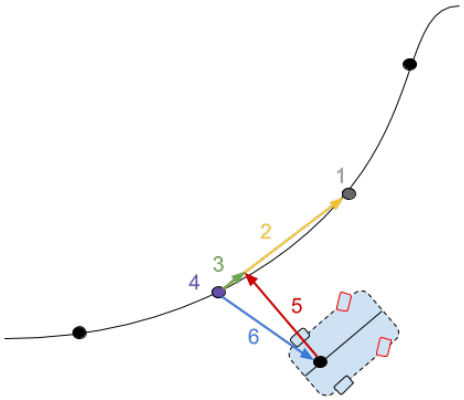
För att till en början kunna styra bilen manuellt, behövde kommunikation uppföras mellan en styrspak och servons styrenhet. Denna kommunikation gick via en pro-



Figur 4.12: Figuren visar de olika sträckor, punkter och vinklar som används vid beräkning av cirkelns radie. (1) Den utvalda punkten på banan. (2) Avståndet till punkten i y-led i lokala koordinater y_{local}. (3) Bilens beräknade färdbana till punkten. (4) Radien på cirkeln minus x_{local}, d_{radius}. (5) Avståndet från bilen till den valda punkten, D. (6) Radien på cirkeln, r_{circle}. (7) Avståndet till punkten i x-led i lokala koordinater x_{local}.

cess som översatte kommandona från styrspaken till Ackermann-meddelanden (se avsnitt 4.4) som ROS-noden för servostyrenheten tog emot (se Figur 4.1, nr 7). Det fanns redan färdiga ROS-noder för denna kommunikation, som gavs av handledaren [35] (ROS-noder beskrivs mer i detalj i avsnitt 4.4). Dessa ROS-noder behövde dock modifieras, eftersom en annan servostyrenhet (*Pololu Micro Maestro* servostyrenhet [26]) användes i detta projekt. Intervallen för vinkeln på hjulen respektive bilens hastighet finnes in Bilaga B. När kommunikationen mellan styrspaken och servostyrenheten fungerade, justerades omvandlingen av värdena från styrspaken. Dessa värden behövde förstoras och förskjutas innan de skickades till servostyrenheten och denna linjära omvandling justerades genom testning.

Eftersom hastigheten som servostyrenheten skickar till motorn inte är densamma som bilen har när den kör på golvet (på grund av friktion och andra störningar), behövde hastigheten bestämmas experimentellt då bilen kör. Anledningen till att gruppen beslutade att uppmäta bilens hastighet i m/s är att denna information kan vara intressant för andra projekt som senare kan använda sig av denna rapport för att exempelvis jämföra med sina egna resultat. Hastigheten beräknades genom att mäta sträckan som bilen kör på en viss tid. Även bilens svängradie uppmättes, liksom maxutslaget på hjulen, bland annat för att kunna avgöra hur snäv banan får lov att vara. Detta gjordes genom att ställa in bilens hjul på maxutslag (skicka motsvarande värde till servostyrenheten) och sedan köra bilen i en cirkel. Värdena som uppmättes vid testningen finnes i Bilaga B.



Figur 4.13: Bilen då den ska styra till banan genom styrningsmetoden Distance Control. (1) Punkt vald på banan. (2) Vektor från punkt 4 till punkt 1. (3) Vektor nr. 6 projicerad på vektor nr. 2. (4) Punkt på banan. (5) Vektor från bilen till banan, vinkelrät mot banan. (6) Vektor från punkt (4) på banan till bilen.

4.3.3 Regulator

ROS-noden för PID-regulatorn implementerades för att ta emot ett värde på avvikelsen (se Figur 4.1, nr 6), beräkna och returnera styrsignalen. För PID-regulatorn användes det färdiga ROS-paketet för PID-reglering [36] som modifierades för att passa den givna regler-loopen. Då avståndsreglering testades, användes alla PID-regulatorns parametrar: K_p, K_i och K_d. I detta fall behövdes K_i för att eliminera ett kvarstående fel. Då systemet sedan testades med Pure Pursuit, alltså vinkelreglering, användes endast K_p medan de andra parametrarna sattes till noll. Detta på grund av att referensvärdet i detta fall var en punkt på banan som ständigt ändrade sig, vilket gjorde de andra parametrarna meningslösa.

Parametrarna bestämdes experimentellt och hela regler-loopen testades många gånger, bland annat för att få fram bra parametervärden. Under arbetsprocessen testades flera olika metoder för styrning (se avsnitt 4.3) och varje metod medförde andra PID-parametrar för optimalt resultat. För att underlätta byte av PID-parametrarna och även för att göra systemet mer modulärt, modifierades ROS-noden för PID-regulatorn för att kunna ta emot parametrar från en launch-fil (se avsnitt 4.4). Testningen utfördes med en kamera på ett stativ, som ställdes upp på bordet för att simulera takhöjd. Senare testades även hela regler-loopen med alla kameror, då dessa hade monterats.

4.4 Informationsflöde & kommunikation

För ett komplett och fungerande system krävs förutom fullt fungerande delsystem ett sätt för dessa delsystem att kommunicera med varandra. För de delsystem som körs som olika processer på samma dator – det vill säga bilen – behövdes ett sätt att dela nödvändig information mellan varandra, men också en metod för kommunikation mellan kamerasystemet och bilen.

För kommunikationen mellan kamerasystemet och bilen användes en dataström över UDP på ett trådlöst lokalt nätverk. Detta gjordes med ett program för paketering och sändning av information på kamerasidan, och ett program för mottagning och uppackning av samma information på bilsidan. I GulliView-paketet fanns tidigare en servermodell för UDP implementerad. Denna server användes som bas när det gäller kommunikationsprotokoll och formatering av meddelanden för kommunikation mellan kamerasystemet och bilen. På bilen utökades modellen genom att inkapsla den i ett ROS-paket för att möjliggöra kommunikation med databasen. Informationen som levereras trådlöst packas upp för att sedan skapa ett ROS-meddelande från det. ROS-meddelandet publiceras sedan enligt publicera-prenumerera metoden och databasen får tillgång till informationen. En tabell med informationsflödet mellan varje nod i systemet kan ses i tabell 4.1.

Nod	**Skickad data /publiceringskanal**	**Mottagen data /prenumerationskanal**
GulliView	x,y, heading/`trådlöst nätverk`	`Videoström från kamera`
GulliView_server	x,y, heading/position	x,y, heading/`trådlöst nätverk`
Database	wanted_heading, wanted_speed/path_error	x,y, heading/position
Controller	steering_angle, speed/control_effort	wanted_heading, wanted_speed/path_error
Pololu_mc	`Styrsignal till motorn`	steering_angle, speed/control_effort
Monitor_mc	`Visualiseringsdata till RViz`	x,y,heading/position

Tabell 4.1: Tabell över informationsflödet i systemet

För att hantera flera oberoende kameraklienter med egna referenspunkter, så behöver servern veta om och hantera deras relativa position. Detta är för att övergången mellan två kamerors synfält ska fungera korrekt. Internt använde varje klient sitt eget koordinatsystem medan servern hanterade det globala, det som banan har som referens. För att göra det skalbart och inte vara bunden till unik identifiering av varje kameraenhet så byggdes logiken in i klienterna, istället för på serversidan. En uppsättning av markörer användes för att representera det globala koordinatsystemet vid origo, medan de övriga uppsättningarna placerades förskjutna relativt det globala koordinatsystemet. Varje kamera hade därför en uppsättning markeringar inom sitt synfält för att bygga upp ett koordinatsystem. För att hantera den relativa förskjutningen markeringarna emellan, mättes denna upp och beskrevs i form av för-

skjutning i millimeter i x- och y-led. Den uppmätta informationen användes sedan vid start av kameranoderna, vilket innebar att de olika kamerorna använde sig av sitt eget koordinatsystem inom sitt synfält, där sedan rätt uppmätta förskjutning lades på i efterhand så att alla kameranoder använde samma globala koordinatsystem. Detta medför att servern inte behöver identifiera vilken kameranod informationen kommer ifrån utan endast packa om meddelandet.

För att kunna övervaka bilens position i realtid så användes visualisering med ROS-verktyget RViz. För att ROS-noder ska kunna kommunicera med varandra behöver de – utan extra konfigurering – köras på samma dator som masternoden. Alla delsystem implementerade som ROS-noder körs på bilen, men då RViz i detta fall behövde köras på en annan dator och samtidigt kunna kommunicera med bilens delsystem fick en exportering av RViz's master nod ske. Detta innebar att genomföra konfigureringar av ROS's miljövariabler på den dator där RViz kördes, så att den kommunicerade med en icke lokal masternod. Detta gjordes genom att sätta ROS miljövariabler enligt:

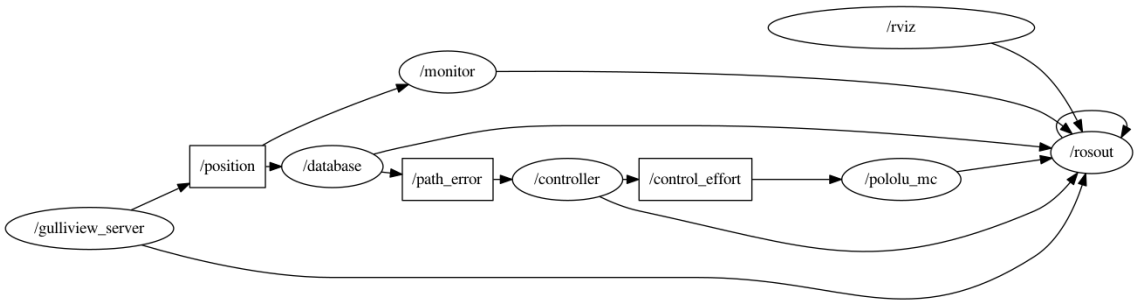
```
export ROS_MASTER_URI=http://bilensip:portnummer
export ROS_IP=egetip
```

5
Resultat

En arkitektur har utvecklats för global visuell lokalisering av ett autonomt fordon som följer en planerad bana. Vid testning uppmättes en medelavvikelse på 57 mm (se Figur 5.4), i förhållande till den cirkulära körbanan (som hade en diameter på 2,4 m). I detta avsnitt beskrivs först arkitekturen som åstadkommits och sedan den uppnådda prestandan gällande bland annat precision och mätbrus.

5.1 Arkitektur

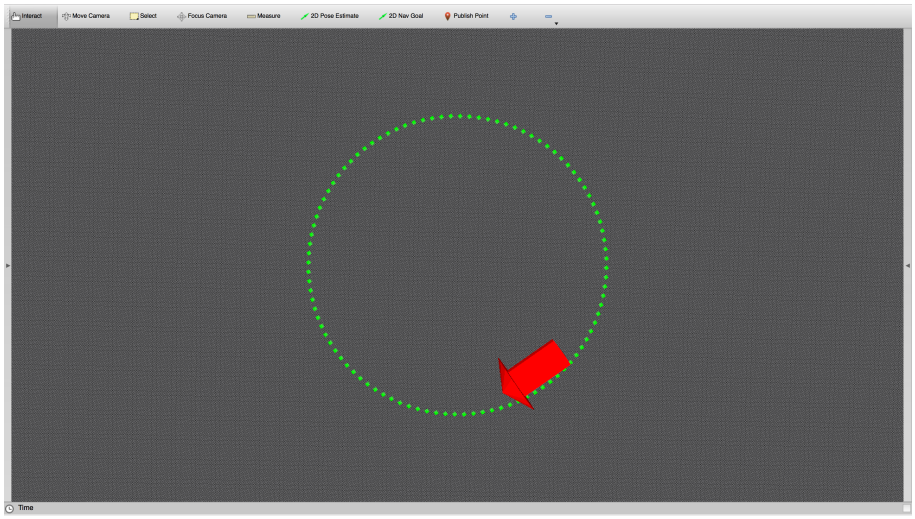
Den framtagna arkitekturen består övergripande av ett system för lokalisering och ett system för autonom styrning av en modellbil. Ett av delsystemen skulle kunna bytas ut vid behov och det kvarvarande skulle fortfarande fungera om en likvärdig motpart som uppfyller vissa krav. Systemet med alla ROS-noder på bilen är oberoende av vilken metod för lokalisering som används, och finns avbildat som en graf i Figur 5.1. Skulle till exempel visuell lokalisering inte fungera för ett speciellt användningsområde kan kamerasystemet bytas ut mot någon annan typ av lokalisering, så länge det ger styrsystemet information i form av fordonets globala koordinater och fordonets vinkel. För att förenkla hanteringen av alla delsystem och noder som ska köras, så finns en körbar fil som startar alla de noder som krävs på bilen.



Figur 5.1: Grafen visar noder och kanaler som utgör systemet som befinner sig på bilen. De ovala symboliserar noder och rektanglarna är kanaler. Pilarna visar dataflödet i systemet.

Till arkitekturen hör ett verktyg för att visualisera och övervaka bilens rörelser. Verktyget består av ROS-paket RViz som hanterar grafiken för visualiseringen, samt en ROS-nod för att läsa av data om bilens position och banans position. För att på ett enkelt och tydligt sätt visualisera detta så modelleras bilen som en pil, som pekar i samma färdriktning som bilen. Banan i sin tur består av en mängd efterföljande

punkter som även skiljer sig i färg jämfört med pilen som modellerar bilen (se Figur 5.2). Positionen kommer ifrån samma prenumeration som databas-noden får sin information ifrån och banans koordinater hämtas från samma datafil som databasen läser av. Detta betyder att RViz är lätt att använda med övriga delsystem, vilket gör att det kan köras trådlöst på en annan enhet, så länge de är på samma lokala nätverk samt har ROS och RViz installerat. Det gör att en operatör till systemet inte behöver befinna sig inom synhåll av bilen utan kan övervaka vad som sker på avstånd.



Figur 5.2: Figuren visar en skärmdump av programmet RViz. Pilen är modellen för bilen och banan visas i form av punkter.

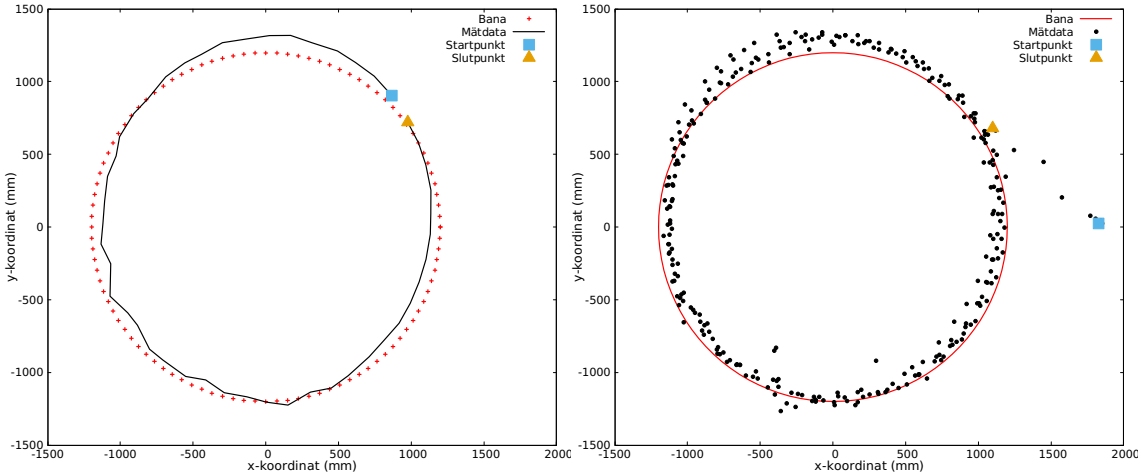
5.2 Prestanda

Flera olika aspekter av testsystemets prestanda har testats såsom precision, mätbrus och bilens förmåga att hantera svåra utgångspositioner. I detta avsnitt presenteras olika grafer som på olika sätt demonstrerar resultaten av testerna inom de olika aspekterna.

Testning av systemets precision skedde genom upprepade körningar av modellbilen på raka, cirkulära och ovala banor. Banan som de flesta testerna gjordes på är en cirkel med radie 1,2 m och startar i origo. Noggrannheten på körningen mättes genom att ta bilens uppmätta koordinater och jämföra dem med den planerade banans. I testerna användes styrmetoden Pure Pursuit och en P-regulator med parametern $Kp = 40$. I Figur 5.3b vid körning 6 varv kan vi se att testbilen följer cirkeln väl, med en medelavvikelse på 57 mm och en standardavvikelse på 6,9 cm.

Figur 5.4 demonstrerar hur bilens avvikelse från banan varierar över tid, tillsammans med medelavvikelsen. Här kan även den största avvikelsen observeras, men det är uppenbart att bilen inte befunnit sig där utan att det är en feldetektering av kamerasystemet.

Med denna styrmetod och en cirkel som bana så blir den största delen av avvikelsen på insidan av cirkeln, vilket kan ses som en större frekvens för negativa x-värden



(a) Ett enskilt varv

(b) Alla positionsdetekteringar under körningen.

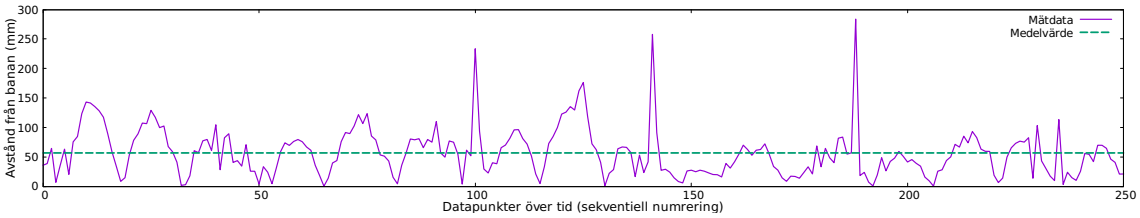
Figur 5.3: Mätdata vid flera varvs körning. Fyrkanten är startpunkten och triangeln är slutpunkten. Kameran är placerad i taket, nästan rakt ovanför banan. Med kameran monterad i taket kan en så liten kameravinkel användas att mätdata blir nästan helt fri från brus. Som en konsekvens av att kameran fortfarande är lätt vinklad återstår endast några få feldetekteringar som återfinns i nedre delen av kamerans synfält, se figur 5.3b.

i histogrammet i Figur 5.5. Att avvikelsen är på insidan av cirkeln kan bero på flera olika faktorer, varav den mest framstående är metoden för avvikelseberäkningen som används, Pure Pursuit. Denna metod (som förklarades i avsnitt 4.3.1) använder sig av ett så kallat ”look-ahead”-avstånd till nästa punkt på banan som resulterar i att bilen väljer nästa punkt innan den hunnit fram till den tidigare valda. På grund av detta ”genar” bilen aningen konstant.

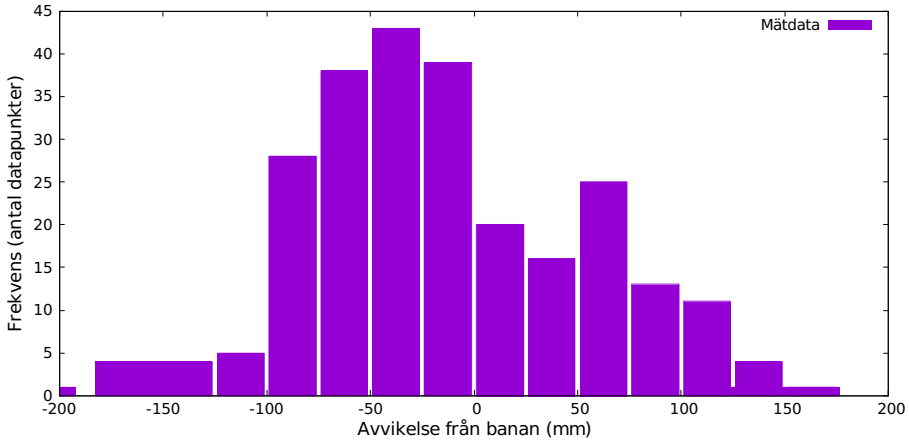
En medelavvikelse med 57 mm från banan vid körning är noggrant med tanke på att mätbruset gör att lokaliseringen av bilen vid stillastående har en maximal avvikelse runt 5 cm (Se Figur 5.7). Utspridningen av detektionerna i koordinatsystemet kan ses i Figur 5.6.

Lokaliseringssystemet kan använda en eller flera kameror, och strömmar bilens positionsdata över det lokala nätverket. Med uEye-kameran kopplad till Odroid-systemet så rapporterar GulliView detektering av markörer med en förhållandevis jämn frekvens på 2,4 Hz, se Figur 5.8.

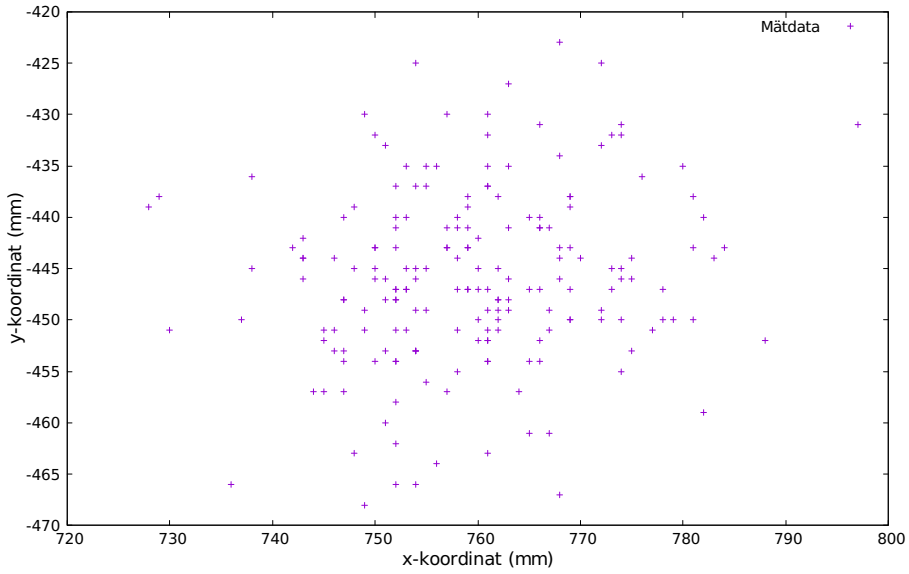
Systemet kan även klara av svåra situationer, då bilen har en svår utgångsposition. Ett sådant exempel kan ses i Figur 5.9a, då bilen startade från en position som var motriktad den planerade banans riktning och ändå klarade att reglera styrningen. Bilen lyckas styra upp till och följa banan efter drygt ett halvt varv, vilket väl demonstrerar systemets förmåga att hantera svåra utgångspositioner.



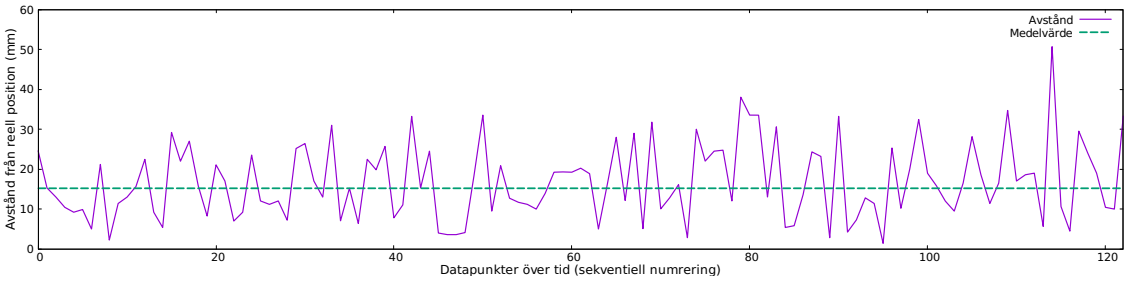
Figur 5.4: I grafen syns avvikelsen från banan över tid samt en horisontell linje som markerar medelvärdet på avvikelsen från banan. Medelvärdet ligger 5,7 cm ifrån referensvärdet, vilket delvis beror på de tre feldetekteringarna som utmärker sig som spikar nedåt i grafen.



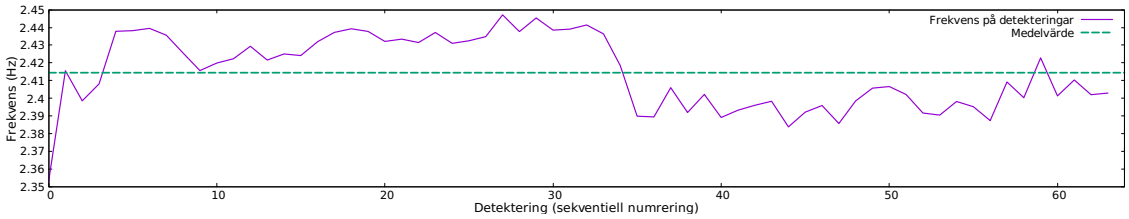
Figur 5.5: Histogram över avvikelse från banan. De mätpunkter med en avvikelse på mer än −200 mm är de tre punkter orsakade av mätbrus som syns på insidan av cirkelns nedre kant i Figur 5.3b. Notera att den största frekvensen av felen är negativa, vilket betyder att det är innanför cirkeln som de noterats.



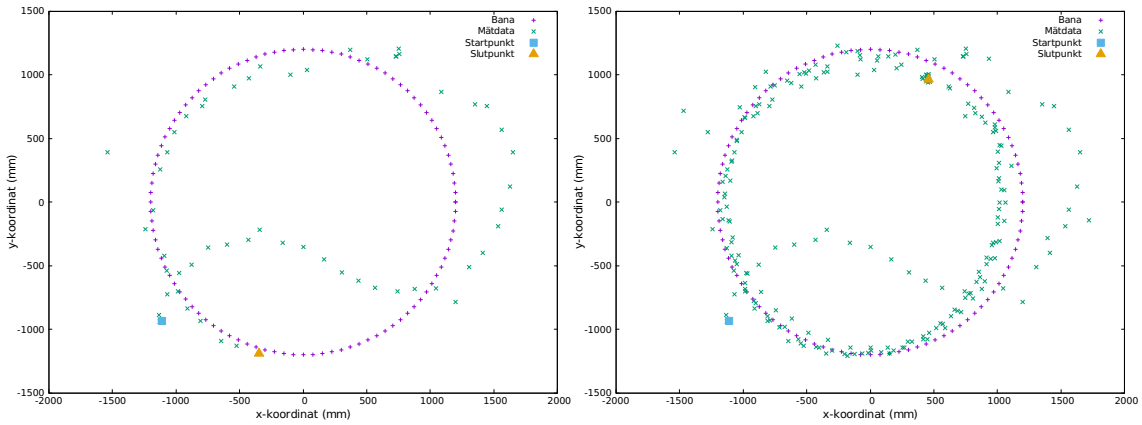
Figur 5.6: Mätbrus från detekteringen med kamerasystemet, då bilen stod stilla. Punkterna är detektioner som gjordes under testningstiden.



Figur 5.7: I grafen syns mätbrus över tid, då bilen stod stilla. Den genomsnittliga avvikelsen från faktisk position var 15 mm, markerat i figuren.



Figur 5.8: Uppdateringsfrekvens (Hz) på GulliViews detekteringar av bilen. Grafen visar även att de första detektionerna tar längre tid, vilket beror på de extra beräkningarna för golvets markeringar (som endast görs i början).



(a) Första varvet. **(b)** Alla varv.

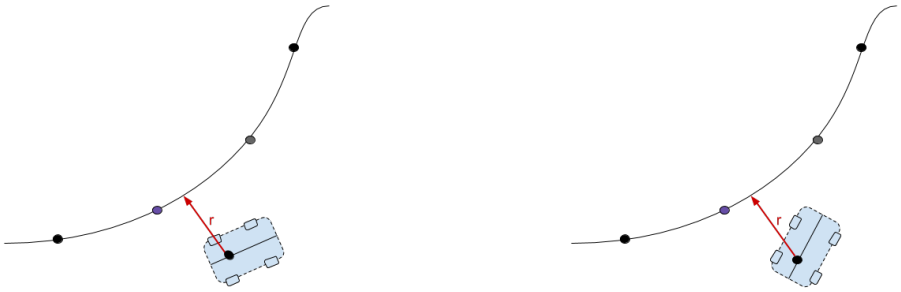
Figur 5.9: Bilen startar på utsidan av banan, riktad motsatt den programmerade körriktningen. Trots den osympatiska utgångspunkten har bilen redan efter lite drygt ett halvt varv korrigerat den initiala störningen och börjat följa banan. Under alla varv därefter följer bilen banan mycket väl.

6
Diskussion & slutsats

Under arbetets gång togs flera beslut som påverkade det slutliga resultatet. De avvägningar som gjordes diskuteras i följande avsnitt. Här jämförs även projektet och dess resultat med andra arbeten som gjorts inom samma område, som även nämndes på ett mer allmänt plan i kapitel 1. Till slut beskrivs även områden för framtida forskning, det vill säga kommande projekt som skulle kunna använda sig av denna arkitektur som en av byggstenarna för utveckling av nya produkter.

6.1 Val av metod för avvikelseberäkning

I kapitel 3 beskrevs flera metoder för att bestämma bilens avvikelse från den planerade banan. I kapitel 4 beskrevs sedan att det beslutades att använda metoden Pure Pursuit, som genererar bilens avvikelse i form av en styrvinkel. Detta beslut togs först efter att alla de ovannämnda metoderna diskuterats teoretiskt och även vissa testats i praktiken.



(a) Avståndsreglering, fall 1. **(b)** Avståndsreglering, fall 2.

Figur 6.1: Figurerna visar avvikelsen i en bestämd punkt för styrmetoden avståndsreglering. Figur 6.1a och Figur 6.1b ses samma avvikelse men med olika vinkel mot banan.

Till en början användes en väldigt enkel metod, där reglerfelet var avståndet till bilen vinkelrätt från banan, även kallat Distance Control i denna rapport. Avståndet växlade tecken beroende på vilken sida av banan bilen befann sig på. PID-regulatorns parametrar bestämdes experimentellt. Denna metod fungerade, men var

ibland instabil då den inte tog hänsyn till bilens vinkel relativt banan, se Figur 6.1. Som figurerna visar kommer styrsignalen vara lika stor i både 6.1a och 6.1b, men då bilens vinkel relativt banan skiljer sig stort mellan fallen kommer oscilleringar och inexakthet bero väldigt mycket på utgångsposition och tillfälligheter. Ett annat specialfall hittades då bilen råkar hamna riktad åt fel håll gentemot banan – eller startar så – och därför följer den åt fel håll. Här sågs många fördelar med Pure Pursuit över avståndsreglering, då Pure Pursuit till skillnad från det som beskrivits ovan tar hänsyn till bilens vinkel relativt banan och alltid följer banan i önskad riktning. Det tack vare att Pure Pursuit siktar på en punkt på banan och avståndsreglering bara jobbar med avvikelsen från denna.

Relativt tidigt noterades att kamerasystemet gav en ganska låg frekvens av detekteringar – en frekvens på ca 2,4 cm – vilket diskuteras mer i avsnitt 6.2. Detta ledde till ännu en fördel för Pure Pursuit. Vid val av banor med snabba förändringar får avståndsregleringen problem, då avvikelsen från banan ändras dramatiskt mellan uppdateringar vilket leder till stora fluktuationer i bilens körbana. Pure Pursuit är bättre anpassad här, då den använder sig av ett planeringsavstånd för att planera sin rutt. Med en beräknad kurvatur till en punkt en liten bit fram på banan rör sig bilen i en mjuk sväng mot den valda punkten. Denna planerade rörelsebana leder till att Pure Pursuit klarar en låg uppdateringsfrekvens av positionsdatan bättre än avståndsregleringen.

Då beslutet togs att testa att byta ut avståndsregleringen mot vinkelreglering, användes först metoden Follow-the-carrot (se avsnitt 3.2). Nackdelen med denna metod är att den ställer in en betydligt större vinkel på hjulen än nödvändigt (se Figur 3.2a). Detta resulterade i svängningar som visserligen kunde hanteras med bra värden på PID-regulatorns parametrar. Det bestämdes ändå efter en teoretisk analys att Pure Pursuit var en bättre modellerad metod, då denna även tar hänsyn till bilens rörelsebana (se avsnitt 3.2). Follow-the-carrot kan fortfarande vara bra för fordon med annorlunda egenskaper än Ackermann-styrda fordon, och lösningen valdes därför att behållas i slutprodukten som alternativ styrmetod trots att den inte är optimal för just vår bil.

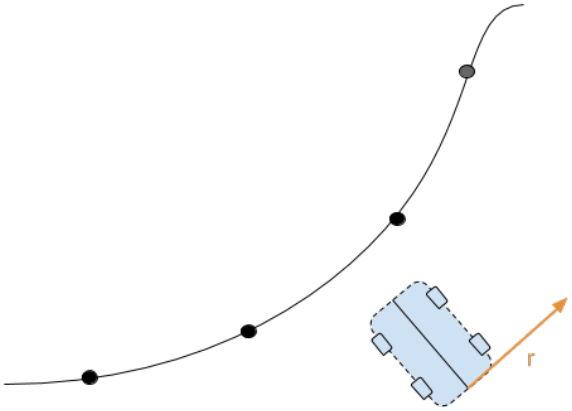
(a) Fall 1. **(b)** Fall 2.

.

Figur 6.2: Med avancerad avståndsreglering kan vi se att avvikelsen r är densamma i båda figurerna.

En annan metod som diskuterades, men inte testades i praktiken, var att använda

avståndet vinkelrätt från bilen som reglerfel, se Figur 6.2a. En förbättring från den förra metoden var att avståndet nu berodde även på bilens vinkel relativt banan. Denna metod användes i ett tidigare projekt, där även Kp-värdet beräknades matematiskt utifrån modellen [37]. I det arbetet togs även systemets fördröjning och bilens rörelsebana med i beräkningarna. Denna metod övergavs dock efter att några specialfall upptäcktes, som krävde komplicerade lösningar. Ett av dessa var som i den enklare avståndsregleringen, att felet kan bli lika stort med olika vinklar på bilen, se Figur 6.2.



Figur 6.3: En utvecklad version av avståndsreglering. Vektorn r ger avvikelsen från banan.

Ett annat specialfall var då bilen körde vinkelrätt ifrån banan, se Figur 6.3. I detta fall är avståndet oändligt och systemet skulle behöva specialanpassas för det.

6.2 Lokaliseringssystemets prestanda

För att ta hand om nivåskillnaden mellan bilens markering och referensmarkeringarna så krävdes som sagt en 3D-projektion. Detta problem kunde adresseras på två olika sätt, antingen genom att projicera referensmarkeringarnas koordinater på bilens plan, eller genom att projicera bilens koordinater på referensmarkeringarna. Den senare metoden valdes, för att minimera beräkningarna som enkortsdatorn Odroid behöver göra. Det visade sig att kamerans detekteringar inte var tillräckligt exakta för att korrekt identifiera markeringarna och rotera bilden. Detta gav upphov till en spridning av de detekterade koordinaterna runt den faktiska positionen (se Figur 5.6). Detta fenomen upplevdes som mätbrus av bilen och ville minimeras så mycket som möjligt, vilket visade sig vara då metoden att projicera bilens koordinater användes. Ett ytterligare steg för att minska effekten av mätbruset hade varit

att implementera ett Kalmanfilter, vilket är en vanlig metod för att hantera mätbrus från exempelvis GPS-system genom att vikta rapporterade koordinater mot tidigare position.

Alla matrisberäkningar som görs av GulliView på Odroid visade sig vara beräkningstunga och det enkeltrådade programmet kom att belasta en tråd i processorn fullt ut. Då referensmarkeringarna är fyra stycken och bilen använder endast en för att bestämma position så minskade antalet beräkningar något. Detta är en fördel för oss som undersöker arkitekturen med en bil men det kan finnas fördelar att göra på det andra sättet vid en uppskalning av antal objekt som ska detekteras.

6.3 Metoddiskussion

Arbetsmetoden fungerade överlag bra för den form av utveckling som gjordes i detta projekt. Gruppen arbetade kontinuerligt med utvecklingen av systemet och fördelade arbetstiden och bördan väl under arbetets gång.

Att vi tidigt började utvecklingen av systemet gjorde att vi hade tid att testa och utvärdera olika tekniker, som var fallet med exempelvis de olika styrmetoderna. Detta hade troligtvis varit svårare att hinna med om arbetet hade börjat med en utförlig fysikalisk modellering av bilen och dess rörelsebana. Å andra sidan hade en bättre teoretisk modellering möjligtvis resulterat i bättre noggrannhet och stabilitet. Just i detta arbete räckte dock mer experimentella metoder för att uppnå tillfredsställande resultat.

Det var även viktigt att beräkningen av kamerornas placering blev klar tidigt, då vi hade förståelse för att monteringen kunde ta tid. Tyvärr hjälpte inte detta då kamerorna inte hann bli monterade innan projektets slut ändå. Gruppen hade dock planerat arbetet väl och började testa hela systemet vid den tidpunkt som hade planerats, fast med bara en kamera på stativ. Ytterligare ett sätt att testa och utvärdera systemet, som i detta projekt inte användes på grund av tidsramarna, hade varit att använda två kameror på stativ. Detta hade hjälpt att täcka in en större yta och samtidigt att testa de två koordinatsystemens samordning.

6.4 Relaterade arbeten

Det bedrivs mycket forskning inom området för global lokalisering, och även om flera projekt tidigare har producerat delvis liknande resultat så är detta projekt unikt i sin prisklass för den precision som uppnås.

För att på ett rättvist sätt kunna jämföra olika lösningar måste man självklart ta i beaktning den budget som krävs för att faktiskt implementera den framtagna lösningen. Som en följd av detta väljer vi att jämföra lösningars implementationskostnad genom att dela in i två kategorier: specialutvecklade lösningar och konsumentelektronik.

En annan viktig detalj för att kunna jämföra resultatet av olika arbeten är detaljer kring testmetoderna; specifikt är fordonets hastighet, och frekvensen med vilken positionen rapporteras, något som inte tas upp i många av de relaterade arbeten

vi har studerat. Vi väljer ändå att direkt jämföra resultaten med våra uppmätta resultat, utan att ta hänsyn till eventuella skillnader i hastighet.

Ett arbete vars resultat hade relativt hög precision var [8], vilket är ett projekt som använde specialutvecklad utrustning. I projektet användes utsatta reflektorer i ett stort varuhus för att lokalisera arbetsfordonen. Kartan och reflektorernas positionering dimensionerades efter fulla hyllor, och därför blev resultaten med tomma hyllor och utan hyllor något sämre än resultaten med fulla hyllor. Det genomsnittliga felet var i det bästa testfallet ca 5 cm och i värsta fall 7 cm. Dessa resultat är i storleksordning med våra resultat under körning (se Figur 5.3), men ungefär fyra gånger sämre än våra resultat vid stillastående mätning (Figur 5.7).

I det projektet användes SICK NAV 350 lasrar, vilka ligger i hundratusenkronorsklassen, på varje fordonsenhet. Vidare behöver reflektorer nogsamt placeras ut, och i stora industrilokaler kan det röra sig om flera tusen reflektorer, så det rör sig om en betydligt större implementationskostnad än för den arkitektur som utvecklats i detta kandidatarbete.

I andra arbeten har avståndsmätning genom *Radio Frequency Identification* (RFID) använts, då svarstationer har placerats ut under golvytan, respektive i taket i ett laboratorium [38] [39]. Varje svarsstation har ett unikt ID-nummer som roboten identifierar då den passerar under eller över svarsstationen. I kombination med odometri nådde resultaten en hög precision med en genomsnittlig avvikelse på ca 1,5–3 cm, beroende på bland annat antalet svarsstationer som användes. I miljöer med skymd syn, som i mörker eller dimma, är RFID mer pålitligt då visuell lokalisering kräver öppet synfält mellan kameran och bilen. I vanliga ljusförhållanden är dock visuell lokalisering en tillräckligt pålitlig metod.

RFID är inte en lika lättillgänglig konsumentvara som kameror, men inte heller en lika dyr specialutvecklad teknik som exempelvis en uppsättning laserskannrar med tillhörande reflektorer. Kostnaden för RFID-tekniken beror till stor del på antalet RFID-taggar som behövs för en viss yta, vilket varierar mycket mellan olika tekniker.

En teknik som är billig att implementera och använder sig av konsumentelektronik är [11]. I projektet används signalstyrkan hos Wi-Fi-åtkomstpunkter (assisterat av odometri och fysiska begränsningar) för att lokalisera en robot som rör sig genom korridorerna i en universitetsbyggnad. Denna metod är både billig och lättillgänglig, men samtidigt resulterar den i ett genomsnittligt fel på 120 cm, vilket är betydligt sämre än vad som demonstreras i vårt arbete, se kapitel 5.

6.5 En grund för framtida forskning

Arkitekturen som utvecklats i detta kandidatarbete kan lägga grund för kommande arbeten som behöver ett globalt lokaliseringssystem med hög precision. Ett exempel på ett sådant projekt är implementationen av så kallad platooning, då ett eller flera fordon följer efter och håller konstant avstånd till ett annat ledande fordon. På detta sätt kan flera fordon följa efter varandra och bilda ett fordonståg. Det ledande fordonet kan då styras och testas med ett system som bygger på arkitekturen som utvecklats inom detta kandidatarbete. Platooning är ett aktuellt forskningsområde: Volvo deltog nyligen i European Truck Platooning Challenge, genom att köra tre Volvo FH lastbilar från Göteborg till Rotterdam i Nederländerna [40].

Andra exempel är projekt som behandlar koordination av flera robotar som rör sig i olika banor på en begränsad yta eller manövrering av drönare inomhus. Vår lösning gör det möjligt att styra varje robot/fordon separat i dess egen bana. Detta kan vara aktuellt vid koordinering av AGV:er (*eng.* Automated Guided Vehicle) på stora lager eller varuhus, som förklarades i kapitel 1. Arkitekturens höga precision är också viktig i sådana sammanhang. I Singapore planeras det redan i vissa restauranger att använda sig av servitörer i form av drönare [21]. För sådana system krävs förutom detektering av hinder även ett globalt lokaliseringssystem med god precision, vilket vår arkitektur erbjuder.

Arkitekturen kan även appliceras i system för privat bruk, som exempelvis för hushållsrobotar som dammsuger eller tvättar golv, eller utför andra sysslor som kräver en global lokalisering med hög precision. Robotar som dammsuger golv existerar redan idag, men dessa använder sig oftast av sensorer för att lokalisera sig [41]. Ett globalt lokaliseringssystem kan möjligtvis optimera dessa robotars arbete genom att ge möjlighet för bättre planering av rörelsebana, så att hela golvet täcks på effektivaste sätt. I sådana system kan det även vara bra att kombinera den globala lokaliseringen med detektering av hinder.

6.6 Vår arkitektur i samhället

Systemarkitekturen som utvecklats i detta projekt lämpar sig väl för lokalisering med hög precision av exempelvis autonoma truckar i lagerlokaler. En möjlig anledning till att många lager inte redan har liknande förarlösa system implementerade beror oftast på de höga kostnaderna för de befintliga lösningarna [8]. Autonoma system med ett mer precist och samtidigt prisvärt alternativ för global lokalisering skulle kunna resultera i ökad effektivitet och färre olyckor [42].

En nackdel är att eftersom noggrannare prisvärda lokaliseringssystem leder till ökad automatisering av arbete, leder detta till att fler arbeten går förlorade. Detta på grund av att arbetaren kan ersättas av en robot som gör samma jobb men är mycket billigare i drift och också pålitligare [43]. Lagerarbetare var den 8:de största yrkesgruppen i Sverige 2014 med drygt 73 000 anställda [44] och om automatiseringen utvecklas kan det leda till en märkbart ökad arbetslöshet. Samtidigt skapas det nya jobb inom automatiseringsindustrin, och med ökad produktivitet inom ett fält så öppnas det ofta upp för nya kategorier av jobb i andra delar av samhället.

Vår arkitektur kan även användas för testning av autonoma personbilar inomhus, då exempelvis vädret inte tillåter testning utomhus. Detta skulle leda till vidareutveckling av självkörande bilar som snart kommer att fylla våra vägar. Det finns många fördelar med att automatisera körningen, då många trafikolyckor orsakas av den mänskliga faktorn [45]. Samtidigt finns det många säkerhetsrisker, speciellt under övergångsfasen då autonoma fordon ska färdas på vägarna bland bilar med mänskliga förare. Under denna övergångsperiod finns stora risker för missförstånd och exempelvis ögonkontakt, som idag är en viktig del utav kommunikationen mellan förare, försvinner helt i vissa fall. Alla sådana detaljer måste tas i åtanke vid utvecklingen av självkörande fordon.

6.7 Slutsats

I detta kandidatarbete utvecklades en arkitektur som svarar mot projektets syfte och mål. Arkitekturen är ett prisvärt alternativ för global lokalisering och autonom styrning av fordon, som kan appliceras såväl för testning av autonoma fordon inomhus som vara en attraktiv lösning för inomhuslokalisering på marknaden. Lösningen bygger på vanlig konsumentelektronik, som kameror och persondatorer, vilket gör arkitekturen prisvärd. Samtidigt visar resultaten som demonstrerades av testsystemet på en hög precision, vilket motsvarar kravspecifikationen.

Arkitekturen testades med en mindre testbil i en labbmiljö på en plan golvyta utan hinder och en kamera på stativ, som senare även ställdes upp på ett bord för att simulera takhöjd. Inom det område som täcktes in av kamerans synfält, körde bilen längst en förbestämd cirkulär bana, vars diameter utökades så mycket som möjligt för att kompensera för bilens svängradie. Även lutningen på kameran anpassades så att ett så stort område som möjligt täcktes in av synfältet, samtidigt som lokaliseringsprogrammets detektering av bilen skedde så felfritt som möjligt. Bilens hastighet hölls konstant och ganska låg, för att förenkla testningen och upptäckandet av fel i systemet som behövde åtgärdas.

Vid denna testning uppmättes en medelavvikelse från bilen till banan på 57 mm. Lokaliseringsprogrammets detektering av bilen då denna stod stilla hade en medelavvikelse på 15 mm och detekteringsfrekvensen var 2,4 Hz. Systemet visade sig även klara av svåra utgångspositioner på bilen, då denna exempelvis startade i motsatt riktning till banan.

Systemet utvecklades och utvärderades för det mesta genom experimentella metoder och en mindre vikt gavs åt fysikalisk modellering av bilens rörelsebana. Detta gav oss möjlighet att hinna testa och jämföra flera styrmetoder för att till slut välja Pure Pursuit, som gav högst precision och stabilitet. En bättre fysikalisk modellering hade möjligtvis förbättrat precisionen något, men det ansågs att precisionen som uppnåddes var tillräckligt hög för detta projekt. Det hade även varit intressant att testa systemet med takmonterade kameror, men detta hanns inte med inom projektets tidsramar, som förklaras i avsnitt 6.3.

Arkitekturen har en precision som kan anses hög relativt befintliga lösningar inom samma prisgrupp. Lösningen har många användningsområden, inom industrin likaväl som för privat bruk, med exempel givna i avsnitt 6.5. Den kan även användas som en grund för framtida projekt som behöver ett välfungerande globalt lokaliseringssystem med hög precision för att utveckla nya intressanta produkter.

7

Litteraturförteckning

[1] "Fork of ap1," https://github.com/lindhe/datx02-16-09-ap1.

[2] "Fork of visionlocalization (gulliview)," https://github.com/lindhe/datx02-16-09-visionlocalization.

[3] "Tesla's autopilot system," 2015. [Online]. Available: https://www.teslamotors.com/sv_SE/blog/your-autopilot-has-arrived

[4] "Volvo's drive me test." [Online]. Available: http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/intellisafe-autopilot/drive-me

[5] "Bmw's self-driving car parks itself and picks you up when you're ready to go," 2016, http://www.forbes.com/sites/aarontilley/2015/01/06/bmws-self-driving-car-parks-itself-and-picks-you-up-when-youre-ready-to-go/#f8f096c1348f.

[6] "Tech Times," http://www.techtimes.com/articles/123214/20160112/study-says-self-driving-cars-are-safer-than-human-driven-vehicles-should-you-believe-it.htm.

[7] J. Pyper, "Self-driving cars could cut greenhouse gas pollution." [Online]. Available: http://www.scientificamerican.com/article/self-driving-cars-could-cut-greenhouse-gas-pollution/

[8] P. Beinschob and C. Reinke, "Advances in 3D data acquisition, mapping and localization in modern large-scale warehouses," in *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*. IEEE, sep 2014, pp. 265–271. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6937007

[9] M. B. Kjærgaard, H. Blunck, T. Godsk, T. Toftkjær, D. L. Christensen, and K. Grønbæk, *Pervasive Computing: 8th International Conference, Pervasive 2010, Helsinki, Finland, May 17-20, 2010. Proceedings.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. Indoor Positioning Using GPS Revisited, pp. 38–56. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12654-3_3

[10] L. B. Wong, Lisa; Evan Graham, Andrew; Goode, Christopher W.; Waltz, "Method and apparatus for using pre-positioned objects to localize an industrial vehicle," jun 2015. [Online]. Available: https://www.google.com/patents/US9056754

[11] J. Biswas and M. Veloso, "WiFi localization and navigation for autonomous indoor mobile robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4379–4384, 2010.

[12] D. Zachariah and M. Jansson, "Fusing visual tags and inertial information for indoor navigation," *Record - IEEE PLANS, Position Location and Navigation Symposium*, pp. 535–540, 2012.

[13] J. Bell, Mark; Chandrasekar, Kashyap; Waltz, Lucas B.; Thomson, "Industrial vehicles with overhead light based localization," oct 2015. [Online]. Available: https://www.google.com/patents/US9170581

[14] "Chalmers: Gulliver Project." [Online]. Available: http://www.chalmers.se/hosted/gulliver-en/

[15] E. Dahlgren, J. Grundén, D. Gunnarson, N. Holtryd, A. Khazal, and V. Swantesson, "Gulliver – en plattform för testning, utveckling och demonstration av transportsystem," 2012. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/161311.pdf

[16] B. Vedder, "Gulliver: Design and Implementation of a Miniature Vehicular System," 2012. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/173682/173682.pdf

[17] S. Dädeby, A. Eriksson, P. Khosravi, K. Onsjö, and K. Sandell, "Simulation av Gulliver - En virtuell robotmiljö för skalade autonoma fordon," Examensarbete för kandidatexamen, Chalmers tekniska högskola, Göteborg, 2015. [Online]. Available: http://studentarbeten.chalmers.se/publication/218962-simulation-av-gulliver-en-virtuell-robotmiljo-for-skalade-autonoma-fordon

[18] J. Hassel, R. Kemi, A. Nordin, F. Nordstedt, J. Svanberg, and C. Ågren, "Utveckling och verifiering av scenarier för autonoma miniatyrbilar," 2014. [Online]. Available: http://studentarbeten.chalmers.se/publication/203566-utveckling-och-verifiering-av-scenarier-for-autonoma-miniatyrbilar

[19] E. Eriksson, F. Hagslätt, R. Nard, and S. Wijk, "Gulliver Simulation - En systemarkitektur för nätverkssimulering av," 2015. [Online]. Available: http://studentarbeten.chalmers.se/publication/218966-gulliver-simulation-en-systemarkitektur-for-natverkssimulering-av

[20] E. Ahlberg, J. Danielsson, M. Isaksson, S. Ivarsson, and A. Johnsson, "Gulliver - Vidareutveckling av ett system för testning av autonoma bilar," Examensarbete för kandidatexamen, Chalmers tekniska högskola, Göteborg, 2014. [Online]. Available: http://studentarbeten.chalmers.se/publication/203218-gulliver-vidareutveckling-av-ett-system-for-testning-av-autonoma-bilar

[21] "Drone Waiters Will Now Take Your Order." [Online]. Available: http://www.popularmechanics.com/flight/drones/a14017/drone-waiters-infinium-robotics-singapore/

[22] "iDS uEye UI-3240LE-M-GL." [Online]. Available: https://en.ids-imaging.com/store/ui-3240le.html

[23] "Odroid UX4." [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

[24] E. Olson and M. Zucker, "GulliView," 2014. [Online]. Available: https://bitbucket.org/thpe/visionlocalization

[25] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2011, pp. 3400–3407.

[26] "Pololu - Micro Maestro 6-Channel USB Servo Controller (Assembled)." [Online]. Available: https://www.pololu.com/product/1350/resources

[27] "R.O.S." 2009. [Online]. Available: http://www.ros.org/

[28] M. Lundgren, "Path tracking for a miniature robot," *Masters, Department of Computer Science, University ...*, 2003. [Online]. Available: http://www8.cs.umu.se/kurser/TDBD17/VT06/utdelat/AssignmentPapers/PathTrackingforaMiniatureRobot.pdf

[29] B. Lennartson, *Reglerteknikens grunder*. Studentlitteratur AB, 2002.

[30] "Rviz," 2014. [Online]. Available: http://wiki.ros.org/rviz

[31] "Ids ueye readme.txt." [Online]. Available: https://en.ids-imaging.com/tl_files/downloads/uEye_SDK/readme/uEye_Linux_ReadMe_4.72.txt

[32] "OpenCV Documentation." [Online]. Available: http://docs.opencv.org/2.4/modules/core/doc/basic_structures.html?highlight=cv_8uc3#mat-depth

[33] "OpenCV projectpoints Documentation." [Online]. Available: http://docs.opencv.org/2.4.12/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=projectpoints#projectpoints

[34] K. Anjyo, H. Ochiai, M. . C. S. D. L. of Engineering, and C. S. (e-book collection), *Mathematical basics of motion and deformation in computer graphics*, 1st ed. San Rafael, California: Morgan & Claypool Publishers, 2014, vol. lecture #17;17.;.

[35] "Ackermann Prototype 1." [Online]. Available: https://bitbucket.org/thpe/ap1/

[36] "PID - ROS Wiki." [Online]. Available: http://wiki.ros.org/pid

[37] H. Tseng, J. Asgari, D. Hrovat, P. van der Jagt, A. Cherry, and S. Neads, "Evasive manoeuvres with a steering robot," *Vehicle System Dynamics*, vol. 43, no. 3, pp. 199–216, aug 2006. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/00423110420002667775

[38] C. Rohrig, D. Hes, and F. Kunemund, "Constrained Kalman filtering for indoor localization of transport vehicles using floor-installed HF RFID transponders," in *2015 IEEE International Conference on RFID (RFID)*. IEEE, apr 2015, pp. 113–120. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7113081

[39] E. DiGiampaolo and F. Martinelli, "A Passive UHF-RFID System for the Localization of an Indoor Autonomous Vehicle," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 10, pp. 3961–3970, oct 2012. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6056563

[40] "Volvo Trucks on a European Tour for Platooning | Platooning UK Haulier." [Online]. Available: http://www.ukhaulier.co.uk/news/road-transport/platooning/volvo-trucks-on-a-european-tour-for-platooning/

[41] "BBC NEWS | Technology | Robot cleaner hits the shops." [Online]. Available: http://news.bbc.co.uk/2/hi/technology/3031219.stm

[42] L. Keating, "The driverless car debate: How safe are autonomous vehicles?" July 2015. [Online]. Available: http://www.techtimes.com/articles/67253/20150728/driverless-cars-safe.htm

[43] E. Dashevsky, "Will robots make humans unnecessary?" February 2016. [Online]. Available: http://www.pcmag.com/article2/0,2817,2498688,00.asp

[44] "30 största yrkena." [Online]. Available: http://www.scb.se/sv_/Hitta-statistik/Statistik-efter-amne/Arbetsmarknad/Sysselsattning-forvarvsarbete-och-arbetstider/Yrkesregistret-med-yrkesstatistik/59064/59071/133973/

[45] "Report that shows that the human factor is the major cause of traffic accidents," 2015. [Online]. Available: http://www-nrd.nhtsa.dot.gov/pubs/812115.pdf

Bilagor

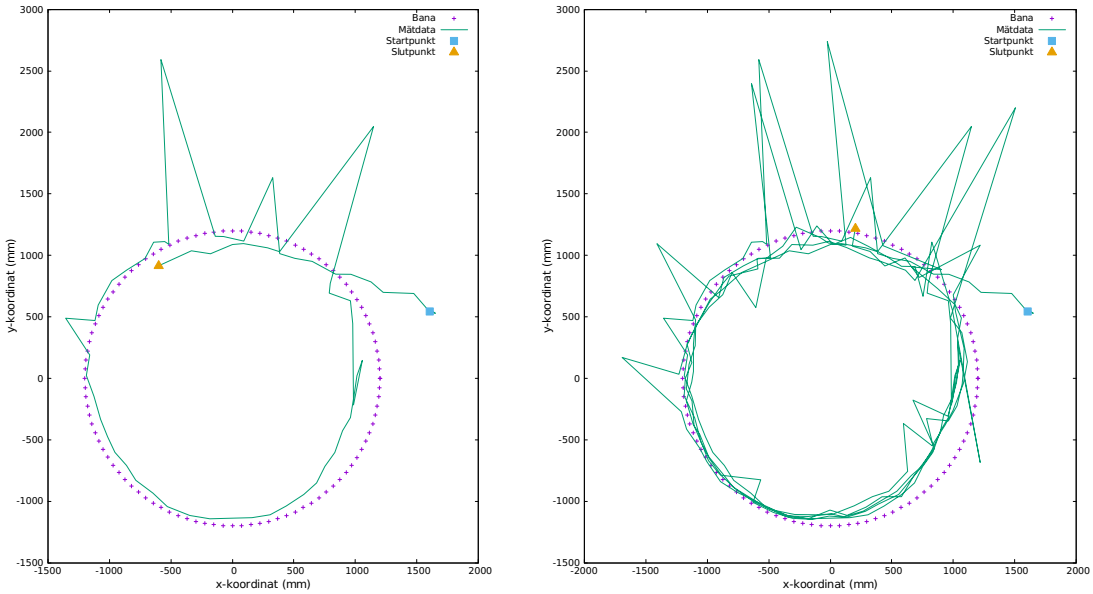
A

Grafer

Följande grafer visar några av resultaten från testningen av styrningen. Varje figur har två delfigurer; en som visar endast det första varvet, och en som visar många varv. Bilen har positionerats på olika sätt för att verifiera att systemet kan hantera olika specialfall.

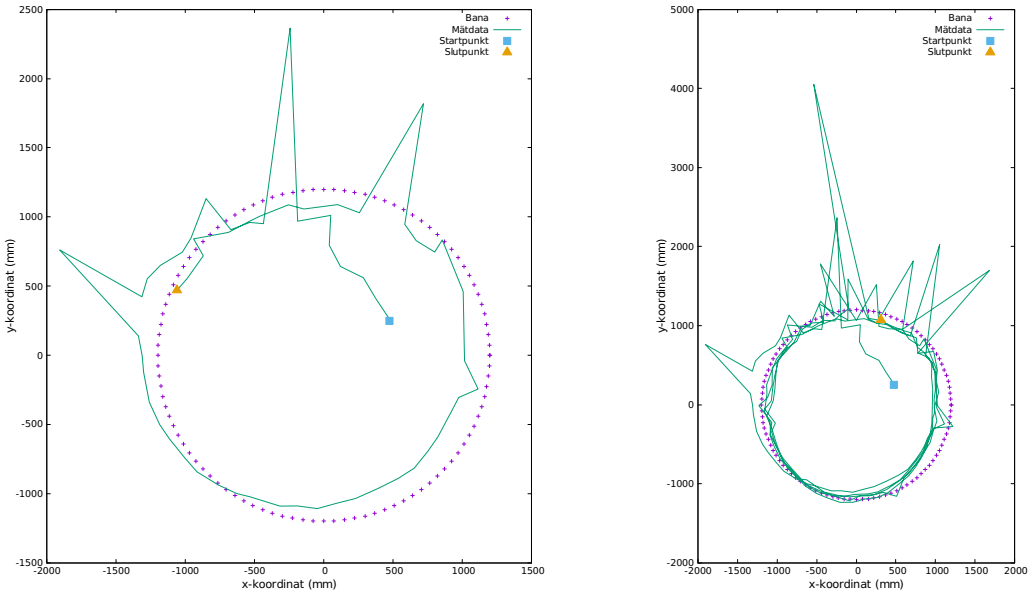
A.1 Pure Pursuit

I dessa grafer syns resultatet av Pure Pursuit-styrningen från olika utgångspunkter. I alla grafer syns ett påtagligt mätbrus, utan att resultatet påverkats märkbart.



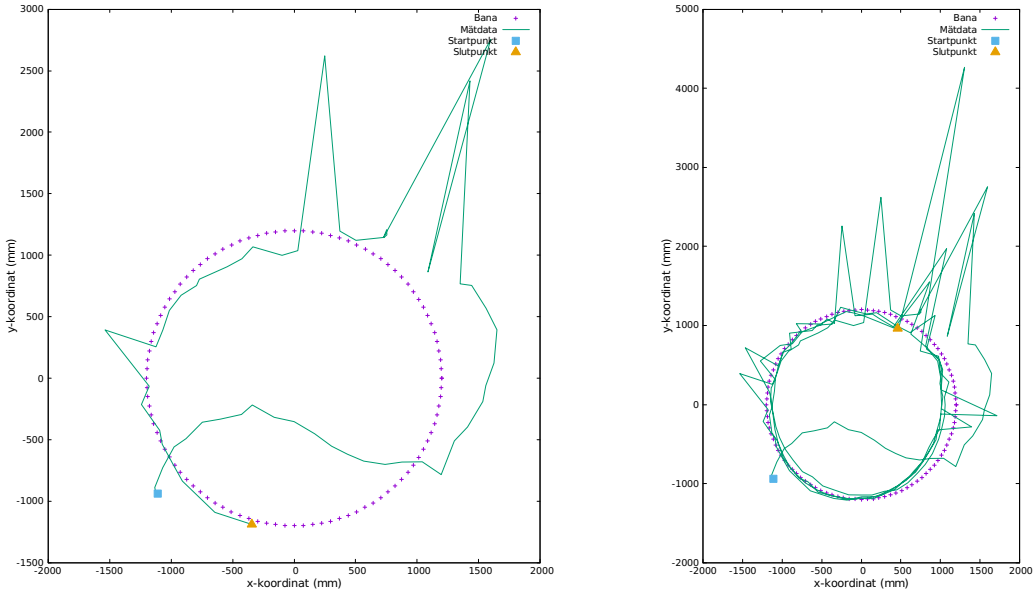
(a) Första varvet **(b)** Alla varv

Figur A.1: Bilen placeras initialt utanför banan, riktad tangentiellt mot banan.



(a) Första varvet **(b)** Alla varv

Figur A.2: Bilen placeras initialt innanför banan, riktad förhållandevis snävt mot banan. Bilen kan snabbt styra upp detta.



(a) Första varvet. **(b)** Alla varv.

Figur A.3: Bilen startar på utsidan av banan, riktad motsatt den programmerade körriktningen. Trots den osympatiska utgångspunkten har bilen redan efter lite drygt ett halvt varv korrigerat den initiala störningen och börjat följa banan. Under alla varv därefter följer bilen banan mycket väl.

B

Polulu styrenhet

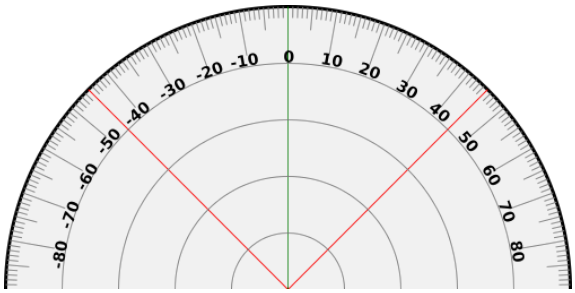
Experimentellt framtagna värden för styrenhetens meddelandeparametrar, och vilka utslag dessa ger på styr- respektive motorservo.

B.1 Styrservo

Hjulens styrvinkel beräknades genom radien på den cirkel som bilen körde i för ett visst utslag. Detta för att minska risken för felmätning.

target	Vinkel
7000	$-22°$ *Max utslag vänster*
6000	$0°$ *Rakt framåt*
5000	$22°$ *Max utslag höger*

Tabell B.1: Styrservo. Vinkeln som anges är vinkeln på framhjulen (det styrande hjulparet), och mäts som i figur B.1, det vill säga 0° är rakt framåt, vänstersväng är negativ vinkel och högersväng är positiv vinkel.



Figur B.1: Figuren visar den referenspunkt som används för angivna vinklar ovan. Omarbetat verk, från Georges Khaznadar (georgesk@ofset.org) CC BY-SA 4.0, via Wikimedia Commons
`https://commons.wikimedia.org/wiki/File%3AProtractor1.svg`

B.2 Motorservo

För att undvika ojämna testresultat ville vi hitta en motorhastighet som medgav körning med jämn hastighet. När vi funnit den lägsta hastigheten som bilen kunde

köra med jämn hastighet mättes bilens hastighet flera gånger och det genomsnittliga värdet antecknades.

target	Hastighet ($\mathrm{m\,s^{-1}}$)	
6000	0	
6075	0.425	*Långsam mjuk körning*

Tabell B.2: Motorservo. För styrsignaler mellan 6000 och 6075 fick motorn inte tillräckligt med kraft för att kunna driva bilen med jämn hastighet. 6075 var den lägsta styrsignalen där bilen kunde hålla jämn hastighet.

C

uEye konfigurationsfil

```
[Versions]
libueye_api.so=4.61.0003
ueyed-usb=4.61.0003
ueye_boot.sys=4.61.0003


[Sensor]
Sensor=UI324xLE-M
Sensor bit depth=0
Sensor source gain=0
FPN correction mode=0
Black reference mode=0
Sensor digital gain=0


[Image size]
Start X=0
Start Y=0
Start X absolute=0
Start Y absolute=0
Width=1280
Height=1024
Binning=0
Subsampling=0


[Scaler]
Mode=0
Factor=1.000000


[Multi AOI]
Enabled=0
Mode=1
x1=0
x2=0
x3=0
```

```
x4=0
y1=0
y2=0
y3=0
y4=0


[Shutter]
Mode=2
Linescan number=0


[Log Mode]
Mode=3
Manual value=0
Manual gain=0


[Timing]
Pixelclock=86
Extended pixelclock range=0
Framerate=59.822673
Exposure=16.671439
Long exposure=0
Dual exposure ratio=0


[Selected Converter]
IS_SET_CM_RGB32=1
IS_SET_CM_RGB24=1
IS_SET_CM_RGB16=1
IS_SET_CM_RGB15=1
IS_SET_CM_Y8=8
IS_SET_CM_RGB8=1
IS_SET_CM_BAYER=8
IS_SET_CM_UYVY=1
IS_SET_CM_UYVY_MONO=1
IS_SET_CM_UYVY_BAYER=1
IS_CM_CBYCRY_PACKED=0
IS_SET_CM_RGBY=8
IS_SET_CM_RGB30=1
IS_SET_CM_Y12=1
IS_SET_CM_BAYER12=8
IS_SET_CM_Y16=1
IS_SET_CM_BAYER16=8
IS_CM_BGR12_UNPACKED=1
```

```
IS_CM_BGRA12_UNPACKED=1
IS_CM_JPEG=0
IS_CM_SENSOR_RAW10=8
IS_CM_MONO10=1
IS_CM_BGR10_UNPACKED=1
IS_CM_RGBA8_PACKED=1
IS_CM_RGB8_PACKED=1
IS_CM_RGBY8_PACKED=8
IS_CM_RGB10V2_PACKED=8
IS_CM_RGB12_UNPACKED=1
IS_CM_RGBA12_UNPACKED=1
IS_CM_RGB10_UNPACKED=1
IS_CM_RGB8_PLANAR=1


[Parameters]
Colormode=6
Gamma=1.000000
Hardware Gamma=0
Blacklevel Mode=1
Blacklevel Offset=116
Hotpixel Mode=2
Hotpixel Threshold=0
Sensor Hotpixel=1
GlobalShutter=0
AllowRawWithLut=0


[Gain]
Master=0
Red=0
Green=0
Blue=0
GainBoost=0


[Processing]
EdgeEnhancementFactor=0
RopEffect=0
Whitebalance=0
Whitebalance Red=1.000000
Whitebalance Green=1.000000
Whitebalance Blue=1.000000
Color correction=0
Color_correction_factor=1.000000
Color_correction_satU=100
```

```
Color_correction_satV=100
Bayer Conversion=1
JpegCompression=0
NoiseMode=0
ImageEffect=0
LscModel=0
WideDynamicRange=0


[Auto features]
Auto Framerate control=0
Brightness exposure control=0
Brightness gain control=0
Auto Framerate Sensor control=0
Brightness exposure Sensor control=0
Brightness gain Sensor control=0
Brightness exposure Sensor control photometry=0
Brightness gain Sensor control photometry=0
Brightness control once=0
Brightness reference=128
Brightness speed=50
Brightness max gain=100
Brightness max exposure=16.700211
Brightness Aoi Left=0
Brightness Aoi Top=0
Brightness Aoi Width=1280
Brightness Aoi Height=1024
Brightness Hysteresis=2
Auto WB control=0
Auto WB type=2
Auto WB RGB color model=1
Auto WB RGB color temperature=0
Auto WB offsetR=0
Auto WB offsetB=0
Auto WB gainMin=0
Auto WB gainMax=100
Auto WB speed=50
Auto WB Aoi Left=0
Auto WB Aoi Top=0
Auto WB Aoi Width=1280
Auto WB Aoi Height=1024
Auto WB Once=0
Auto WB Hysteresis=2
Brightness Skip Frames Trigger Mode=4
Brightness Skip Frames Freerun Mode=4
Auto WB Skip Frames Trigger Mode=4
```

```
Auto WB Skip Frames Freerun Mode=4


[Trigger and Flash]
Trigger mode=0
Trigger timeout=0
Trigger delay=0
Trigger debounce mode=0
Trigger debounce delay time=1
Trigger burst size=1
Trigger prescaler frame=1
Trigger prescaler line=1
Trigger input=1
Flash strobe=0
Flash delay=0
Flash duration=0
Flash auto freerun=0
PWM mode=0
PWM frequency=62500000
PWM dutycycle=62500000
GPIO state=3
GPIO direction=0


[Sequence AOI]
NumberUsedAOI=0
StartX1=0
StartY1=0
Gain1=0
Exposure1=16.700211
ReadoutCycle1=1
BinningMode1=0
SubsamplingMode1=0
ScalerFactor1=0.000000
DetachImageParameter1=0
StartX2=0
StartY2=0
Gain2=0
Exposure2=16.700211
ReadoutCycle2=1
BinningMode2=0
SubsamplingMode2=0
ScalerFactor2=0.000000
DetachImageParameter2=0
StartX3=0
StartY3=0
```

```
Gain3=0
Exposure3=16.700211
ReadoutCycle3=1
BinningMode3=0
SubsamplingMode3=0
ScalerFactor3=0.000000
DetachImageParameter3=0


[Vertical AOI Merge Mode]
Mode=0
Position=0
Additional Position=0
Height=1
```

D

cameraAngle.m

cameraAngle.m

A script for positioning ceiling mounted cameras

Andreas Lindhé

lindhea@student.chalmers.se

https://bitbucket.org/lindhea/cameraangle.git

2016-05-14

Calculating what field of view a camera has can be tedious. Especially if the camera is tilted. This is the documentation for a script that makes this task much easier. This documentation has a focus on the mathematics behind it, and not very much on how to use the script. The script is well commented. Combined with the README in this repo, it should be quite clear how to use it.

`f`	Focal length.
`h`	Vertical distance from horizontal viewing plane to camera lens.
`i`	Inclination.
`sensorWidth`, `sensorHeight`	The physical sensor size (in mm).
A_h, A_v	The angle of view in horizontal and vertical direction respectively, as seen from the camera view finder.
AOV	Angle of view.
FOV	Field of view.
Inclination/camera angle	The smallest angle between the vertical and the center line of the camera view.

Table 1: Variables, abbreviations and terms

1 Camera parameters and angle

For this script to work properly, a few camera parameters needs to be set before the FOV can be calculated. The variable `f` is the focal length, which

simply is the focal length of the camera lens (in mm). The physical sensor size (also in mm) is defined as `sensorWidth` and `sensorHeight`. These numbers should be found in the camera documentation.

With these camera parameters, the camera view angle for each direction can be calculated by equation 1. The angle of view, A, in the direction measured depends on the sensor size in that direction, d.

$$\alpha = A = 2 \cdot \arctan\left(\frac{d}{2 \cdot f}\right) \tag{1}$$

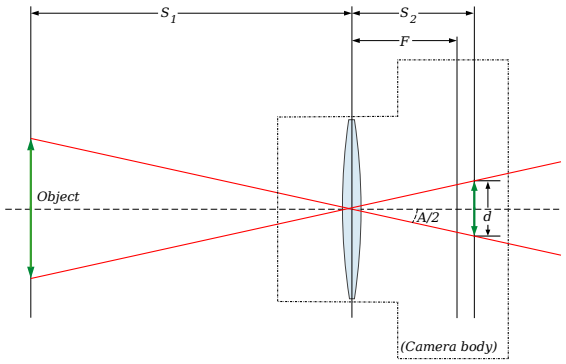


Figure 1: FOV of the camera when tilted in one direction.
By Moxfyre at English Wikipedia, CC BY-SA 3.0,
`https://commons.wikimedia.org/w/index.php?curid=6545057`

The height, `h`, is the vertical distance from the horizontal plane of the subject (often the ground plane) to the outwards facing center point of the lens. With no inclination, this will project a rectangular shape (with similar proportions to the sensor) onto the horizontal plane which spans $2 \cdot h \cdot tan(\frac{A_h}{2})$ long and $2 \cdot h \cdot tan(\frac{A_v}{2})$ wide.

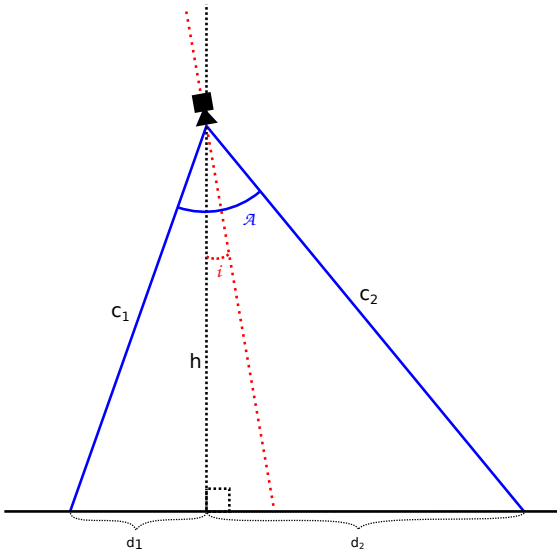


Figure 2: FOV of the camera when tilted in one direction.

2 Tilting

To calculate the FOV when tilting the camera, things are not quite as straight forward. The viewing distance, however, is quite easy.

2.1 View distance

If we first consider the view distance on the horizontal plane, d_2 in figure 2, we can tell the top angle is $\frac{A}{2}+i$, where i is the inclination. Thus the distance

$$d_2 = h \cdot \arctan(\frac{A}{2} + i)$$

Similar for the opposite end of the field of view, the distance

$$d_1 = h \cdot \arctan(\frac{A}{2} - i)$$

The total view distance is $d_1 + d_2$. To make scripting easier this calculation is made by the function `fov` in `fov.m`.

2.2 View width

The width of the area on the horizontal viewing plane, covered by the field of view, is a linear variation over distance (as is often the case with trapezia). The shortest edge, let's call it w_1, is at the furthest away tip of d_1. The width of it could be seen as the base of a triangle like figure 2.2.

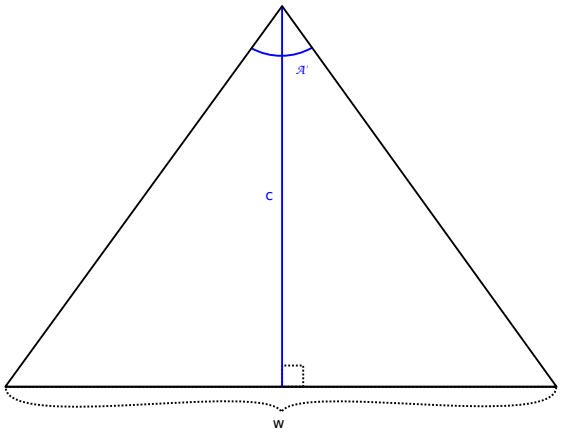


Figure 3: The height c (where c is either c_1 or c_2; the side of the triangle that goes from the furthest most tip of d_1 (or d_2) to the top of the camera lens) with the top corner angle equal to the AOV in the opposite direction, A'.

Trivially (but not obviously),

$$c = \frac{h}{\sin\left(\frac{\pi}{2} - \left(\frac{A}{2} - i\right)\right)}$$

, and thus $w = 2 \cdot c \cdot \tan\left(\frac{A'}{2}\right)$.

These steps are done in `perspectiveWithIncline.m`, leaving the script to calculate w_1 and w_2 and printing the dimensions of the trapezium to the user.

3 Camera positioning

One of the parts that is really hard to solve by hand is to calculate the angle, given a certain desired view distance. The equation for the view distance d given the inclination i (assuming reasonable AOV and an inclination less than or equal to that) looks like

$$d(i) = h \cdot \left(\tan\left(\frac{A}{2} + i\right) + \tan\left(\frac{A}{2} - i\right)\right)$$

, and we want to solve for i, given d.

Luckily, MATLAB solves this numerically for us. In the code, this is implemented with `fzero`:

```
guess_h = 0.2;
horizInclination = @(i_h) h*(tan(A_h/2 + i_h)+tan(A_h/2 - i_h)) ...
-horizontalDistance;
i_h = fzero(horizInclination, guess_h);
```

Of course, MATLAB's algorithm needs a bit of heuristics to give a reasonable result, so we need to give a guess. In most cases, 0.2 rad is a good enough guess.

This last part is really easy, but for convenience the script also prints what position the camera should have, relative to the closest wall (trivially equal to d_1).

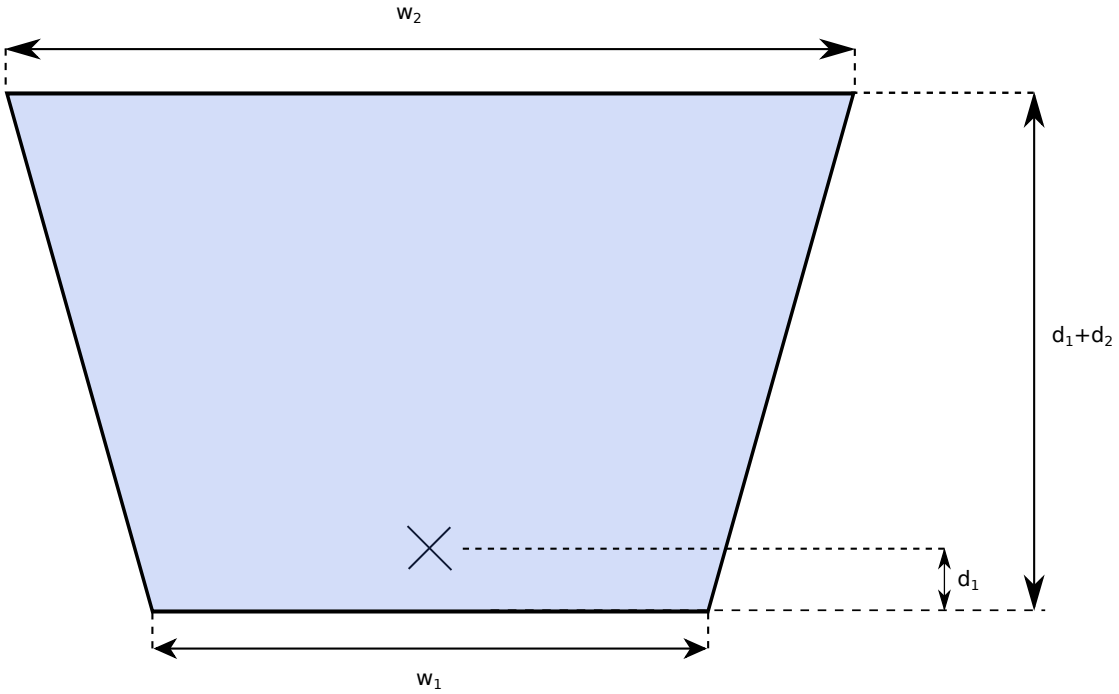


Figure 4: The FOV and camera placement, as calculated by the script.

The script outputs these measurements, and it's up to the user to pick and choose the relevant numbers, and to piece these together for the real world camera positioning. Autonomous drones mounting the cameras for you comes in the next release.

4 Example output

```
--------------------------------
###  Camera position script  ###
--------------------------------

View area without inclination is: 3699*2959 mm (horizontal, vertical)

------------------------
#  View area inclined  #
------------------------

----------------
~~ Horizontal ~~
----------------

View area with 0.349 rad horizontal inclination is a isosceles trapezium with
height 4485 mm and the top and bottom edge 3067 and 5191 mm respectively.

--------------
~~ Vertical ~~
--------------

View area with 0.349 rad vertical inclination is a isosceles trapezium with
height 3499 mm and the top and bottom edge 3750 and 5693 mm respectively.

----------------------
#  Camera position   #
----------------------

----------------
~~ Horizontal ~~
----------------

To have 4100 mm horizontal FOV the camera needs a horizontal incline of 14.8 deg

This renders a trapezium with edges 3158 and 4608 mm.

The camera should be placed in the horizontal center, 976 mm from the wall.

The longest rectangle you can have is 1226 mm long

--------------
~~ Vertical ~~
```

To have 3500 mm vertical FOV the camera needs a vertical incline of 20.0 deg

This renders a trapezium with edges 3750 and 5695 mm.

The camera should be placed in the horizontal center, 436 mm from the wall.

The widest rectangle you can have is 2870 mm wide

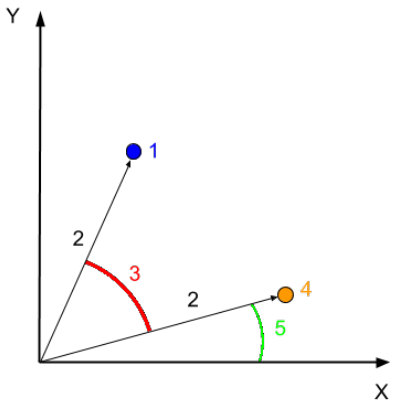
E

Byte av koordinatsystem

I detta bilaga presenteras härledningen formeln för att konvertera en punkt i det globala koordinatsystemet till en punkt i bilens lokala koordinatsystem.

$$x_{local} = (x_{goal} - x_{car})\cos(\theta) + (y_{goal} - y_{car})\sin(\theta) \tag{E.1}$$
$$y_{local} = -(x_{goal} - x_{car})\sin(\theta) + (y_{goal} - y_{car})\cos(\theta) \tag{E.2}$$

Härledning och förklarande figur:



Figur E.1: Figuren visar rotation av en punkt i koordinatsystemet. (1) Punkten efter rotation, ($x_{rotated}$,$y_{rotated}$). (2) Avståndet till origo, r. (3) Rotationsvinkeln, θ. (4) Punkten innan rotation, (x,y). (5) Punktens vinkel innan rotation, ϕ

$$x_{offset} = x_{goal} - x_{car} \tag{E.3}$$
$$y_{offset} = y_{goal} - y_{car} \tag{E.4}$$

$$x = r\cos(\phi) \tag{E.5}$$
$$y = r\sin(\phi) \tag{E.6}$$

$$x_{rotated} = r\cos(\theta + \phi) \tag{E.7}$$
$$x_{rotated} = r\cos(\theta)\cos(\phi) - r\sin(\theta)\sin(\phi) \tag{E.8}$$
$$x_{rotated} = x\cos(\theta) - y\sin(\theta) \tag{E.9}$$

$$y_{rotated} = r\sin(\theta + \phi) \tag{E.10}$$
$$y_{rotated} = r\sin(\theta)\cos(\phi) + r\cos(\theta)\sin(\phi) \tag{E.11}$$
$$y_{rotated} = x\sin(\theta) + y\cos(\theta) \tag{E.12}$$

$$x_{rotated\ and\ moved} = (x_{goal} - x_{car})\cos(\theta) - (y_{goal} - y_{car})\sin(\theta) \tag{E.13}$$
$$y_{rotated\ and\ moved} = (x_{goal} - x_{car})\sin(\theta) + (y_{goal} - y_{car})\cos(\theta) \tag{E.14}$$

$$x_{local} = (x_{goal} - x_{car})\cos(\theta - 90^\circ) - (y_{goal} - y_{car})\sin(\theta - 90^\circ) \tag{E.15}$$
$$y_{local} = (x_{goal} - x_{car})\sin(\theta - 90^\circ) + (y_{goal} - y_{car})\cos(\theta - 90^\circ) \tag{E.16}$$

$$x_{local} = (x_{goal} - x_{car})\cos(\theta) + (y_{goal} - y_{car})\sin(\theta) \tag{E.17}$$
$$y_{local} = -(x_{goal} - x_{car})\sin(\theta) + (y_{goal} - y_{car})\cos(\theta) \tag{E.18}$$