CHALMERS
UNIVERSITY OF TECHNOLOGY

Autonom styrning av två sammankopplade dollys

Utveckling, simulering och implementering av ett autonomt system på två separata dollys

Kandidatarbete inom system- och reglerteknik

Samuel Armgren Filip Elster Carl Henrysson Pontus Strandvik Hannes Lundberg Rasmus Bäckström

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2024
www.chalmers.se

Kandidatarbete 2024

Autonom styrning av två dollys

Utveckling, simulering och implementering av ett autonomt system på två separata dollys

Samuel Armgren
Filip Elster
Carl Henrysson
Pontus Strandvik
Hannes Lundberg
Rasmus Bäckström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2024

Autonom styrning av två dollys
Utveckling, simulering och implementering av ett autonomt system på två separata dollys

SAMUEL ARMGREN
FILIP ELSTER
CARL HENRYSSON
PONTUS STRANDVIK
HANNES LUNDBERG
RASMUS BÄCKSTRÖM

Handledare: Pavel Anistratov, Institutionen för Elektroteknik
Examinator: Jonas Fredriksson, Institutionen för Elektroteknik

Kandidatuppsats 2024 EENX16-24-81
Institutionen för Elektroteknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Sverige
Telefon +46 31 772 1000

Omslag: Skalmodell använd för testning, implememententering och validering av styralgoritmer.

Göteborg, Sverige 2024

Abstract

This report presents a comprehensive investigation into the integration and enhancement of lane detection and adaptive cruise control within autonomous vehicles, leveraging sensor technologies and image processing algorithms. By integrating these technologies, the aim is to improve the autonomy of vehicles, laying the groundwork for future advancements in autonomous transportation and for future bachelor theses.

The lane detection algorithm include Canny Edge Detection for edge detection and the Hough Transform for line detection, coupled with a Proportional-Integral-Derivative (PID) regulator to maintain the vehicle within its lane accurately. This setup is further refined using a Kalman Filter to enhance the precision and robustness of lane detection.

Concurrently, the adaptive cruise control is developed around a Proportional (P) regulator, which utilizes the input from the ultrasonic sensor mounted on the vehicle to adjust speed and maintain a safe following distance from the vehicle ahead. Additionally, ArUco marker detection is used to aid in a docking sequence. This integrated setup employs a Raspberry Pi mounted on the vehicle, facilitating real-time image processing for lane detection and communication with vehicle control systems to enable autonomous driving.

The effectiveness of the lane detection algorithm is simulated within the CARLA simulation environment. The findings underscore a well-functioning lane detection algorithm, paving the way for a safer, more efficient autonomous driving solution. The lane detection algorithm, combined with a ultra sonic sensor, is then tested in reality on a prototype vehicle. The tests, after the physical implementation of the algorithm, confirmed that the lane detection algorithm was capable of accurately identifying lanes. Furthermore, the integration of the ultrasonic sensor demonstrated effective speed adjustment and distance maintenance from the vehicle ahead.

Nyckelord: Automation, dolly, vägföljning, bildanalys, lastbil, transport, ruttplanering, dockning, avståndshållning.

Förord

Denna rapport är resultatet av kandidatarbetet EENX16-24-81 vid institutionen för elektroteknik vid Chalmers Tekniska Högskola. Arbetet gjordes tillsammans med kandidatarbetet EENX16-24-05. Examinatorn för arbetet är Jonas Fredriksson, programansvarig för mastersprogrammet Systems, control and mechatronics. Handledare för arbetet är Pavel Anistrov som är postdok på instutitionen för Mekatronik. Vi vill tacka vår handledare Pavel för givande information och rutin inom ämnet. Vi vill även passa på att tacka gruppen bakom arbetet EENX16-24-05 som genom bra samarbete gjorde projektet möjligt.

SAMUEL ARMGREN, FILIP ELSTER, CARL HENRYSSON, PONTUS STRANDVIK, HANNES LUNDBERG, RASMUS BÄCKSTRÖM

Göteborg, Maj 2024

Lista över förkortningar och benämningar

Nedan finns en alfabetisk lista på förkortningar och benämningar som använts i arbetet:

Förkortningar

API	Application programming interface
ArUco	Augmented Reality University of Cordoba
CASE	Chalmers Autonomous Solutions and Electronics
GPIO	General-purpose Input/Output
I/O	Intput/Output
P	Proportional
PI	Proportional Integral
PID	Proportional Integral Derivate
PWM	Pulse Width Modulation
RGB	Red, Green, Blue
ROI	Region Of Interest
RPi	Raspberry Pi
USB	Universal Serial Bus

Benämningar

Arduino Mega	Mikrokontroller från Arduino.
ArUco-markör	En markör likt QR-kod.
Carla	Öppen simuleringsplattform för autonoma körningssystem.
ChArUco-bräde	Hybrid markör som kombinerar ArUco-markörer med ett schackbrädesmönster.
Dolly	Den körande delen av ekipaget.
Dupont kabel	Flexibla, löstagbara hopkopplingskablar som används för att enkelt ansluta komponenter på elektroniska kretskort och prototypsystem.
Ekipage	Dolly med kopplat släp.
Firmware	Inbyggd programvara som styr hårdvarans grundläggande funktioner.
Jetson Nano	Mikrokontroller från NVIDIA
Kalmanfilter	Algoritm som använder en serie mätningar observerade över tid för att uppskatta en okänd variabels tillstånd och minska osäkerheten från brusiga data.
Kinematik	Matematisk modell som beskriver rörelser utan hänsyn till krafter.
MATLAB	Ett datorprogram och programspråk skapad av MathWorks
OpenCV	Programbibliotek för maskininlärning och bildanalys.
Raspberry Pi	Mikrokontroller från Raspberry Pi Inc.
Simulink	En utökning av MATLAB som används för simuleringar
Ziegler–Nichols	Metod för att dimensionera PID-regulator.

Variabeldeklaration

Nedan följer nomenklaturen för index, vektorer och matriser som har använts i denna rapport.

Index

m	Radindex
n	Kolumnindex

Vektorer

$\boldsymbol{x}$	Tillståndsvektor
$\boldsymbol{u}$	Styrsignalvektor
$\boldsymbol{y}$	Utsignalvektor
$\boldsymbol{T}$	Translationvektor

Matriser

$\boldsymbol{A}$	Systemmatris
$\boldsymbol{B}$	Insignalsmatris
$\boldsymbol{C}$	Utsignalsmatris
$\boldsymbol{D}$	Direktmatris
$\boldsymbol{A}_i$	Yttre kameramatris
$\boldsymbol{R}$	Rotationsmatris
$\boldsymbol{R}\|\boldsymbol{T}$	Inre kameramatris
$\boldsymbol{A_k}$	Tillståndsövergångsmatris
$\boldsymbol{B_k}$	Styrsignalsmatris

$\boldsymbol{Q}$	Kovariansmatris med processbrus
$\boldsymbol{H}$	Mätmatris
$\boldsymbol{R_k}$	Kovariansmatris med mätbrus
$\boldsymbol{I}$	Identitetsmatrisen
$\boldsymbol{P}$	Kovariansmatris

Innehåll

Figurer

Tabeller

1

Inledning

Godstransporter spelar en avgörande roll inom transportsektorn, där över 106 373 miljoner tonkilometer gods transporterades i Sverige år 2022, med lastbilstrafiken som den dominerande aktören, stående för mer än 50% av transportarbetet [1]. Denna omfattande verksamhet medför inte bara leverans av varor utan även koldioxidutsläpp och risk för trafikolyckor.

Ett av de främsta motiven för införandet av autonoma fordon är att minska bränsleförbrukningen, förkorta transporttiden och öka trafiksäkerheten [2]. Genom att automatisera godstransporter, särskilt med lastbilar, förväntas antalet olyckor minska, bränsleeffektiviteten förbättras och leveranstiderna kortas. Enligt forskning kan tekniker som platooning, där lastbilar färdas i tåg efter varandra för att bland annat minska luftmotståndet, bidra till betydande bränslebesparingar och förbättrad trafiksäkerhet [3].

Trots fördelarna med autonoma lastbilar kan längre fordonståg möta utmaningar vid navigering i vissa miljöer, såsom rondeller, vilket kan leda till ökad transporttid och risk för andra trafikanter. Införandet av autonoma lastbilar ställer därför höga krav på navigeringssystemen, inklusive snabb datahantering från sensorer och robusta algoritmer som kan fatta korrekta beslut.

Autonomi hos fordon graderas vanligtvis på en skala från 0 till 5, där nivå 0 innebär helt manuell styrning och nivå 5 innebär full autonomi utan mänsklig interaktion [4]. Genom att förstå dessa nivåer kan utvecklingsstadiet för autonoma fordon bedömas och vilka utmaningar som måste övervinnas för att uppnå nästa steg i autonomin av ett fordon.

Autonoma fordon är ett område med kontinuerlig forskning och utveckling, där ledande företag och organisationer som Google, universitet och fordonsföretag som Toyota och Skoda är aktiva [5]. Tekniker som filassistenter, som används för att hålla fordonet inom körfältet, representerar ett viktigt steg mot automatisering av godstransporter med lastbil [6].

1.1 Bakgrund

Arbetet bygger på tidigare kandidatarbeten och information och inspiration har tagits från dessa källor. Framförallt har inspiration tagits från kandidatarbete EENX16-23-15, Autonom styrning av lastbil.

1.1.1 Tidigare arbeten

Detta kandidatarbete bygger vidare på tidigare versioner, som först påbörjades 2021. De tidigare arbetena har haft olika fokus, men alla har haft som mål att få en lastbil med släp att utföra vissa manövrar autonomt. I detta arbete konstrueras en ny dolly eftersom den tidigare modellen skrotades på grund av designproblem i styrningen och andra komponenter. Kandidatarbetet från 2023 använde kameror för att identifiera ArUco-koder för att positionera dollyn i rummet [7]. Dessa kameror återanvänds nu för att identifiera vägbanan framför dollyn samt ArUco-koder på dollyns baksida. Ultraljudssensorer, som tidigare har använts för att mäta avstånd, återanvänds också i detta projekt. Kamerorna kalibreras med hjälp av ett chArUco-bräde, som också används för kalibrering i denna upplaga.

1.2 Syfte

Syftet med arbetet är att utveckla mjukvara för två miniatyrdollyer så att de kan köra autonomt. Denna målsättning kan delas upp i tre huvudetapper:

- Utveckling och implementering av en vägföljningsalgoritm.
- Utveckling och implementering av en avståndshållaralgoritm.
- Utveckling och implementering av en autonom dockningssekvens.

1.3 Avgränsningar

- Ekipagen kommer endast att köra på en plan yta. Eventuella ojämnheter och defekter som finns i en verklig vägbana försummas.
- Den operativa miljön kommer att vara begränsad till inomhusmiljöer med optimal belysning för algoritmen.
- För att förenkla den dynamiska modellen och den mekaniska konstruktionen kommer ekipagen endast att framföras i låga hastigheter.
- Dockning av de två ekipagen genomförs endast när det ledande ekipaget är stillastående.

- Styralgoritmen kommer vara anpassad specifikt för användning inom en definierad skalmodell och dess tillhörande sensorer. Denna specialisering indikerar att algoritmens tillämpbarhet inte nödvändigtvis kan utvidgas till andra fordonstyper, inklusive andra skalmodeller eller fullskaliga autonoma lastbilar.

- Färdbanan kommer bestå av en körbana med två avgränsande kantlinjer. Dessa linjer ska vara vita och heldragna och utgör den primära guidningen för dess navigering. Färdbanan ska inte heller innehålla några hinder.

- Inget globalt positioneringssystem kommer undersökas.

1.4 Uppdelning av kandidatarbete

Kandidatarbetet ”Autonom Lastbil” är resultatet av ett samarbete mellan två kandidatgrupper. I denna rapport, författad av mjukvarugruppen, har fokus legat på utvecklingen av de autonoma funktionerna för ekipagen. Arbetet har inkluderat implementeringen av autonom styrning och linjeföljning, avståndsmätning samt autonom dockning.

Den andra gruppen, känd som hårdvarugruppen, har inriktat sig på konstruktionen av ekipagen samt elektroniken. Deras arbete har främst omfattat utvecklingen av den fysiska delarna samt dimensioneringen av motorer och styrutslag.
Rapporten kommer enbart presentera mjukvarugruppens resultat.

2
Teori

I detta kapitlet förklaras den teorin som använts i projektet, med hjälp av detta har relevanta simuleringar och metoder skapats.

2.1 Vägföljningsalgoritm

Vägföljning är en teknik som används över hela världen och är en integrerad del av fordonsindustrin. Den används för att säkerställa säker och stabil autonom körning av fordon och kan även fungera som ett stöd- och hjälpmedel för förare. Tekniken bygger på en eller flera kameror som, via bildhantering och algoritmer, håller fordonet i mitten av körfältet under körning. Bildhanteringen innefattar vanligtvis flera steg av algoritmer som bearbetar kamerans bild så att fordonet kan använda den effektivt.

För att lättare kunna behandla bilder och identifiera kanter och väglinjer omvandlas bilden först till gråskala. Detta är eftersom att för att kunna identifiera kanter behövs det en jämförelse mellan ljusintensiteten i närliggande pixlar. Att filtrera bort färg och bara visa bilden i gråskala filtrerar bort svårigheter i denna identifikation och är standard inom bildanalys. Eftersom gråskala bara är en representation av RGB i samtliga tre dimensioner, kan gråskalan ses som ett medelvärde av den röda, blåa och gröna färg i respektive pixel.

För att reducera mängden brus i en bild görs bilden mer suddig. Detta tar även bort en viss mängd detaljer som kan anses vara oväsentliga vid detektion av kanter och linjer senare i algoritmen. För att möjliggöra detta används ett gaussianskt filter [8]. Ekvationen för filtret beskrivs enligt

$$G_{\text{filt}}(i,j) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right); 1 \leq i,j \leq (2k+1), \quad (2.1)$$

$$G_{\text{filt}}(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \quad (2.2)$$

Vid applicering av det gaussianska filtret från Ekvation 2.2 bestäms först en mittpixel i bilden. Koordinaterna (x,y) i Ekvation 2.2 representerar i sin tur avstånd från mittpunkten där varje pixel har storleken 1. Detta resulterar då i en normalfördelning som centreras kring ett maximalt värde i mittpunkten.

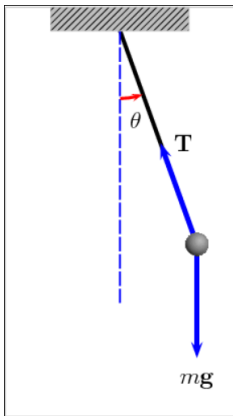
Kantdetekteringsalgoritmen som används kallas för Canny-Egde Detection och tar fram eventuella kanter i bilden som senare kan användas av Houghs Transform-algoritmen. För att att kunna urskilja vad som är kanter och inte i en bild beräknas gradienten i varje pixel, både veriktalt och horisontellt [9]. Vidare skriver Mathworks att därefter filtreras pixlar bort utifrån de två trösklarna som är satta i funktionen, samt delar upp pixlar i starka och svaga, vilket gör bilden mindre känslig för brus. Det sista steget i kantdetekteringsalogritmen är att kolla om varje stark pixel har en närliggande svagare pixel som möjligtvis också kan vara en del av kanten skriver Mathworks. Denna algoritm lägger en bra grund för Houghs Transform-algoritmen som används för att detektera linjer från den kantdetekterade bilden.

Efter att ha använt Canny Edge Detection för att hitta kanter i en bild, appliceras Hough Transform algoritmen för att identifiera och extrahera linjer från den kantbehandlade bilden. En rät linje kan beskrivas med $y = kx + m$ där där k är lutningen och m är y-skärningen [10]. I Hough transformen omvandlas varje punkt (x, y), alltså kanterna från Canny-algoritmen, i bildrummet till kurvor av möjliga (k, m) värden i parameterutrymmet, skriver Aarthy R. När vi har en samling av punkter som antas ligga på samma linje, kommer deras motsvarande kurvor i parameterutrymmet att korsa vid en punkt som representerar den gemensamma linjens lutning (k) och y-skärning (m). Vidare skriver Aarthy R att för att effektivt hantera detta parameterutrymme används ofta ett form av röstningssystem genom att skapa en ackumulator. I praktiken initieras då en tvådimensionell matris där varje cell representerar en ackumulerad summa av korsningar för ett specifikt (k, m)-par, hävdar Aarthy R. Genom att räkna antalet gånger varje möjligt linjepar korsar i ackumulatorn, kan vi identifiera de mest framträdande linjerna i bilden, skriver Aarthy R.

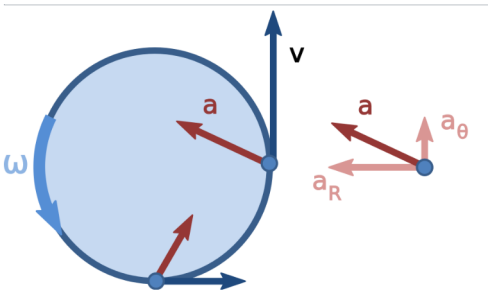
2.2 Simulering av avståndshållning

För att kunna simulera ett system behövs en dynamisk modell av det som ska simuleras. En dynamisk modell beskriver ett system och hur det beter sig över tid. Modellen består av en eller flera matematiska uttryck som kan härledas genom ett antal metoder, bland annat genom Newtons andra lag, $\sum F = m\ddot{x}$.

Här är $\sum F$ summan av alla krafter som verkar på en kropp, m är kroppens massa och $\ddot{x}$ är kroppens acceleration. Vid användadet av denna metod skapas en modell som tar hänsyn till alla krafter i systemet. Detta hänvisas till som en kinetisk modell och ett exempel på detta visas i Figur 2.1. Beskrivs systemet endast med hänsyn till geometriska förhållanden kallas det en kinematisk modell vilket visas i Figur 2.2. Kinetiska eller kinematiska modeller kan användas i exempelvis simuleringar för att beskriva ett system samt iaktta hur systemet beter sig beroende på olika exciteringar.



Figur 2.1: Kinetisk modell av en pendel med kraftpilar markerade [11].



Figur 2.2: Kinematisk modell av partiklar som rör sig i en cirkelbana. Hastighet, vinkelhastighet och acceleration är utmarkerade [12].

Efter att en dynamisk modell har tagits fram behövs en tillståndsmodell som relaterar insignalen till systemet till utsignalen. En tillståndsmodell är ett matematisk verktyg som vanligtvis består av två ekvationer som visas i Ekvation 2.3. A, B, C och D är matriser och $\mathbf{x}$, $\mathbf{y}$, $\dot{\mathbf{x}}$ och $\mathbf{u}$ är vektorer. Ett systemet som beskrivs i kontinuerlig tid kan ritas upp enligt

$$\begin{aligned} \dot{\mathbf{x}}(t) &= A(t)\mathbf{x}(t) + B(t)\mathbf{u}(t), \\ \mathbf{y}(t) &= C(t)\mathbf{x}(t) + D(t)\mathbf{u}(t). \end{aligned} \tag{2.3}$$

I en tillståndsmodell definieras systemets tillstånd som ett antal tillståndsvariabler som definieras i $\mathbf{x}(t)$ vektorn. Tillståndsvariablerna i sin tur innehåller informationen som krävs för att beräkna hur systemet kommer att reagera på en insignal. Ekvationerna som används i en tillståndsmodell består av första och/eller andra ordningens differentialekvationer [13].

För att styra systemet som är beskrivet av den dynamiska modellen och tillståndsekvationen kan det behövas en regulator. Regulatorer används för att styra ett system till en given punkt, vanligtvis med hjälp av ett fel som beräknas som skillnaden mellan på vilken punkt systemet vill regleras till och vart systemet befinner sig vid det specifika tillfället. En vanlig klass av regulatorer är PID regulatorerna som kan delas upp i olika varianter beroende på systemet som man önskar reglera [14]. Den mest grundläggande är en P-regulator och reglerar systemet genom att justera styrsignalen proportionellt mot felet. Kallas regulatorn F blir ekvationen för en P-regulator

$$F_P = K_p, \tag{2.4}$$

där K_p är den proportionella förstärkningen. Dessa regleringar kan ibland generera ett statiskt reglerfel och därför kan det läggas till en integerande faktor i regulatorn. Regulatorn blir då en PI-regulator och beskrivs på följande sätt

$$F_{PI} = K_p + \frac{1}{T_i}\int_0^t dt, \tag{2.5}$$

där T_i är integraltidskonstanten. Ett system som har krav på att snabbt regleras till sin referens utan att översvängningar kan även behöva en deriverade faktor i regulatorn. Detta kallas för PID-regulator och beskrivs då på detta sättet

$$F_{PID} = K_p + \frac{1}{T_i}\int_0^t dt + T_d\frac{d}{dt}, \tag{2.6}$$

där T_d är derivatatidskonstanten. Dessa regulatorer har olika fördelar och nackdelar vilket visas i Tabell 2.1.

Tabell 2.1: Fördelar, nackdelar och användningsområde för P, PI och PID regulatorer [14].

	P	PI	PID
Fördelar	Snabb responstid	Minimerar det kvarstående felet	Minskar översvängningar av systemet
Nackdelar	För ej systemet till önskad referenspunkt. Kan ge översvängningar vid hög förstärkning	Gör inget åt översvängningar	Kan var känslig för mätbrus
Användningsområde	Kan användas för system där en offset är tolerbar	Kan ännvändas för system där en viss översvängning är tolererbar	Används med fördel för system med begränsat mätbrus

2.3 Kalmanfilter

Ett Kalmanfilter är en algoritm som kort sagt uppskattar kända och okända parametrar från mätvärden tagna under ett visst tidsförlopp [15]. Filtret eller algoritmen kan då trots dåliga och osäkra mätvärden uppskatta kommande värden med stor säkerhet. Algoritmen är i grunden väldigt generisk och kan appliceras på många områden och på olika sätt. Processen fungerar i princip på samma sätt, det börjar med en initiering av algoritmen med de initiala tillståndet $\hat{x}_0$ och den initiala kovariansmatrisen P_0, se Tabell 2.2 för variabelförklaring. [16]. Andra steget i processen är att uppskatta ett nytt tillstånd

$$\hat{x}_{k|k-1} = A_k\hat{x}_{k-1} + B_k u_k, \tag{2.7}$$

och en kovariansmatris

$$P_{k|k-1} = A_k P_{k-1} A^T + Q. \tag{2.8}$$

Här används systemets insignal u, tidigare tillstånd $\hat{x}_{k-1}$, P_{k-1} och kovariansmatrisen med processbrus (Q) för att skapa en prediktion av nästa tillstånd. Detta kallas vanligtvis prediktionssteget.

När ett uppskattat tillstånd har tagits fram är nästa steg att en mätning z_k inhämtas för att jämföra med den aktuella skattningen $\hat{x}_k$. Genom

$$y_k = z_k - H\hat{x}_{k|k-1}, \tag{2.9}$$

som används för att förutspå nästa värde. Nästa steg är beräkna Kalman-förstärkningen K_k. Detta görs för att kunna beräkna vilken vikt man vill ge den nya mätningen jämfört med både tidigare mätningar och skattningar. Med ekvationen

$$K_k = \frac{P_{k|k-1}H^T}{HP_{k|k-1}H^T + R_k}, \tag{2.10}$$

kan sedan processen fortsätta för att ta fram nästa tillstånd

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k y_k, \tag{2.11}$$

och kovariansmatrisen

$$P_k = (I - K_k H)P_{k|k-1}. \tag{2.12}$$

Med dessa värden kan man sedan gå tillbaka till Ekvation 2.7-2.8 för att rekursivt fortsätta mätningar och skattningar i real-tid. Då algoritmen använder sig av dessa ekvationer kan en säker uppskattning av nästa tillstånd x_k göras trots både mycket mätbrus och processbrus [16].

Tabell 2.2: Variabler för Kalmanfiltrets ekvationer.

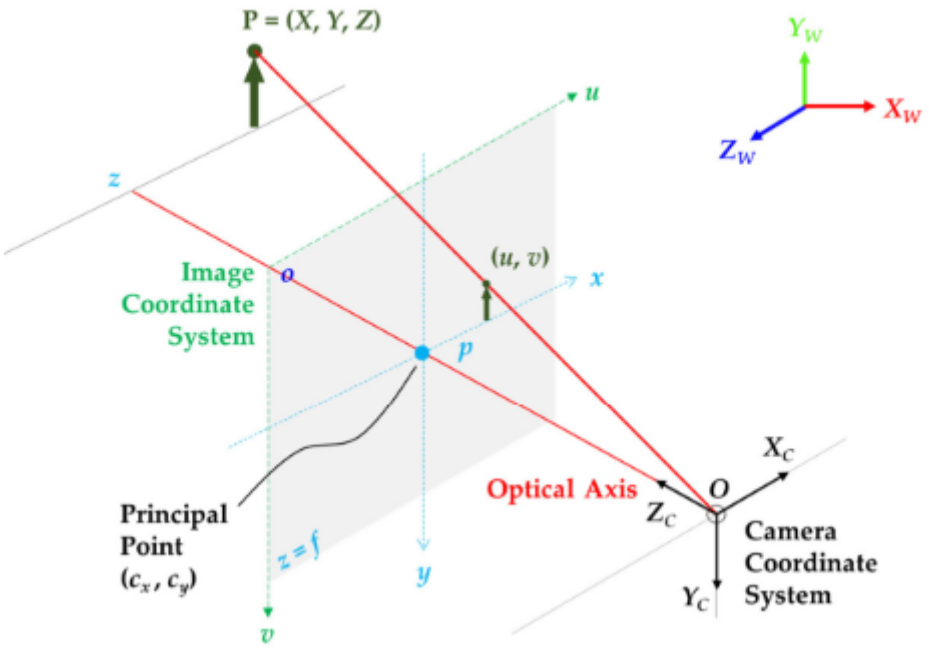
Variabler	**Beskrivning**
A_k	Tillståndsövergångsmatris
B_k	Styrsignalsmatris
u	Styrsignal
Q	Kovariansmatris med processbrus
z	Mätvärde
H	Mätmatris
R_k	Kovariansmatris med mätbrus
I	Identitetsmatrisen
P	Kovariansmatris
Index	**Beskrivning**
k	Aktuellt tidssteg
$k\|k-1$	Variabel beroende av tidigare tidssteg

2.4 Detektion av ArUco-markörer

Detektion av ArUco-markörer är en metod för att uppskatta kameraposition [17]. Detta åstadkoms genom användningen av binära fyrkantiga fiducialmarkörer vilket är en typ av märkning som används vid bildbehandling och datorseende. Genom att analysera positionen och orienteringen av en markör i kamerans bild kan dess exakta position och riktning i det verkliga rummet bestämmas. För att kunna detektera Aruco-markörer och bestämma kamerans position är det första steget att

kalibrera kameran. Detta görs genom att använda ett ChArUco-bräde, som är en kombination av ett Aruco-bräde och ett schackbräde.

Kamerakalibreringen innebär att man erhåller kamerans intrinsiska parametrar och distorsionskoefficienter [18]. De instrinsiska parametrarna består av f_x och f_y som är brännvidden uttryckt i pixlar samt c_x och c_y som representerar referenspunkten (principal point). Att använda ChArUco för kalibrering är betydligt mer mångsidig än att använda traditionella schackmönster, eftersom det tillåter ocklusioner eller partiella vyer. Dessutom noteras att kamerakalibreringen bara behöver utföras en gång, eftersom de intrinsiska parametrarna och distorsionskoefficienterna förblir oförändrade, såvida inte optiken på kameran ändras.



Figur 2.3: Hålkameramodell [19]

För att omvandla 3D-punkter till 2D-koordinater på en bild används en modell känd som hålkameramodell som visas i Figur 2.3 [20]. I denna modell utförs den geometriska projektionen av en kamera med ett enda hål utan några fokallinser. En 2D-punkt $\boldsymbol{p}$ på bildplanet skapas genom att multiplicera kameramatrisen $\boldsymbol{C}_m$ med den motsvarande 3D-punkten $\boldsymbol{P}_w$ enligt

$$\boldsymbol{p} = \boldsymbol{C}_m \boldsymbol{P}_w, \tag{2.13}$$

där $\boldsymbol{p} = [u, v, 1]$ och $\boldsymbol{P}_w = [X_w, Y_w, Z_w, 1]$ är representerade i homogena koordinater [20]. Kameramatrisen består av inre och yttre parametrar som beräknas med hjälp av OpenCV och biblioteket `ArUco`. Ekvationen för kameramatrisen ges av

$$\boldsymbol{C}_m = \boldsymbol{A}_i[\boldsymbol{R}|\boldsymbol{T}], \tag{2.14}$$

A_i är den inre matrisen som representeras som en 3x3 matris enligt

$$\boldsymbol{A}_i = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.15}$$

samt den yttre matrisen $\boldsymbol{R}$ representeras av en 3x3 matris med elementen r_{mn} där m är radindex och n är kolumnindex, följt av en 3x1 translationmatris. Med hjälp av Ekvation 2.13-2.15 kan den utökade Ekvationen uttryckas enligt

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \tag{2.16}$$

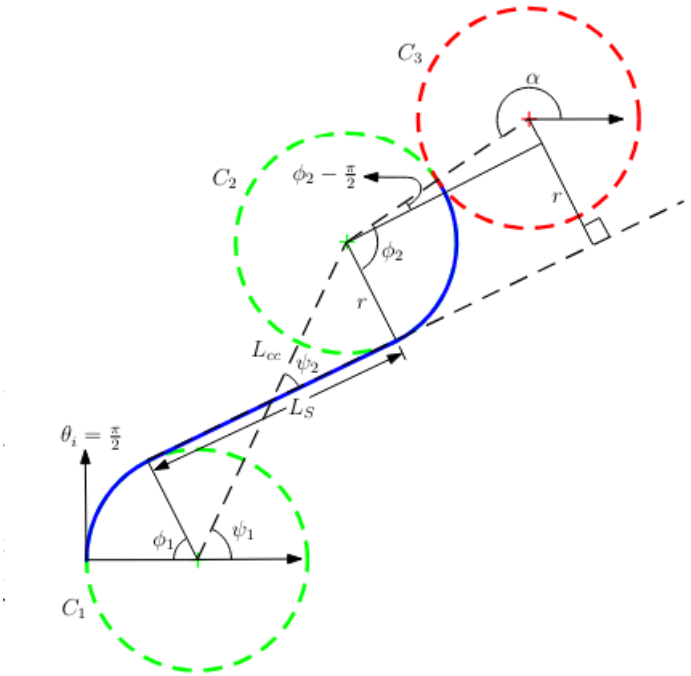
Efter att kamerans parametrar och distorsionskoefficienter har erhållits kan nu en 3D-position uppskattas för kameran i förhållande till en enskild ArUco-markör [21]. Detta utförs med hjälp av den inbyggda funktionen *estimatePoseSingleMarker* från OpenCV. Funktionen returnerar translations- och rotationsvektorerna som krävs för att omvandla en 3D-punkt från objektkoordinatsystemet till kamerakoordinatsystemet [22]. Med dessa vektorer kan man bestämma hur markören är placerad och orienterad i förhållande till kameran, vilket möjliggör exakt positionering av objekt i den verkliga världen baserat på deras detektion i kamerans bild.

2.5 Dubins path

Dubins path är en välkänd vägplaneringsalgoritm som har fått bred användning inom området för autonoma fordon [23]. Syftet med algoritmen är att effektivt finna den kortaste vägen från en given startposition och riktning till en angiven målposition och riktning. Denna algoritm har visat sig vara av stor betydelse för att möjliggöra smidig och pålitlig navigering för autonoma fordon, och den har använts i en mängd olika tillämpningar och situationer.

Dubins Path-algoritmen används för att beräkna den kortaste vägen till en given position med hjälp av fordonets maximala krökning vid maximalt styrutslag [23]. Denna metod resulterar i en väg bestående av högst tre segment, där varje segment kan vara antingen en höger (H) eller vänster (V) sväng, eller en rak linje (R). Detta ger sex möjliga kombinationer av vägsegment: HRH, VRV, HRV, VRH, HVH eller VHV.
En illustration av en HRV-väg genererad av Dubins Path-algoritmen visas i Figur 2.4. För att generera denna väg antas initialpositionen vara vid origo med utgående riktning längs den positiva y-axeln, vilket resulterar i en initial riktningsvinkel $\theta_i = \pi/2$ [23]. Slutpositionens koordinater (x, y) och slutriktningen θ bestäms därefter. Avståndet L_{cc} mellan de två cirkulära delarna av vägen beskrivs enligt

$$L_{cc} = \sqrt{(x - r\sin\theta - r)^2 + (y + r\cos\theta)^2}. \tag{2.17}$$



Figur 2.4: HRV väg [23].

Vidare kan φ_1, φ_2 samt L_S skrivas till

$$L_S = \sqrt{L_{cc}^2 - 4r^2}, \tag{2.18}$$

$$\varphi_1 = \text{mod}(-\psi_1 + \psi_2 + \frac{\pi}{2}, 2\pi), \tag{2.19}$$

$$\varphi_2 = \text{mod}(\theta + \varphi_1 - \frac{\pi}{2}, 2\pi), \tag{2.20}$$

där ψ_1 och ψ_2 beskrivs enligt

$$\psi_1 = \text{atan2}\left(\frac{y + r\cos\theta}{x - r\sin\theta - r}\right), \tag{2.21}$$

$$\psi_2 = \text{atan2}\left(\frac{y + r\cos\theta}{x - r\sin\theta - r}\right). \tag{2.22}$$

Slutligen för att uttrycka L_{cc}, ψ_1 och ψ_2 med de nya variablerna $(x, y) = (c + r\cos\alpha, r + \sin\alpha)$ och $\theta = \alpha - \frac{\pi}{2}$, kan följande uttryck användas

$$Lcc = \frac{p}{(c + 2r\cos\alpha - r)^2 + (d + 2r\sin\alpha)^2}. \tag{2.23}$$

$$\psi_1 = \arctan\left(\frac{d + 2r\sin\alpha}{c + 2r\cos\alpha - r}\right), \tag{2.24}$$

$$\psi_2 = \arctan\left(\frac{2r}{L_S}\right). \tag{2.25}$$

Genom att utföra liknande beräkningar för de återstående vägsegmenten tillämpas samma metodik och strategi för att effektivt planera vägen genom hela vägsystemet.

3
Metod för simulering

I detta kapitel behandlas metoden för simuleringar och utvecklingsprocesser, där de processer som används för att utveckla systemet noggrant beskrivs.

3.1 Carla - för simulation av vägföljning

Carla är ett Unreal Engine 4 baserat öppenkodsimuleringsverktyg som är konstruerat specifikt för simulering av autonoma fordon. Carla är flexibelt med fordonstyper och simulationsmiljö, vilket gör det skräddarsytt för genomförande av projektet.

Genom en API, även kallat "Application programming interface" går det att manipulera en simulation för passa det avsedda ändamålet. Genom Python API går det exempelvis att ladda in olika banor, välja vilket eller vilka fordon som ska användas i simulationen, varav varje fordon har egna fysikaliska egenskaper framtagna för att efterlikna ett realistiskt fordon. Dessa egenskaper är exempelvis maximala varvtal, massa och tröghet i hjul som samverkar för att skapa ett så realistisk simulationsscenario som möjligt. Dessa faktorer påverkar acceleration, toppfart och bromstid för respektive fordon.

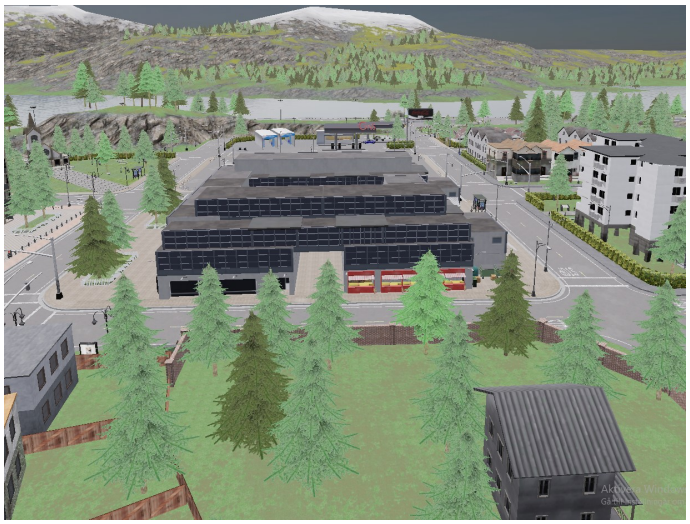
Python API ger upphov till att operatören av simulationen ska ha så mycket frihet som möjligt i att manipulera simuleringsomständigheterna på ett sätt som avser ändamålet. Vid skapande av en ny aktör som det kallas i Carla, användes en brandbil för att efterlikna en lastbil så mycket som möjligt, samt en monterad kamera för att möjliggöra detektion av väglinjer med hjälp av den konstruerade vägföljningsalgoritmen.

Figur 3.1: Brandbilen som användes i Carla

För att kalibrera parametrar i vågföljningsalgoritmen för att optimalt kunna följa helsträckade vita linjer, användes först Carlas egen autopilot funktion. Denna fungerar genom att Carla styr fordonet i simulationen med hjälp av ett större omfång av olika sensorer. Genom att använda autopiloten, följer fordonet vägbanan i princip felfritt. Därmed är autopiloten central i månen av att styrningen från autopiloten kan användas som en referens kring hur den egen implementerade styrningen bör se ut när den är färdigställd.

Utöver analys av färdig produkt är autopiloten essentiell i processen av att testa vägföljningsalgoritmens kapacitet att detektera och rita adekvata linjer i vägbanan. Detta utförs genom att kameran fästs på fordonet och bilderna från kameran processeras av algoritmen för att få ut bilder där följningslinjerna är applicerade. Genom att använda Carlas autopilot underlättas processen av att justera parameterar. När koden är tillräckligt välanpassad för simulationsmiljön implementerades istället en egen styrningsalgoritm med hjälp av en PID regulator.

Eftersom att Carla är Unreal Engine 4 baserat, innebär det att programmet är väldigt beräkningsintensivt vilket ställer höga krav på hårdvara. Banorna som redan finns färdiga i Carla, emulerar statsmiljöer då dessa generellt är eftersträvade för tester av autonoma fordon. Dessa stadsmiljöer innefattar allt från stora byggnader, till vegetation, trafikskyltar, vägar och trottoarer. För att simulationsresultaten ska bli någorlunda rättvisande gäller det att hårdvaran orkar med den specifika uppdateringsfrekvensen på kameran. För att avlasta grafikkort och processor skapades då en funktion för att eliminera onödiga element i miljön. Eftersom att Python API är såpass omfattande, finns det alternativ för att stänga av ett urval av element i miljön. Funktionen tar bort byggnader, staket, vegetation och annat onödigt som är oväsentliga för simulation av linjeföljning. Denna funktion körs i samband med simulationskoden för att hårdvaran ska få vila och ge mer kraft till själva simulationsexekveringen. För att säkerställa att det inte kostar för mycket på hårdvaran att ta bort dessa miljöfaktorer, har det även implementerats en tidspaus, där huvudprogrammet vilar i några sekunder innan huvudprogrammet kör själva simulationen.



Figur 3.2: Orginalmiljö

Figur 3.3: Optimerad miljö

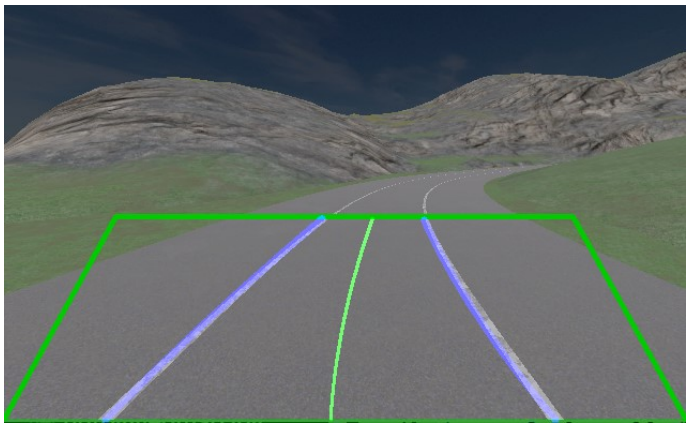
Utöver att optimera bildfrekvensen i själva hårdvaruaspekten med hjälp av den ovannämnda funktionen, eftersträvas även en simuleringsmiljö som är någorlunda lik den simulationsmiljö som ska användas i praktiken. Med hjälp av Python API valdes en karta över en stad som hade en större mängd av vita heldragna väglinjer än övriga. Det går i koden att specificera begynnelsekoordinater för var fordonet ska påbörja simulationssekvensen. Detta testades fram genom att slumpmässigt välja koordinater på olika platser till dess att simuleringskoordinaterna ansågs välanpassade för syftet. Därefter valdes samma koordinater varje gång. Simulationsmiljön utformades till sist så att fordonet får köra i en lätt kurva med heldragna vita linjer, vilket är väldigt snarlikt det som ska utföras vid fysisk tillämpning.

Figur 3.4: Brandbil kör i vald simuleringsmiljö

3.1.1 Region of interest och färgfiltrering

Bilderna från kameran dimensioneras med hjälp av antalet pixlar i både horisontal och vertikal riktning, där en bildbredd respektive en bildhöjd definieras. Utifrån detta skapas sedan en parallelltrapets eller rektangel som definierar ett spann av pixlar inom bilden som är intressanta att rita linjer på. Storleken på region of interest även kallat ROI kan varieras, men beroende på storleken kommer linjerna som kalkyleras vara olika stora samt medelvärdet ljusintensiteten i bilden är direkt beroende av storleken på området. För att skapa en bättre och mer adaptiv färgfiltrering delades detta område upp i två sektioner. Detta var av nytta eftersom varje ny sektion omfattande varsin väglinje. Detta gör det lättare att detektera större skillnader i ljusspektrumet mellan de två sektionerna som i sin tur gör att olika undre färgtrösklar behöver plockas fram för respektive sektion. Genom att införa denna uppdelning av ROI resulterar det i att koden blir mer adaptiv till olika ljusförhållanden, vilket i sin tur gör algoritmen mer robust.

En annan viktig anledning till att definiera ett ROI är att hela bilden inte är av intresse. Exempelvis går det att säkerställa att det som är av intresse faktiskt är en del av vägbanan, och inte delar av andra miljöelement som är irrelevanta för vägföljning. Att bara processera ett intressant område i bildspannet filtrerar ut brus, men resulterar också i att algoritmen inte behöver ta hänsyn till hela bilden, vilket i sin tur reducerar beräkningsintensiviteten i vägföljningsalgoritmen.



Figur 3.5: Bild från kameraflöde av brandbil i simulation, med ett ROI definierat med tjocka gröna linjer i form av ett parallelltrapets.

Eftersom avgränsningar i projektet innefattade att fordonet ska kunna se och förhålla sig till vita väglinjer implementerades ett färgfilter i vägföljningsalgoritmen. Då är det onödigt att detektera kanter i ett färgspektrum utanför ramarna av intresse. Det skapas därför ett intervall mellan en nedre respektive övre färgtröskel. En övre tröskel definieras som [255, 255, 255] vilket representerar vit färg i RGB. Den nedre tröskeln beror på omgivningen, men eftersom det är vita linjer som är intressanta ligger värdet oftast mellan [200, 200, 200] och [240, 240, 240] beroende på omgivningens ljusförhållanden. Med hjälp av ett uppdelat ROI, bestäms den nedre tröskeln på vardera sida av ROI separat. Den primära aspekten som påverkar dessa trösklar är dels färgen på väglinjerna och underlaget, men också ljusförhållanden i rummet. Det kan exempelvis vara så att det är ljusare på ena sidan i rummet och mörkare i andra. Detta tillämpas därför för att göra algoritmen mer adaptiv och robust.

3.1.2 Beräkning av referenslinje och framtagning av styrvinkel

Hough transformen, se Avsnitt 2.1, tar fram raka linjer genom en röstningsprocess. Dessa linjer delas sedan upp i två kategorier, högersida, och vänstersida. Denna indelning sker genom genom att beräkna linjens lutning. Om lutningen är positiv kommer det ge upphov till en linje i vänsterfältet, och motsatsen för högerfältet. Däremot beräknas bara raka linjer, vilket skapar problem i kurvor. Det är viktigt att kunna anpassa detekteringen av väglinjer på ett sätt som inte bara uppmuntrar en rak vägbana. För att kunna hantera detta skapas en funktion i Python som omvandlar linjerna till ett andragradspolynom. Funktioner som används för detta är *Polyfit* och *Poly1d* som är delar av `numpy` paketet i Python.

Polyfit i `numpy` används för att ta fram koefficienter som bäst passar andragradspolynomet $x = Ay^2 + By + C$ genom minsta kvadratmetoden [24]. I sammanhanget är x den beroende variabeln och y den oberoende. *Polyfit* kan även användas för att hitta koefficienter till polynom av högre grad, men dessa ansågs inte att bidra till någon signifikant förbättring av kvaliteten på de genererade linjerna. Högre grader

av polynom resulterar också i att funktionens beräkningsintensivitet ökar, vilket inte är önskvärt för en snabb och effektiv vägföljningsalgoritm. Det skapades sedan för både höger- och vänsterlinje en gemensam uppsättning av y-värden, med hjälp av de redan existerande datapunkterna i höger och vänsterlinje. *Poly1d* användes för att ta fram x-värden i ekvationen för både vänster och högerlinje [25]. Detta resulterade i två andragradspolynom som ger upphov till kurvor i de detekterade väglinjerna. För att detta ska bli bra införs en parameter sufficient fit som determinerar ett minimum för hur många punkter som ska kunna utgöra en andragradskurva. Ett högre värde på sufficient fit resulterar i bättre kurvor.

Referenslinjen beräknas genom att först generera en uppsättning av y-värden som härstammar från listor av y-värden i vänster respektive höger linje. Python funktionen *linspace*, i `numpy` biblioteket, används för att skapa en lista av allmänna y-värden där startvärdet är det minsta värdet och slutvärdet är det högsta. Baserat på de nya y-värdena skattas x-värden i vänster och högerfil med *Poly1d* från `numpy` biblioteket i Python. Ett medelvärde av dessa evaluerade x-värden tas i sin tur fram och en linje dras med koordinater från de allmänna y-värden och de nya framtagna x-värdena. Därmed genereras en referenslinje i form av ett andragradspolynom och efterliknar ett medelvärde av de genererade väglinjerna.

För att beräkna en styrvinkel valdes först en punkt på referenslinjen. Denna punkt kallas vidare i texten för måltavla. Vid detta y-värde beräknades motsvarande x-värde för att sedan jämföras med x-värdet som representerar mitten av bildbredden. Differensen mellan dessa två värden resulterar i ett fel som går att räkna på.

Vid framtagning av styrvinkel så testades två metoder. Den första visade sig vara för beräkningsintensiv och därmed ett sämre alternativ. Den första metoden gick ut på att vid måltavlan beräkna en tangent längs referenslinjen. Tangenten kommer i sin tur skära x-axeln i ett specifikt y-värde. Skillnaden mellan måltavlan och detta y-värde resulterar då i en bas i en rätvinklig triangel. Felet mellan x-värdet i måltavlan och mitten av kameran representerar i sin tur en höjd. En vinkel genereras då genom att beräkna vinkeln mellan höjden och basen.

Den andra metoden gick ut på att bara använda felet mellan måltavlans x-koordinat och mittenlinjen. Följande formel användes

$$\theta = \arctan(error \cdot k), \tag{3.1}$$

där k representerar ett förhållande mellan höjd och bas. En fördel med den andra metoden utöver reducerad beräkningsintensivitet är också att det går att skala felet baserat på hur finkänslig styrningen bör vara.

3.1.3 Implementering av PID regulator i Carla

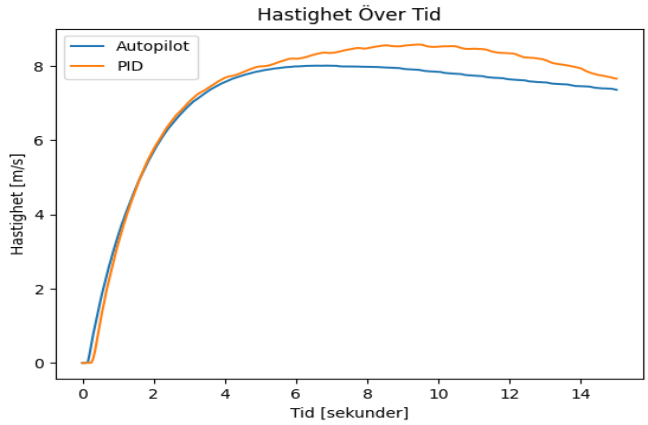
En ”Proportional Integral Derivate” regulator även kallad PID konstruerades. För att beräkna fram ett fel till regulatorn behövdes referensvinkeln från Ekvation 3.1. Utöver referensvinkeln behövdes även fordonets egna styrvinkel. Fordonets styrvin-

kel genererades med hjälp av inbyggda funktioner i Carla. Differensen mellan referensvinkleln och den nuvarande styrvinkeln på fordonet skickas in i regulatorn som i sin tur justerar styrningen baserat på detta värde. Styrningens begynnelsevärde är noll. Regulatorn ställdes in för att uppdateras tjugo gånger per sekund, vilket korresponderade till samma bildfrekvens som på kameran. Värdet tjugo bilder per sekund valdes för att beräkningsintensiviteten blev för hög vid högre antal bilder per sekund. Att använda en bildfrekvens som är relativt låg garanterar att regulatorn och kameran kan arbeta synkroniserat.

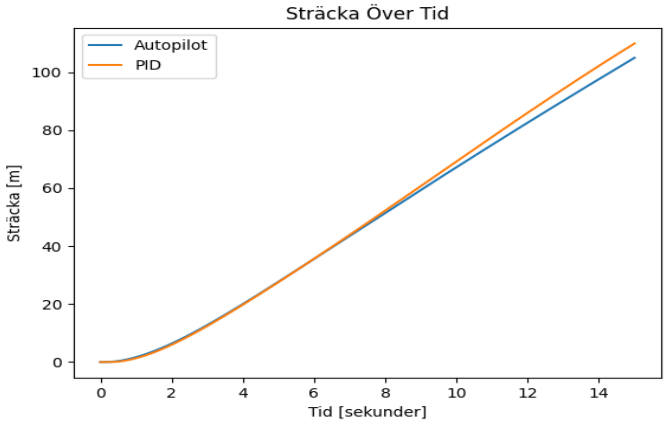
I simulationen testades olika värden på skalfaktorn k för att undersöka och jämföra systemsvaren mellan simulationsfordonet under Carlas autopilot och under körning med den egen implementerade PID regulatorn. Eftersom att Carlas autopilot fungerar i alla miljöer och också nyttjar en regulator kan autopiloten användas som ett börvärde för den egenkonstruerade PID regulatorn. Olika värden på k jämfördes för att resultera i en regulator som liknar systemsvaren från Carlas autopilot.

Carlas egen autopilot anpassar brandbilens acceleration efter omgivningen på ett mer sofistikerat sätt än den egenkonstruerade regulatorn. Autopiloten är försiktigare med acceleration vid kurvtagning för att det aktuella fordonet inte ska åka av vägen. Även regionella hastighetsbegränsningar som är associerade med specifika regioner på kartan tas till hänsyn. Accelerationen vid olika punkter i simulationsförloppet var med de två regleringsmetoderna inte identiska, detta resulterade i att det blev svårt att jämföra styrvinklarna vid specifika koordinater i simulationsbanan. För att då kunna jämföra styrvinklar på ungefär samma platser i simulationen har skillnader i hastigheter mellan de de två metoderna jämförts och integrerats. Med hjälp av integration genereras sedan en sträcka som respektive styrningsmetod har kört under simuleringen. I de punkter där det skiljer mindre än fem procent mellan de två punkterna har styrvinkeln jämförts. Dessa diskreta tidspunkter säkerställdes att vara inom ett kontinuerligt tidsintervall för att grafen också skulle kunna läsas som kontinuerlig.

I samtliga resultat har samma värden på PID parametrarna i regulatorn använts. Detta eftersom att en förändring av PID parametrarna inte var nödvändigt för att följa en vägbana med olika värden på skalfaktorn k. Därmed är skalfaktorn av felet vid den kalkylerade referensvinkeln den intressanta parametern i följande framtagna resultat.

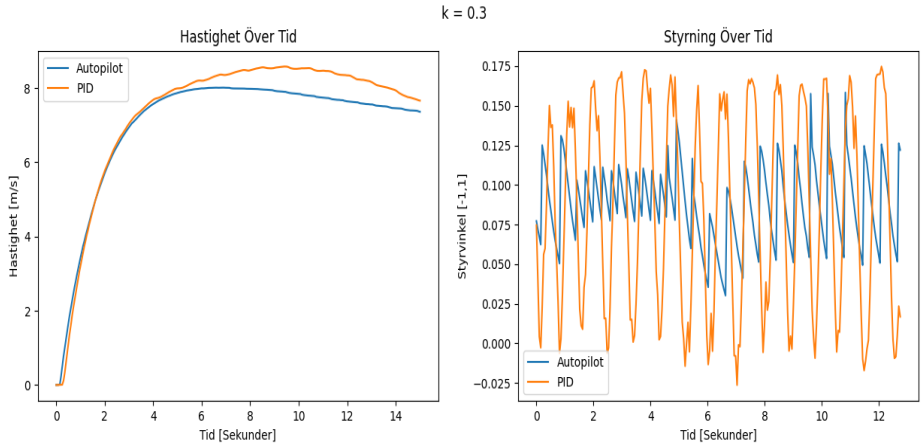


Figur 3.6: Arean under graferna representerar den köra sträckan för respektive styrningsmetod.



Figur 3.7: Resultat av integration av graferna i Figur 3.6.

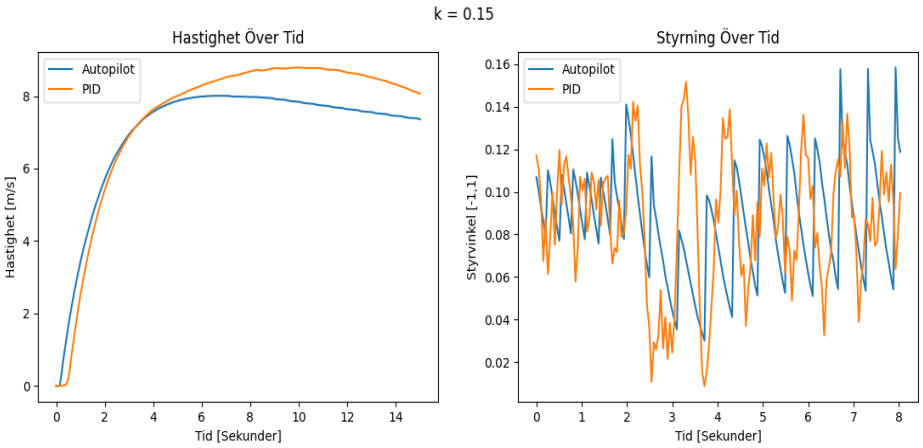
Figur 3.6 och 3.7 är framtagna med en skalfaktor $k = 0.3$. För samtliga testade värden på skalfaktorn k är figuren för sträcka över tid nästan identisk, därför redovisas bara hastighet kontra styrning över tid i resterande grafer. Styrvinkeln i Figur 3.8, 3.9 och 3.10 härstammar från Ekvation 3.1 där felet beräknas med hjälp av differensen i antal pixlar mellan mittpunkten och måltavlan. Det beräknas sedan som en vinkel uttryckt i radianer, som är enheten på styrvinkeln illustrerat i figurerna.



Figur 3.8: Hastighet respektive styrvinkel över tid vid $k = 0.3$. Tidsintervallet för styrningen över tid är intervallet då körsträckan över tid för de respektive styrmetoderna är inom en avvikelse på maximalt fem procent.

Resultaten i Figur 3.8 visar skillnaden i systemsvar på när brandbilen körs i autopilot respektive med den egenkonstruerade PID regulatorn. Figuren illustrerar att den egenkonstruerade regulatorn resulterar i ett starkare systemsvar. Detta innebär att styrvinkeln förändras kraftigare med den egna regulatorn gentemot Carlas autopilot. Det är däremot tydligt att frekvensen är högre med autopiloten, vilket då innebär att regulatorn gör förändringar oftare, men inte lika stora som med den egenkonstruerade regulatorn.

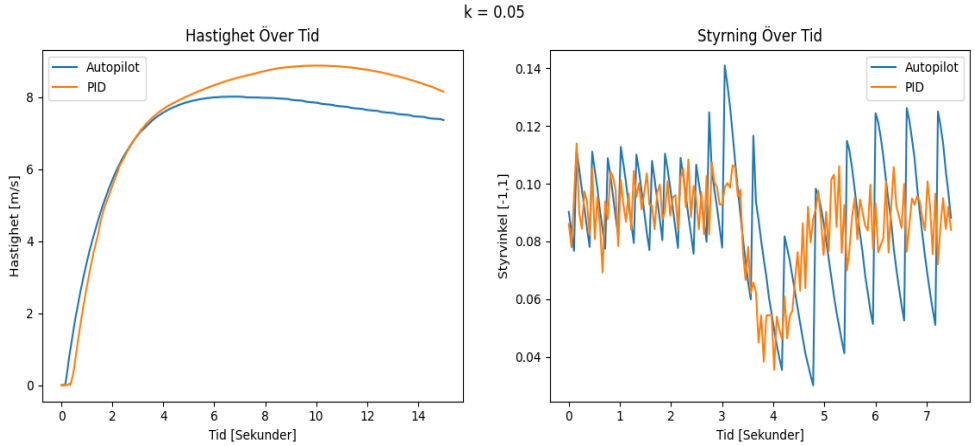
Hjulen på den PID styrda brandbilen oscillerar också mer än vad Carla autopiloten gör. Detta kan ses i amplitudtopparna i den högra grafen i Figur 3.8. Det går också att observera oscillationerna i styrningen i hastighetsgrafen. Detta innebär att regulatorparametrarna i sig hade kunnat optimeras så att styrvinkeln inte gör små oscillationer på korta intervall.



Figur 3.9: Hastighet respektive styrvinkel [rad] över tid vid $k = 0.15$.

I Figur 3.9 illustreras samma process som konstaterats tidigare med $k = 0.15$. Tidsintervallet där körd sträcka är inom ett marginal på fem procent har minskat. Även

amplituderna för PID regulatorn har reducerats. I figuren är amplituderna nu mer i linje med vad Carlas autopilot åstadkommer. I hastighetsgrafen är det också tydligt att körningen är lite jämnare än respektive graf i Figur 3.8



Figur 3.10: Hastighet respektive styrvinkel över tid vid $k = 0.05$.

När $k = 0.05$ som illustrerat i Figur 3.10 blir systemsvarets amplituder från PID regulatorn lägre än Carlas autopilot. Frekvensen av förändringen i styrvinkeln har även ökat vid användning av PID regulatorn. Styrningen gör små förändringar oftare. I den vänstra grafen för hastighet över tid är det också tydligt att hastigheten nu håller en jämnare linje än respektive graf i Figur 3.8 och Figur 3.9. Oscillationer har alltså minskat.

Figur 3.8 visar att den egenkonstruerade regulatorns systemsvar är av högre magnitud än Carlas autopilot. Detta kan härstamma från flera faktorer. Fundamentalt kan det bero på att styrvinkeln beräknas annorlunda och att det därmed ger upphov till större systemsvar. I Ekvation 3.1, beräknas styrvinkeln som är baserad på en skalfaktor k. Om k minskar resulterar det också i att den beräknade vinkeln minskar, vilket då kan resultera i ett mindre systemsvar. Därmed kan en metod för att åtgärda ett högt systemsvar vara att minska faktorn k som demonstreras i Figur 3.9 och 3.10. Graferna i Figur 3.9 och Figur 3.10 visar att ett optimalt värde för att efterlikna Carlas egna autopilot ligger mellan $k = 0.05$ och $k = 0.15$. I den valda simulationsmiljön där den virtuella brandbilen ska köra längs en lätt kurva fungerar ett lägre värde som $k = 0.05$. Däremot i andra sammanhang där brandbilen kan behöva ta skarpare svängar, kan detta visa sig vara ett för lågt värde. En högre proportionell del i regulatorn kan då visa sig vara essentiell för att kompensera för underskattningen av felet.

I Figur 3.8 är frekvensen i PID regulatorn lägre i systemsvaren i jämförelse med Carlas autopilot. Samtidigt observeras det att frekvensen på PID regulatorns systemsvar ökar i samband med ett lägre k värde. Detta samband mellan en ökning av frekvens i korrelation med en minskning av k-värdet demonstreras i Figur 3.9 och Figur 3.10. Detta beror på att PID parametrarna i regulatorn är konstanta för samtliga simuleringsfall. Resultatet blir då att den proportionella delen blir svagare i samband

med en reducerad skalfaktor k. Konsekvent påverkar det då systemsvaret genom att resultera i en högre frekvens av styrvinkelsförändringar med minskade magnitud på amplitudtoppar. Utöver frekvensskillnaden är Carlas autopilot också betydligt jämnare i sin respons. Detta tyder på att regulatorparametrarna i sig kan optimeras ytterligare för den egenkonstruerade regulatorn.

Däremot innebär detta inte att skillnaden i systemsvar nödvändigtvis behöver indikera att den egen implementerade regulatorn är suboptimal för sitt ändamål. Carlas autopilot är konstruerad för att dels kunna hantera högre hastigheter än vad dollyn ska köra i den fysiska testmiljön, men också för att hantera hinder och allmän stadskörning. Avgränsningarna i projektet ställer en grund för att även om den implementerade regulatorn inte är allsidig i sitt användningsområde, kan den fortfarande åstadkomma en fin och jämn körning i avsett ändamål. En autonom bil som kör i högre hastigheter kan behöva fler finjusteringar än en dolly som ska köras i en kontrollerad fysisk testmiljö, utan hinder och i låg hastighet. I avseende på projektet är det därmed viktigt att regulatorparamterarna samt skalfaktorn k, finjusteras mer i samband med den fysiska tillämpningen snarare än att eftersträva perfektion i Carlas simuleringsverktyg.

En slutsats efter test av olika k-värden är att små skillnader i skalfaktorn gör stora förändringar i systemsvaret, vilket är viktigt att förstå vid fysisk implementering. Vid samtliga undersökta k-värden klarade brandbilen att ta sig igenom simulationsbanan. Men den jämnaste körningen, med minst oscillationer i både brandbilens relation till mitten av väglinjerna och hjulplacering var när $k = 0.05$.

3.2 Simulering av avståndshållning

Simuleringen av avståndshållningen gjordes för att undersöka hur väl det gick att reglera systemet med de sensorer som skulle användas samt för att analysera de resultat som framkom av simuleringen för att bedöma snabbheten i systemet och huruvida det gick att reglera. Simuleringen skedde med hjälp av Matlab och Simulink. En dynamisk modell konstruerades som skulle modellera följebil och ledarbil. Se Appendix A.2. Modellen som användes visas i Figur 3.11 och är en kraft framåt ska simulera gaspedalen samt en bromsande kraft som kombinerar motstånd i form av friktion och vindmotstånd. Detta sattes upp som två olika system med två olika insignaler. Ledarbilen gavs en konstant hastighet och följebilen gavs styrsignalen u som input. Därefter plockades positionerna ut från de två systemen där följarbilens position subtraherades från ledarbilens position för att på så sätt få fram deras relativa position gentemot varandra. Detta användes som referenssignal som sedan kopplades in i ett Kalmanfilter och via en PID regulator återkopplades in i mot följarbilens insignal. För att efterlikna ultraljudssensorns mätbrus modellerades insignalen med ett normalfördelat brus pålagt över följebilens utsignal

3.2.1 Framtagning av tillståndsmodell och överföringsfunktion för avståndshållning

En dynamisk modell för avståndshållning mellan två ekipage behövdes för att kunna simulera avståndshållningen. Denna konstruerades som en massa som påverkas av en framåtriktad kraft samt en bakåtriktad kraft vilket illustreras i Figur 3.11.

Figur 3.11: Frikroppsdiagramet som användes som bas vid simulering.

Från denna modell skapas en differentialekvation utgående från Newtons andra lag, $\sum F = m\ddot{x}$. I uttrycket är b motståndet och m är massan av ekipaget. Motståndet och massan sattes experimentellt till låga värden för att försöka efterlikna de värden som gällde för den fysiska modellen. Detta resulterar i ekvationen

$$F(t) - b\dot{x}(t) = m\ddot{x}(t). \tag{3.2}$$

Denna ekvation kan skrivas om till

$$\begin{aligned} \ddot{x}(t) = \frac{-b}{m}\dot{x}(t) + \frac{F(t)}{m}, \\ y(t) = x(t), \end{aligned} \tag{3.3}$$

där utsignalen även har satts till modellens position. Laplacetransformeras ekvationerna i 3.3 fås

$$\begin{aligned} X(s) = \frac{1}{s}\frac{\frac{1}{b}}{(1+\frac{m}{b}s)}F(s), \\ Y(s) = X(s), \end{aligned} \tag{3.4}$$

Kraften $F(s)$ kommer från motorn som antas ha en enklare dynamik än fordonet i modellen. Detta gör att överföringsfunktionen från kraften $F(s)$ till styrsignal u modeleras som ett första ordningens system med statisk förstärkning K och tidskonstant T

$$F(s) = \frac{K}{1+Ts}U(s). \tag{3.5}$$

Om $F(s)$ ersätts i Ekvation 3.4 av Ekvation 3.5 ges följande uttryck

$$X(s) = \frac{1}{s}\frac{\frac{1}{b}}{(1+\frac{m}{b}s)}\frac{K}{1+Ts}U(s). \tag{3.6}$$

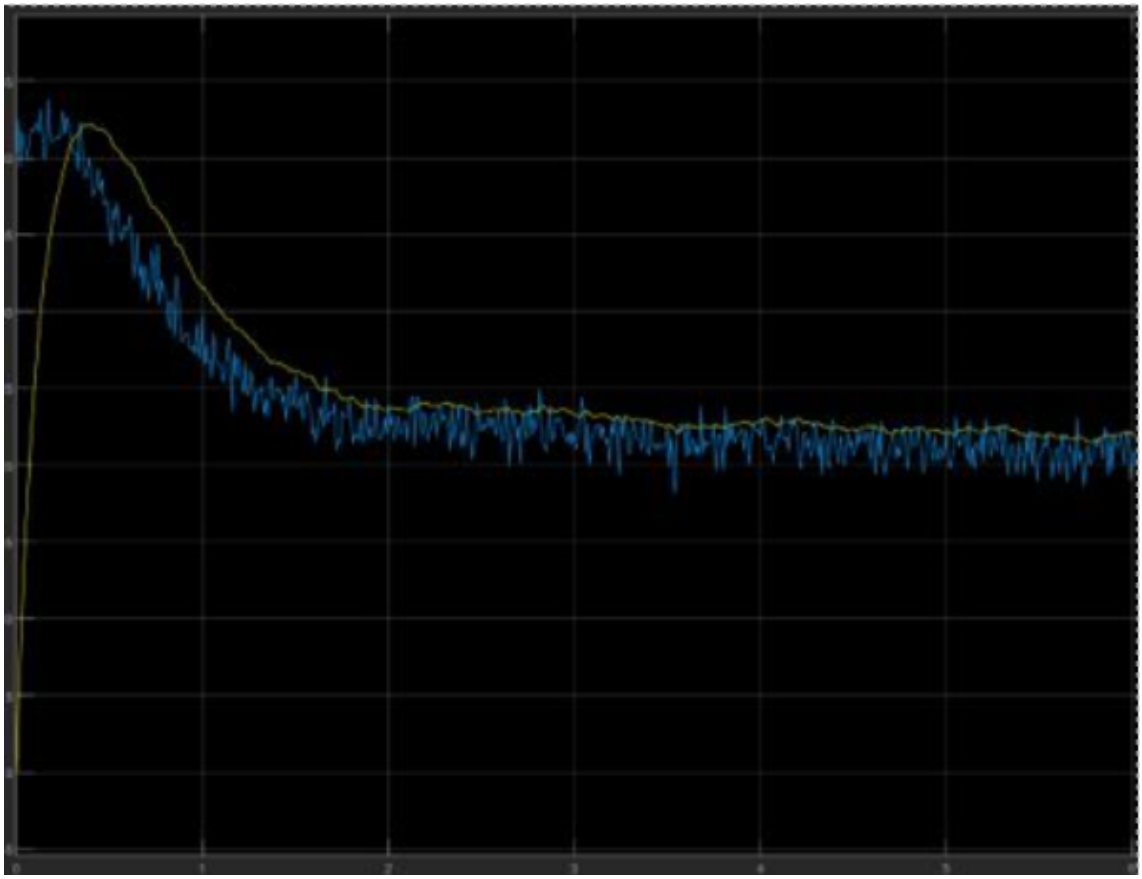
Efter ett utbyte av $X(s)$ till $Y(s)$ och division med $U(s)$ fås uttrycket

$$G(s) = \frac{1}{s} \frac{\frac{1}{b}}{(1 + \frac{m}{b}s)} \frac{K}{(1 + Ts)}, \tag{3.7}$$

där $G(s)$ är överföringsfunktionen från utsignal $Y(s)$ till styrsignal $U(s)$. Om vi antar att dynamiken i motorn är snabbare än den i systemet kan Ekvation 3.5 ersättas med bara $U(s)$. Detta resulterar i överföringsfunktionen

$$G(s) = \frac{1}{s} \frac{\frac{1}{b}}{(1 + \frac{m}{b}s)}, \tag{3.8}$$

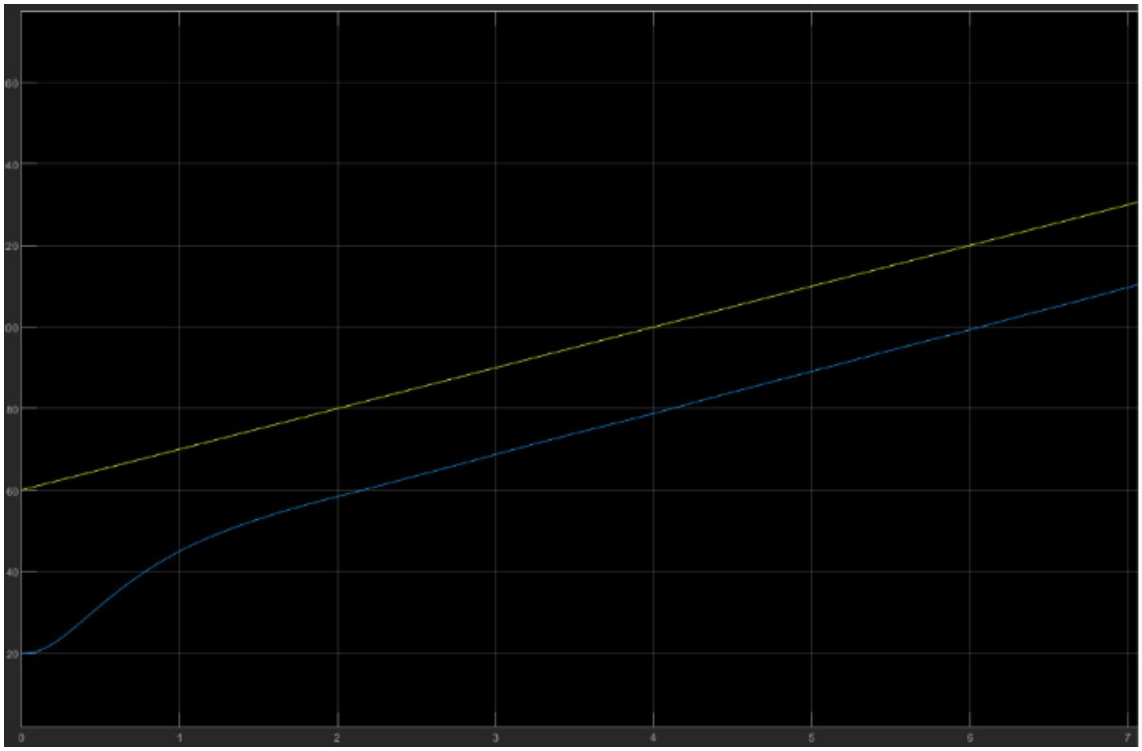
som användes i Simulink som systemmodell.



Figur 3.12: Följebilens hastighet över tid. Blå linje representerar följebilens hastighet innan Kalmanfiltrering och gul linje visar följebilens hastighet efter filtrering.

Figur 3.12 visar följefordonet hastighet över tid där simuleringens initialtillstånd består av ett stillastående följefordon och ett ledarfordon som kör med en konstant hastighet på 20 m/s. Resultatet blir att följefordonet accelererar snabbt för att hamna i position bakom ledarfordonet för att sedan bromsa in och sedan hålla en jämn hastighet. Figuren visar också att att Kalmanfiltreringen har en effekt på den

brusiga mätsignalen och gör den jämnare. Signalen innan filtrering är ett normalfördelat vitt, högfrekvent brus som varierar med ungefär ±3 enheter. Jämförs detta med signalen efter filtrering har mycket brus reducerats vilket innebär en jämnare hastighetshållning av följefordonet. Bruset som användes vid simulering visade sig överensstämma bra med bruset som erhölls vid mätningar från avståndsmätaren som användes i det verkliga systemet vilket ökar trovärdigheten av simuleringen.

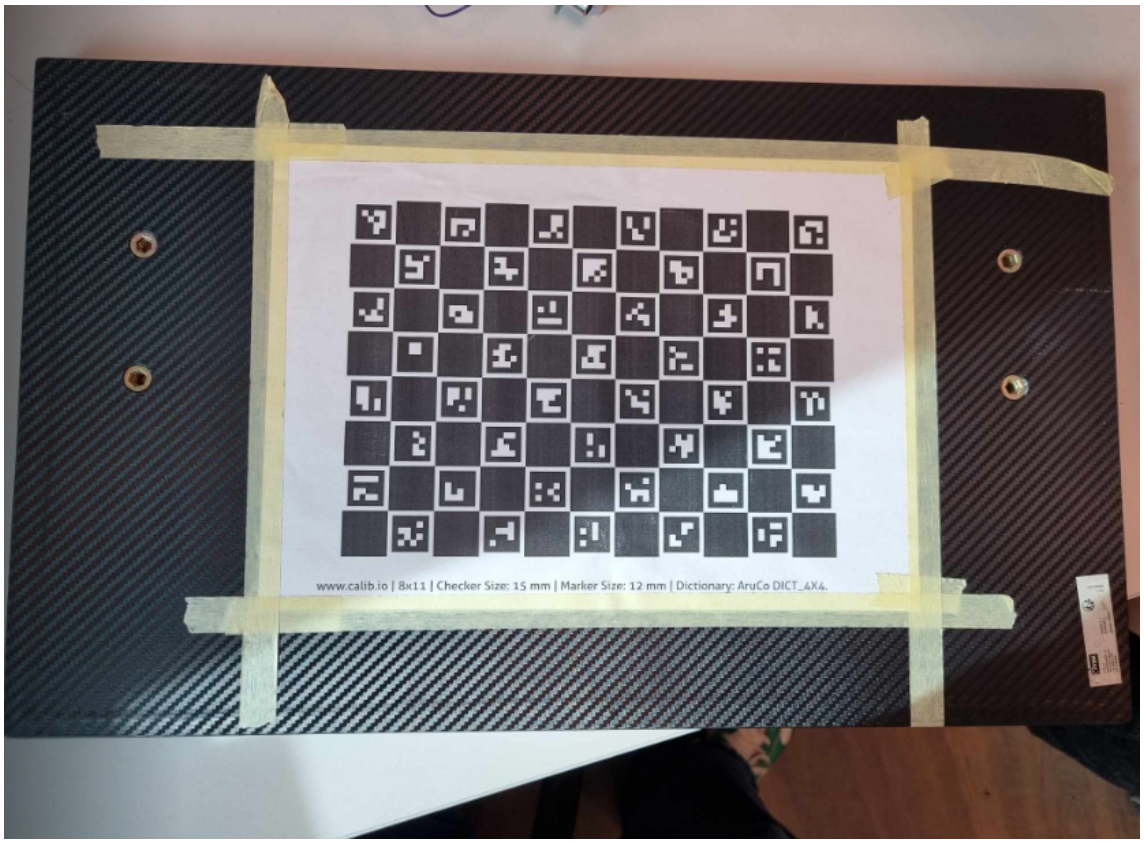


Figur 3.13: Jämförelse av position av följefordon och ledarfordon.

Ledarfordonet i simuleringarna startar på position 60 och följefordonet startar på position 20. Det önskade avståndet mellan dem är 20 enheter. Efter den initiala accelerationen av följefordonet under de första sekundrarna av simuleringen stabiliseras positionen till 20 enheter bakom ledarfordonet vilket visas i Figur 3.13. Stabiliseringen kring referenspunkten sker efter ca 1 sekund och avståndet hålls sedan jämt under resten av simuleringen. Dessa simuleringar visar att en reglering av avståndshållning är möjlig i en dimension med hjälp av en ultraljudssensor och en PID-regulator.

3.3 Detektion av ArUco-markörer

Vid implementeringen av ArUco-markörer studerades tidigare års arbete [7]. Däremot kunde ingen kod användas till följd av OpenCV som gjort uppdateringar i `ArUco` biblioteket där gamla funktioner ändras och nya funktioner tillkommit. Däremot kunde tidigare års ChArUco-bräde användas Figur 3.14 vid kamerakalibrering.



Figur 3.14: ChArUco-bräde från tidigare års arbete.

3.3.1 Kamerakalibrering med ChArUco-bräde

Det första steget involverade kalibrering av kameran för att möjliggöra positionerings- och orienteringsuppskattningar av en enskild ArUco-markör. Ett skript utvecklades i Python med hjälp av OpenCV:s bibliotek `ArUco`. Skriptet fungerade genom att rikta kameran mot ChArUco-brädet och samla in 50 bilder från olika vinklar på brädet. Därefter utfördes en kalibrering med hjälp av funktionen *calibrateCameraCharuco*. Denna funktion utnyttjade hörnen på ArUco-markörerna samt deras ID för att beräkna kameramatrisen, distorsionskoefficienter, rotationsvektorer, translationsvektorer och reprojectionsfel. De beräknade parametrarna sparades sedan i en NPZ-fil (NumPy ZIP) för enkel åtkomst och användning.

3.3.2 Detektering och positionsuppskattning av ArUco-markörer

Nästa steg involverade detektering av ArUco-markörer och identifiering av deras position i förhållande till kameran. För detta ändamål utvecklades ett nytt Python-skript där kameramatrisen och distorsionskoefficienterna hämtades från NPZ-filen.

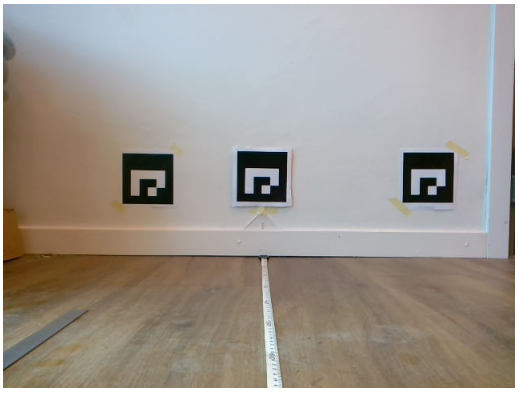
Genom att använda OpenCV:s *detectMarkers*-funktion kunde markörer identifieras med hjälp av kameran. Därefter användes *estimatePoseSingleMarkers*-funktionen för att erhålla rotations- och translationsvektorer. Funktionen använder sig av ka-

meramatrisen, distortionskoefficienterna och de detekterade markörerna från *detectMarkers* som indata. Translationsvektorn representerar markörens position i 3D-rymden med avseende på kameran, medan rotationsvektorn indikerar markörens rotation i förhållande till kameran, uttryckt genom rotation kring x-, y- och z-axlarna. Där x-axeln representerar horisontalledet, y-axeln representerar vertikalledet och z-axeln representerar djupet.

För att få ut gir, lutning och rullning från rotationsvektorn användes en ytterligare Python-funktion *rot2eul*. Funktionen tar rotationsvektorn som indata och beräknar sedan gir-, lutning- och rullning-vinkeln enligt en 3-2-1 Euler-sekvens, vilken är en metod för att konvertera en rotationsvektor till vinklar x-,y- och z-led. Gir-vinkel beskriver rotation kring y-axeln, lutning beskriver rotation kring x-axeln och rullning beskriver rotation kring z-axeln.

Under testningsprocessen utfördes endast mätningar på girvinkeln samt avståndet i x- och z-led. Ett av testen, som illustreras i Figur 3.15, visas två ArUco-markörer vinklade 90.0°mot varandra som mättes med en gradskiva. Resultatet redovisades i Tabell 3.1, där avvikelsen var 0.56°gentemot ArUco-funktionernas uppskattning. Ytterligare ett test för att mäta avstånd i x- och z-led visas i Figur 3.16. De faktiska avstånden mättes med en lasermätare och kontrollerades sedan med ett måttband för att jämföras med värdena beräknade med hjälp av ArUco-funktionerna. Mätningarna i x-led gjordes utifrån avståndet mellan markörerna dvs avstånden från mittenmarkören ut till sidomarkörerna, detta gjordes för att minimera felmätningar. Till följd av detta presenteras inget resultat av mittenmarkören i x-led. Inte heller presenteras något avstånd i z-led för sidomarkörerna då endast avståndet till mittenmarkörer noterades. Resultatet presenteras i Tabell 3.2.

Figur 3.15: Testmätning av girvinkeln mellan markörer.



Figur 3.16: Testmätning av avstånd i x-led och z-led för 3 markörer.

Tabell 3.1: Vinkelmätning av markörer.

	Vinkel
Faktiskt värde	90°
Uppmätt värde	90.56°

Tabell 3.2: Avstånd mellan markörer i x- och z-led.

	Fakiskt avstånd (cm)		Uppmätt avstånd (cm)	
	x-led	z-led	x-led	z-led
Markör vänster till mitten	30.0	-	30.2	-
Markör mitten	-	102.4	-	104.4
Markör höger till mitten	44.6	-	44.4	-

3.4 Simulering av Dubins path algoritmen

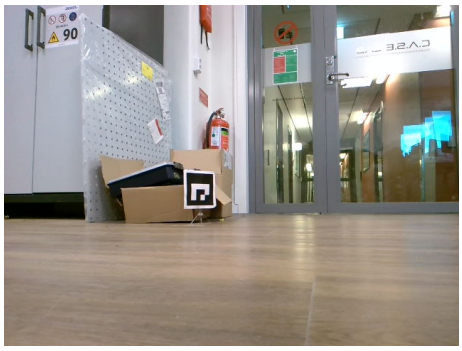
För simulering av Dubins path-algoritmen användes en implementation som hämtades från GitHub [26]. Denna kod utförde beräkningar av samtliga vägsegment och plottade den kortaste vägen till målet. Mindre justeringar gjordes för att anpassa koden till vårt specifika användningsfall. Första ändringen som gjordes var att beräkna dollyns krökning som ges av Ekvationen $K = \frac{1}{R}$ där R är bilens svängradie vid maximalt styrutslag. R i sin tur ges av

$$R = \frac{L}{\tan \delta_{max}}, \tag{3.9}$$

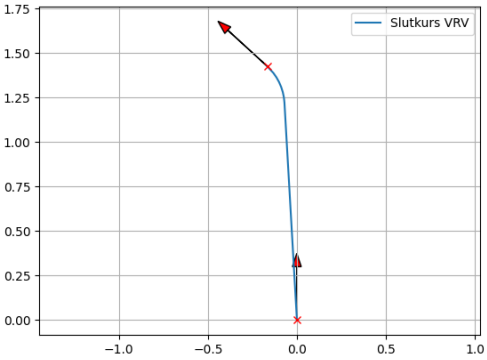
där L är axelavståndet mellan bilens hjul och δ_{max} är det maximala styrutslaget. Krökningen kan då beräknas till

$$K = \frac{\tan 40}{0.255} \approx 3.29. \tag{3.10}$$

Ytterligare justeringar som krävdes var att definiera start- och slutpositioner för dollyn. Vid startpunkten sattes dollyns position till (x, y) = (0, 0) med en svängvinkel på 90 grader för att säkerställa korrekt position relativt markören. För slutpositionen utnyttjades en ArUco-markör, vilket möjliggjorde mätning och användning av både x- och y-positionen samt svängvinkeln. Ett exempel på vägplanering med Dubins path kan ses i Figur 3.17.



(a) Bild tagen på en ArUco-markör för navigering.



(b) Planerad körbana till ArUco-markören genererad av Dubins path.

Figur 3.17: Illustration av en körbana från utgångspunkten till AruUo-markören, med hjälp av Dubins path för ruttplanering.

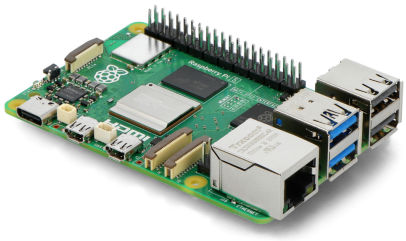
4

Fysisk Implementering

I detta kapitel presenteras den praktiska tillämpningen av de algoritmer som tidigare simulerats, på en fysisk dolly. Detta innefattar huruvida vägföljningsalgoritmen och avståndshållningssystem integreras och implementeras på en dolly, med hjälp av en Raspberry Pi 5-mikrodator som centralenhet.

4.1 Raspberry Pi 5

För att kontrollera styrning och motorn på dollyn användes mikrodatorn Raspberry Pi 5. En av huvudorsakerna till att den valdes för projektet var på grund av dess förmåga att hantera tunga beräkningar. Detta ansågs nödvändigt då vägföljningsalgoritmen kräver betydande processorkraft för att i realtid analysera videoströmmar, identifiera vägmarkeringar och utifrån det tillhandahålla nödvändig styrning. Mikrodatorn kommer även att behöva hantera flera processer samtidigt då systemet kräver konstant uppdatering och respons från olika sensorer och inputenheter för att körningen ska ske optimalt. Därför lades stor vikt på en kraftfull processor vid valet av mikrodator. Raspberry Pi 5 erbjuder även en generös mängd av GPIO-pinnar vilket kan ses i Tabell 4.1. Detta är väsentligt för skalmodellens expansionsmöjligheter.

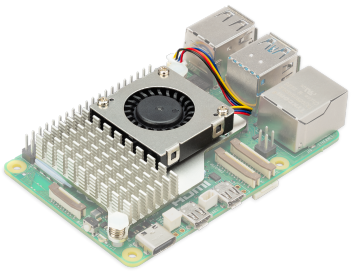


Figur 4.1: Raspberry Pi 5 [27].

Tabell 4.1: Specifikationer Raspberry Pi 5 [28].

Raspberry Pi 5	
RAM-minne	8 GB
Processor	64-bit Quad-core ARM Cortex-A76
Digitala I/O	40
Lagringsutrymme	MicroSD-kortplats
Anslutningar	2st USB 3.0, 2st USB 2.0, USB-C (5V/5A), 2st Micro-Hdmi, Gigabit Ethernet

Eftersom Raspberry Pi 5 har en högpresterande processor finns risken att den kan bli överhettad. För att bibehålla prestanda och minimera slitage investerades det i en aktiv kylare till mikrodatorn. Kylaren består av en temperaturstyrd fläkt och ett parti med aluminiumkylflänsar vilket kan ses i Figur 4.2.



Figur 4.2: Active cooler för Raspberry Pi 5 [29].

4.1.1 Utmaningar med Raspberry Pi 5

Under tiden som hårdvarugruppen arbetade med att konstruera den nya dollyn, genomfördes tester av algoritmer och liknande på den dollyn som hade utvecklats under tidigare år. Mikrodatorn som användes på den gamla dollyn var en Jetson Nano och för programmering användes Python ihop med `Rpi.GPIO`. `Rpi.GPIO` är ett bibiotek i Python som är designat för att kontrollera GPIO-pinnarna på en Raspberry pi [30]. Detta bibliotek användes för att underlätta en smidig övergång till en Raspberry Pi 5 (RPi 5) i framtiden samt att `Rpi.GPIO` var kompatibelt även med Jetson Nanon. När algoritmerna sedan skulle implementeras på den nya RPi 5 uppstod det problem.

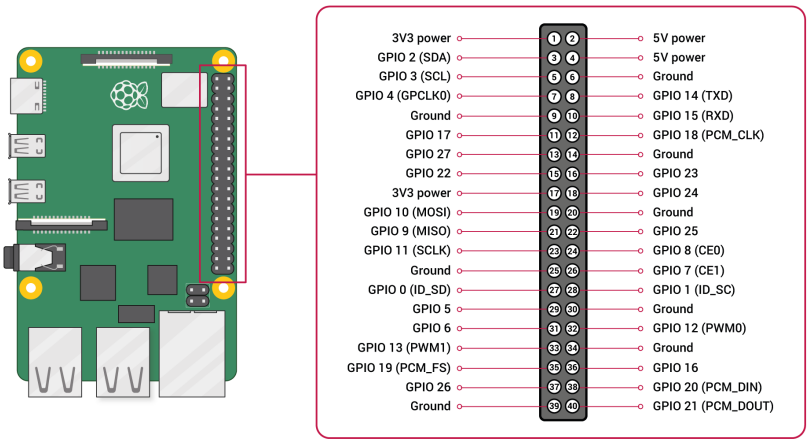
Det första problemet var att biblioteket `Rpi.GPIO` inte kunde hitta GPIO pinnarna på vår RPi 5. Detta beror på att de har förändrat chip-arkitekturen på RPi 5 där GPIO-pinnarna nu är anslutna till det nya RP1 southbridge-chippet. Eftersom RPi 5 lanserades för mindre än ett år sedan, var det en utmaning att hitta mjukvara som kunde ersätta `Rpi.GPIO`. Efter att ha sökt på olika RPi-forum hittades Python-biblioteket `gpiozero`. Detta bibliotek är en förenklad version av `Rpi.GPIO` och inkluderar flera färdiga funktioner för att hantera till exempel avståndssensorer samt styrning av servon och motorer [31].

Både motorn och servot styrs av PWM-signaler vilket är en effektiv teknik för att

kontrollera hastighet och position med hög precision. PWM, eller pulsbreddsmodulering, innebär att en elektrisk puls genereras med en variabel bredd, det vill säga att bredden av varje puls kan justeras. Genom att variera pulsbredden kan man påverka mängden energi som levereras till en motor eller servo och därmed påverka motorns hastighet samt servots position. Först användes *PWMOutputDevice* från `gpiozero` biblioteket för att skicka PWM signaler till servot och motorn. Under initiala tester av servot observerades markant jitter, vilket resulterade i oönskade rörelser och osäker prestanda. För att undersöka detta problem ytterliggare skickades PWM-signalen in i ett oscilloskop. Genom visualisering av signalen på oscilloskopet kunde fluktuationer i pulsbredden tydligt ses. Läser man i dokumentationen hos `gpiozero` står det noterat

”To reduce servo jitter, use the pigpio pin driver” [32].

Men på grund av den nya chip-arkitekturen på RPi 5 är inte pigpio längre tillgänglig. Problemet med jittereffekten grundar sig i att `gpiozero` använder mjukvarubaserad PWM generering. Mjukvaru-PWM är särskilt känslig för systemets belastning och kan påverkas av andra processer som körs parallellt. Eftersom mjukvaru-PWM inte hanteras av dedikerad hårdvara, måste den istället konkurrera om processorns tid och resurser, vilket kan leda till en jittrig PWM-signal. Ser man på GPIO-layouten för Raspberry Pi 5, som presenteras i Figur 4.3, framgår det att enheten är utrustad med hårdvaru-PWM-utgångar på GPIO-pinnarna 12 och 13. Dessa pinnar är specifikt designade för att hantera PWM-signaler med högre precision och stabilitet jämfört med mjukvarubaserade PWM-lösningar.



Figur 4.3: Pin layout för RPi 5 [33].

Nästa problem som uppstod var att det än så länge inte finns någon officiell mjukvara tillgänglig för att hantera dessa PWM-pinnar. Efter att ha utforskat flertalet Raspberry Pi-forum hittades `rpi-hardware-pwm`, ett Python bibliotek skapat av Cameron Davidson-Pilon, designat specifikt för att hantera hårdvaru-PWM-pinnarna på RPi 5 [34]. Men för att kunna använda biblioteket krävs vissa systemanpassningar, inklusive ändringar i firmware för att säkerställa att rätt kernelversion används. Kerneln är den del av operativsystemet som hanterar systemresurser som CPU,

minne och drivrutiner för in- och utenheter. Den fungerar som en bro mellan hårdvaran och mjukvaran, vilket gör den avgörande för att datorns program ska kunna kommunicera effektivt med dess fysiska komponenter, i detta fall PWM-pinnarna. På ett RPi forum beskrev användare ett problem med inkompatibla kernelversioner när de använde `rpi-hardware-pwm` som de framgångsrikt löste genom att manuellt installera rätt kernel version [35].

Utöver rätt kernel version skriver Davidson-Pilon att innan man kan använda `rpi-hardware-pwm` krävs en specifik konfigurationsändring i systemfilen för Raspberry pi. Genom att lägga till raden

dtoverlay=pwm-2chan,pin=12,func=4,pin2=13,func2=4

i systemfilen aktiverar man de nödvändiga PWM-kanalerna på enheten samt tvåkanals PWM-styrning. Med tvåkanals PWM-styrning kan två olika enheter, i detta fall en motor och ett servo, kontrolleras oberoende av varandra med olika signaler från samma mikrokontroller.

Så i slutändan så användes `rpi-hardware-pwm` för styrning av motor och servo kombinerat med `DistanceSensor()` funktionen i `gpiozero` biblioteket för att effektivt läsa värden ifrån avståndssensorn. För att kunna hantera RPi trådlöst, det vill säga utan att behöva koppla in skärm och tangentbort, användes en SSH-nyckel. Denna metod gör det möjligt att fjärransluta till enheten över ett nätverk, vilket bland annat möjliggör att exekvera kod trådlöst från en extern dator.

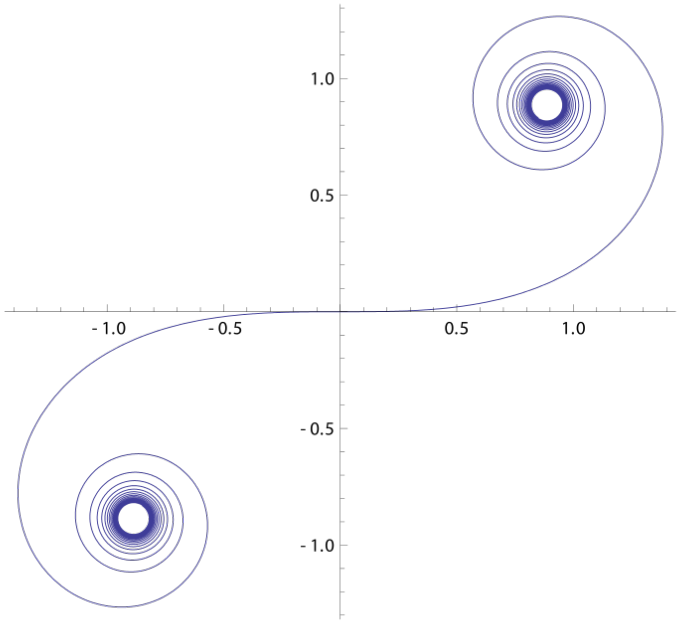
4.2 Fysisk implementering av vägföljning

Vid en fysisk implementering av vägföljning används samma algoritm som beskrivs i Avsnitt 3.1. Istället för att tillämpa algoritmen i simuleringsmjukvaran Carla, används den på ett direkt flöde av bilder från en kamera monterad i fronten på dollyn. Kameran, en DFRobot FIT0729, drivs och överför information via en USB-kabel till vår mikrokontroller [36]. Med en kompakt storlek på 38x38 mm och en upplösning på 8 megapixlar är den lätt att montera på dollyn och kraftfull nog för det avsedda användningsområdet.

Huvudtanken med den fysiska implementeringen är att skapa en referenslinje i mitten av bildskärmen, likt i simuleringen. Därefter kan felet mellan dollyns aktuella position och den linje som ritas av algoritmen mätas. Även här kommer en regulator att använda detta fel för att justera styrvinkeln och korrigera felet i vårt system.

För att få ett så robust system som möjligt kräver algoritmen att den utvalda körbanan ska efterlikna simulationen till viss grad. Detta betyder framförallt att väglinjerna är i samma färg som i simuleringen, det vill säga vita väglinjer. Detta går att kalibrera i algoritmen men efter vissa fysiska tester var vita väglinjer det optimala valet på de flesta körytorna. Detta kan läsas mer av i Avsnitt 3.1 där det beskrivs hur färgförhållanderna spelar roll i algoritmen och hur man kan lösa det. En annan parameter som kommer påverka robustheten gravt är höjden på den aktuella

kameran. Då det behövs ett visst synfält för att kunna fånga in båda väglinjerna i kameran, detta gäller framför allt i kurvor. Då dollyn svänger kan felet mot väglinjerna bli för stor och därav kan linjerna hamna utanför dollyns synfält.
Eftersom många motorvägar är uppbyggda med en så kallad övergångskurva (track transition curve) som ger en mjuk övergång mellan kurvor och raksträckor eller andra kurvor, det är för att den ska vara uppbyggd efter Eulers spiral eller en klotoid [37]. Övergångskurvan gör då att en kurva inte startar skarpt utan mjukt övergår från raksträcka till kurva, se Figur 4.4. Eftersom kurvorna då inte ändras för drastiskt ger det även en mindra styrvinkel och därav kan kameran se linjerna längre i samma bild. Då kan slutsatsen dras att ett allt för stort synfält inte behövs för att följa en vanlig körbana.



Figur 4.4: Bild på klotoid [38].

Med övergångskurvan i tanke kan en preliminär höjd sättas på kameran då en uppfattning om synfältet som behövs finns. Genom att sätta en höjd där båda linjerna tydligt är innanför synfältet av kameran. I Figur 4.5 kan en kamerahöjd på cirka 10 cm ses. Här kan avtydas att linjerna syns och kan avläsas av algoritmen. Då det även finns krav från hårdvarugruppen om vissa specifikationer på sensorerna, som till exempel höjden, så togs beslutet att detta var en bra höjd för att testa på dollyn.

Figur 4.5: Bild av dollyns synfält vid 10 cm.

4.3 Avståndshållning

Vid körning av dollyn kommer ena dollyn att behöva hålla ett fast avstånd till dollyn framför. Detta kommer att göras med hjälp en distansbaserad avståndshållare där ett avstånd kommer mätas för att sedan basera hastigheten efter hur långt bort kontra nära dollyerna är till varandra. Då en distansbaserad avståndshållare mäter avståndet kontinuerligt under färd kommer den även att kunna detektera eventuella hinder i körbanan och undvika dem.

4.3.1 Ultraljudssensor

Ultraljudssensorn fungerar kort genom att den skickar en utlösningssignal(trigger) och tar emot en eko signal(echo) för att sedan skicka tillbaka tiden det tar för ekot att träffa sensor till styrdatorn. Med denna informationen kan man få ett pricksäkert sätt bestämma det valda avståndet.

Trots detta gjordes valet att Ultraljudsensor ska användas för att på ett så enkelt och säkert sätt som möjligt skapa en jämn distansmätning. Valet av ultraljudssensor nalkas ner i två olika varianter Hc-sr04 och PING))) Ultrasonic Distance Sensor(PPING)))") [39][40].

Tabell 4.2: Jämförelse av Ultraljudssensorer.

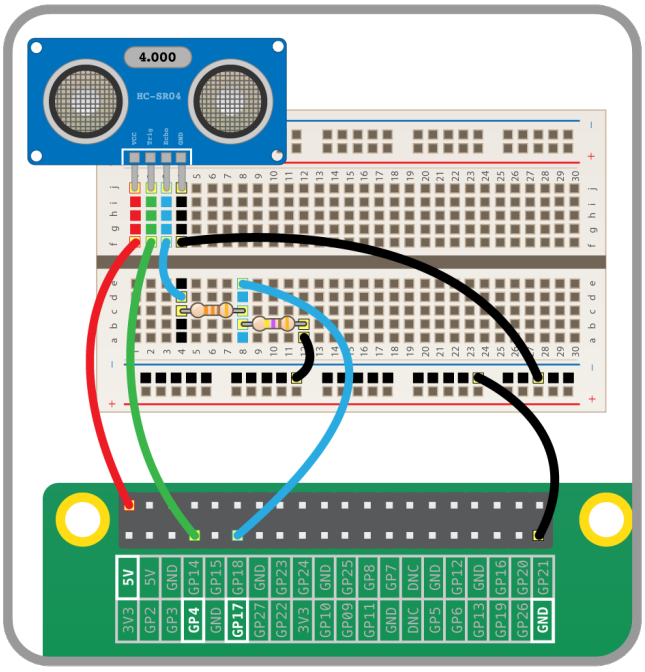
Sensor	PING)))	Hc-sr04
Distans max	300cm	400cm
Distans min	2cm	2cm
Ungefärligt pris	340 kr	34 kr
I/O pins	3	4

Utifrån Tabell 4.2 kan sensorerna jämföras, och det är tydligt att det inte finns några betydande skillnader i prestanda. HC-SR04 har visserligen längre räckvidd, men eftersom projektet inte kräver avstånd över 200 cm är detta inte en relevant faktor.

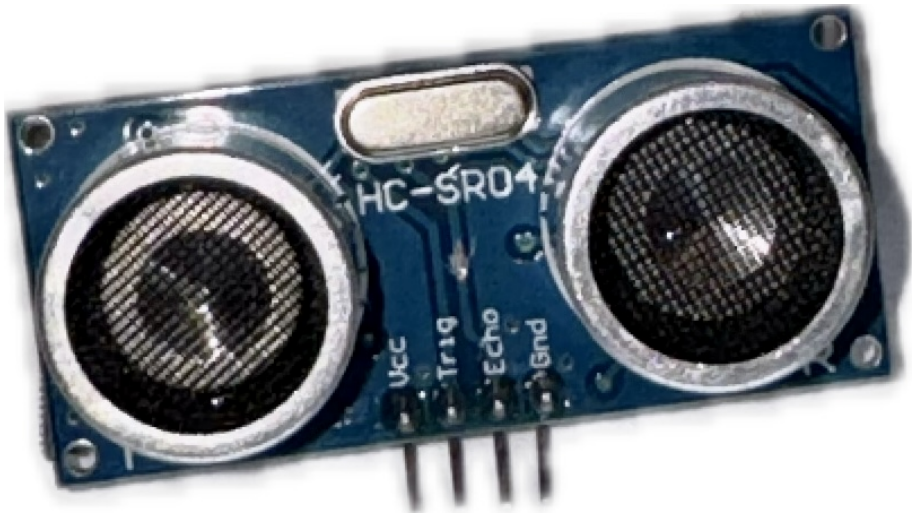
Dock är prisskillnaden markant lägre för HC-SR04, vilket gör den till ett attraktivt val.

En annan viktig parameter att beakta är att PING))) använder 3 I/O-pinnar, vilket kan skapa problem vid användning med Raspberry Pi 5 [27]. Raspberry Pi 5 använder vanliga GPIO-pinnar för att skicka 3,3 volt signaler, precis som andra mikrokontrollers. Problemet uppstår då båda dessa sensorer kräver 5 volts signaler, vilket kan skada GPIO-pinnarna eftersom de inte kan ta emot signaler starkare än 3,3 volt. Eftersom PING)))-sensorn använder samma pinne för att skicka utlösningssignalen och eko-signalen kan spänningsnivån inte sänkas individuellt, vilket kan göras om två olika pinnar används.

Genom en enkel koppling kan man dock spänningsdela signalen (se Figur 4.6). Genom att använda en 470 ohm och en 330 ohm resistor kan spänningen sänkas till en nivå som eliminerar risken för skador på mikrokontrollern. Detta gör HC-SR04 till det föredragna valet för projektet, med tanke på dess prestanda, pris och kompatibilitet med Raspberry Pi 5.



Figur 4.6: Extern koppling för HC-SR04 ultraljudssensor [41].



Figur 4.7: Bild på Hc-sr04 ultraljudssensor.

Ett annat sätt än den externa kopplingen, se Figur 4.6, att få ECHO-signalen från ultraljuds sensorn i Figur 4.7 att bli 3.3V, är att modifiera sensorn direkt. Detta kan göras med hjälp av resistorer som kopplas parallellt som får ner spänningen i ECHO-signalen från 5V till 3.3V. Detta sätt är att föredra då detta tar mindre plats då både användadet av Dupont kablar och kretskort kan tas bort. Speciellt när det redan är platsbegränsat på de två dollys som byggts.

Idén kom från Autodesk Instructables [42] som hade gjort samma metod innan och fått det att fungera. Metoden, enligt anjoschu, går ut på att först kapa den initiala ECHO-signalen (5V) genom att bryta den krets som går från ECHO-pinnen till en integrerad krets på baksidan av ultraljudssensorn. Därefter skapas en spänningsdelare med hjälp av två stycken parallellkopplade resistorer som gör om ECHO-signalen från 5V till 3.3V, skriver anjoschu. Utifrån de komponenter, i form av resistorer, som finns i CASE används 4.7kΩ och 2.2kΩ resistorer. Detta innebär att följande ekvation beskrivs engligt

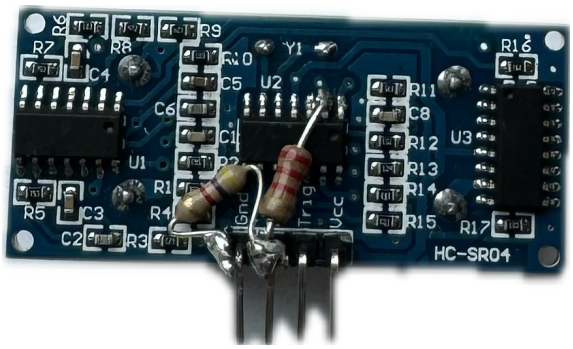
$$\Omega = \frac{R1}{R1 + R2} \Rightarrow \frac{4.7k}{4.7k + 2.2k} \approx 0.681. \tag{4.1}$$

Detta ger vidare ekvationen

$$V = 5 \cdot 0.681 \approx 3.4V. \tag{4.2}$$

Med facit i hand som visar att resistorerna uppfyller de krav som satts upp kan resistorerna lödas på. 2.2kΩ resistorn börjas lödas på från ECHO-porten på den integrerade kretsen på baksidan som sedan löds fast på ECHO-pinnen, skriver anjoschu. Därefter löds 4.7kΩ resistorn fast på ECHO-pinnen samt på GROUND-pinnen vilket fastställer en sluten krets där ECHO-signalen som ultraljudssensorn skickar ut har spänningen 3.4V, skriver anjoschu. Eftersom att den nya ECHO-signalen har spänningen 3.4V istället för 3.3V dubbelkollades detta innan ultraljudssensorn kopplades in. Enligt [43] står där i kapitel 6 att den maximala input-signalen max får vara 4.1V vilket innebär att den nya ECHO-signalen inte kommer att skada Raspberry Pi:n.

Innan den nu modifierade ultraljudssensorn kopplas in i Raspberry Pi:n kopplas den istället in i en Arduino Mega samt ett oscilloskop där signalens styrka och form kan visualiseras. Anledningen till varför ultraljudssensorn först kopplas in i en Arduino är för att dess digitala pinnar klarar av, och förväntar sig, en ECHO-signal på 5V. Den nya spänningsdelade kretsen består därför endast av två olika resistorer, se Figur 4.8



Figur 4.8: Modifierad ultraljudssensor (3.3V).

4.4 Reglering av fysisk vägföljning och avståndshållning

För att säkerställa att de tidigare nämnda funktionerna i Avsnitt 4.2 och 4.3 är användbara och effektiva i sina respektive uppgifter, krävs det en väl dimensionerad regulator som kan hantera både hastighet och styrvinkel. Eftersom dessa är två separata funktioner, krävs det att regulatorerna anpassas för varje specifikt syfte, men det är möjligt att använda samma typ av regulator för att styra båda funktionerna i systemet. I detta sammanhang är en Proportional Integral Derivative (PID) regulator en passande lösning för att modellera både hastighet och styrvinkel, vilket gör att systemet kan anpassa sig till olika situationer och krav.

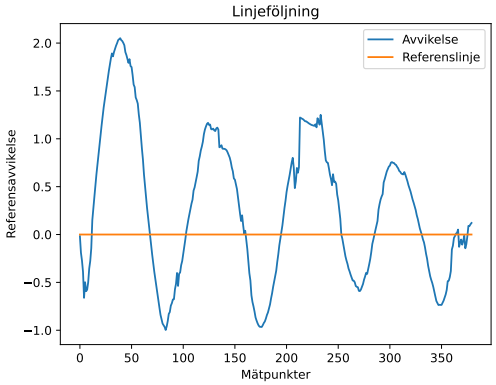
För att dimensionera dessa två regulatorer användes Ziegler-Nichols metod, som är välkänd och intuitiv för att reglera olika typer av system [44]. Metoden bygger på ett värde, K_u eller $K_{\max}$, som representerar det proportionella förstärkningsvärdet (K_p) vid vilket systemet oscillerar stabilt när integral- och derivatevedelarna (K_i och K_d) är noll. Genom att bestämma detta K_u-värde och oscillationstiden T_u kan de övriga regulatorparametrarna beräknas. För detta ändamål valdes ett system som intuitivt kan tillämpas på vårt specifika system.

Den resulterande regulatorn kan sedan finjusteras med hjälp av Tabell 4.3. Notera att för att uppnå en korrekt dimensionering krävs fysiska tester för att få fram de nödvändiga värdena, såsom K_u och T_u. Dessa värden erhålls genom att observera systemets respons på en stegvis inmatning, vilket ger en stabil oscillation som kan analyseras för att bestämma de optimala regulatorparametrarna.

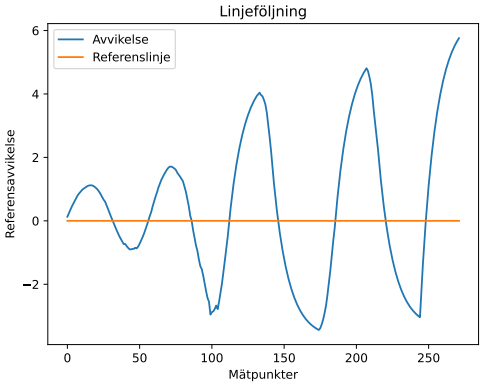
Tabell 4.3: Ziegler-Nichols Dimensionerings Parametrar.

K_u	T_u	K_p	T_i	K_i	T_d	K_d
K_u	T_u	$0.6 \cdot K_u$	$0.5 \cdot T_u$	$\frac{1.2 \cdot K_u}{T_u}$	$0.125 \cdot T_u$	$0.075 \cdot K_u \cdot T_u$

För att visa hur denna metod appliceras i praktiken, presenteras i Figur 4.9 och 4.10 oscillationen vid olika värden på K_p i fallet av linjeföljningen. Dessa experimentella resultat ger värdefull insikt i systemets beteende och möjliggör finjustering av regulatorparametrarna för att uppnå önskad prestanda.

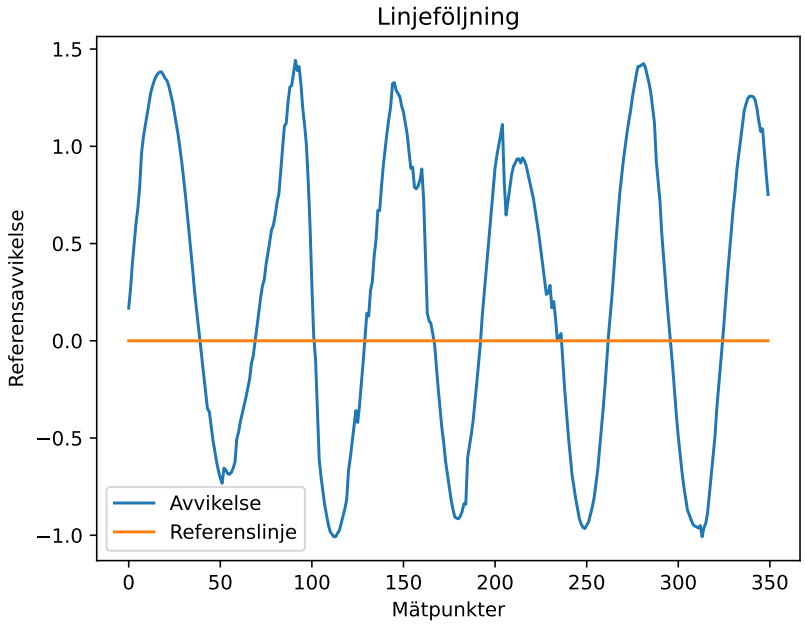


Figur 4.9: Oscillering vid $K_p = 0.03$. (cm)



Figur 4.10: Oscillering vid $K_p = 0.08$. (cm)

Vid utföring av Ziegler Nichols metod vill som sagt ett värde på K_p hittas så att en stabil oscillering kan uppnås, detta kan ses i Figur 4.11 som till skillnad från Figur 4.9 och 4.10 varken ökar eller minskar med tiden.



Figur 4.11: Oscillering vid $K_p = 0.0425$. (cm)

När ett värde på K_u, det vill säga $K_u = K_p = 0.0425$ när $K_i = K_d = 0$, tagits fram kan även T_u beräknas med hjälp av Figur 4.11. Genom fysiska tester kan tiden mellan de olika mätpunkterna tas fram, detta ger en periodtid för ocsillitionerna på cirka $T_u = 2.5$ sekunder då varje mätpunkt i grafen är lika med 0.04 sekunder. Med dessa värden kan hela dimensioneringen utföras med hjälp av Tabell 4.3. De slutgiltiga värdena på vägföljningsregulatorn kan ses i Tabell 4.4.

Tabell 4.4: Värden på PID parametrar.

K_u	T_u	K_p	T_i	K_i	T_d	K_d
0.0425	2.5	0.0255	2.4	0.02	0.35	0.008

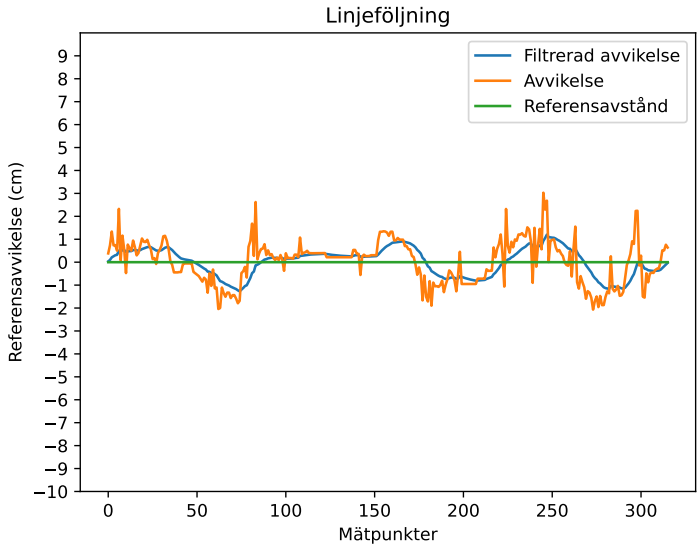
5
Resultat

I detta kapitel diskuteras resultat som erhållits utifrån projektets målsättning. Detta innefattar vägföljning, avståndshållning samt dockning.

5.1 Fysisk vägföljning

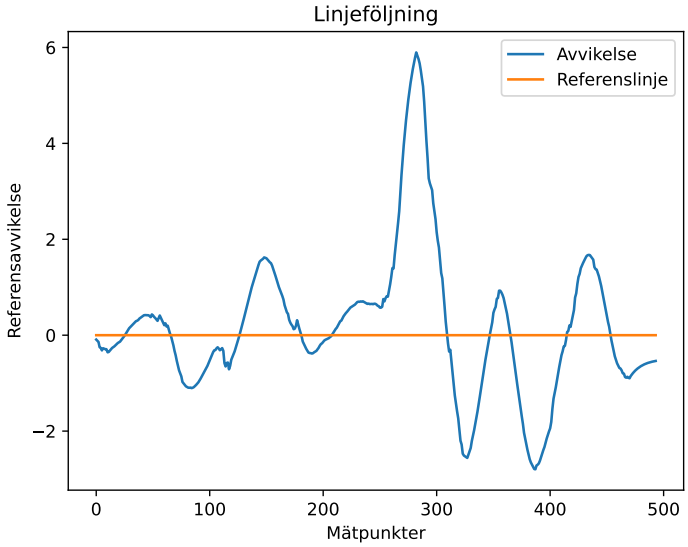
Under de fysiska testerna för vägföljningsalgoritmen konstruerades en körbana med hjälp av vit eltejp för att möjliggöra en rättvis utvärdering. Det första testet involverade följning av en raksträcka som övergick till en skarp kurva. Denna testsekvens genomfördes innan en väl dimensionerad regulator eller ett Kalman-filter hade implementerats i systemet, det vill säga innan tillämpningen av Ziegler-Nichols-metoden.

För att utnyttja sensordata från kameran krävdes en grundlig analys av datan för att identifiera potentiella system- och mätfel, vilket var avgörande för att kunna korrigera och reglera signalen korrekt. När rimliga mätvärden kunde utläsas från systemet implementerades ett Kalmanfilter som använder både aktuella och tidigare mätvärden som in-signal för att förutse en korrekt avvikelse. Figur 5.1 visar tydligt hur filtret effektivt rensar bort oväntade in-signalstörningar från kameran och smidigt jämnar ut mätvärdet, vilket resulterar i en utmärkt filtrering av sensordatan.



Figur 5.1: Graf över över filtrerad signal kontra in-signal från kameran.

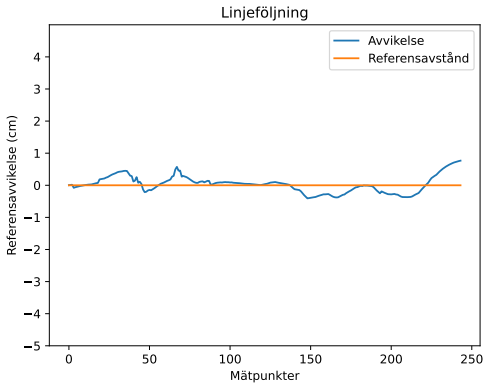
Med ett Kalmanfilter implementerat möjliggjordes en noggrann körning på körbanan. Trots att ett dimensionerat reglersystem ännu inte hade implementerats, kunde testningen utföras. I Figur 5.2 visas avvikelsen från systemet under körningen.



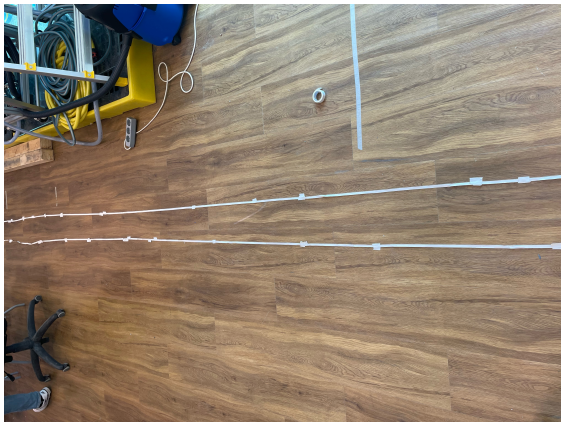
Figur 5.2: Graf över referensavviskelse i cm kontra mätpunkter på kurvig bana, odimensionerad PID.

Grafen kan beskrivas såsom körbanas utseende, där det började med en raksträcka som sedan går in i en skarp kurva där vi kan se den stora toppen i grafen. Som kan tydas är det relativt stora ocillationer som nuddar upp mot ±2 cm och framför allt efter kurvan har systemet stora ocsillationer. Detta beror som sagt på att PID regulatorn som används i detta fallet inte har blivit dimensionerad på ett korrekt sätt. Men den ger en tydlig bild på att algoritmen kan implementeras på den fysiska dollyn och kan följa en körbana.

Efter att reglersystemet dimensionerades med Ziegler-Nichols-metoden genomfördes en jämförande testning på både rak och kurvig vägsträcka, vilket återspeglas i Figurerna 5.3 och 5.5. I detta testet kördes dollyn i 28.1 cm/s vilket är den maximala hastigheten den kan köra. I Figur 5.3 framgår det tydligt att amplituden hos svängningarna har reducerats avsevärt, från cirka ±2 cm till ungefär ±0.5 cm. Denna minskning motsvarar endast cirka 2% av den totala vägbredden, vilket klart och tydligt indikerar en förbättring i systemets stabilitet vid körning på rak väg. Trots eventuella ojämnheter i vägbanan, som kan observeras i Figur 5.4, upprätthåller regleralgoritmen en stabil körning utan avsevärda oscillerande fenomen.

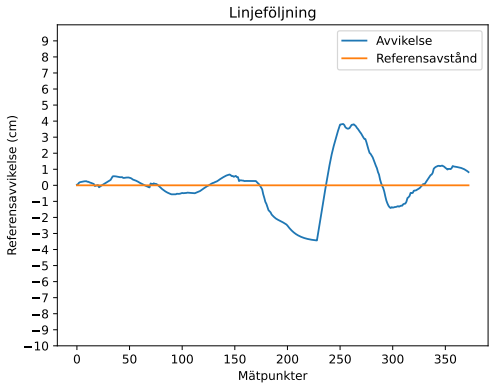


Figur 5.3: Graf över referensavviskelse i cm kontra mätpunkter på rak väg, dimensionerad PID.

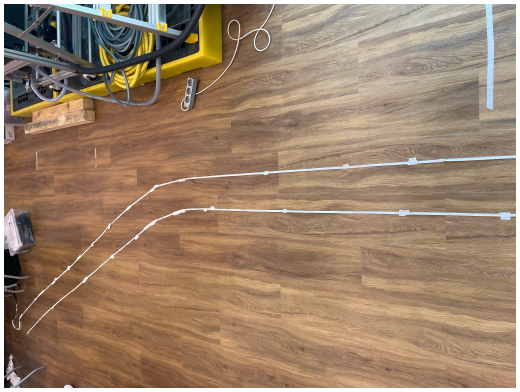


Figur 5.4: Bild över vägen som kördes i Figur 5.3.

I Figur 5.5 presenteras istället påverkan på reglersystemet vid körning i en kurva. Grafen illustrerar hur dollyn bibehåller stabilitet under raksträckan innan kurvan, följt av en markant negativ förändring när kurvan börjar. Detta resulterar i en snabb respons från reglersystemet med en kraftig utslagning av regulatorn. Denna reaktion leder till ett tillfälligt överutslag, som sedan snabbt korrigeras, och dollyn fortsätter sedan att följa vägen fram till slutet av vägen.



Figur 5.5: Graf över referensavvikelse i cm kontra mätpunkter på väg med kurva, dimensionerad PID.



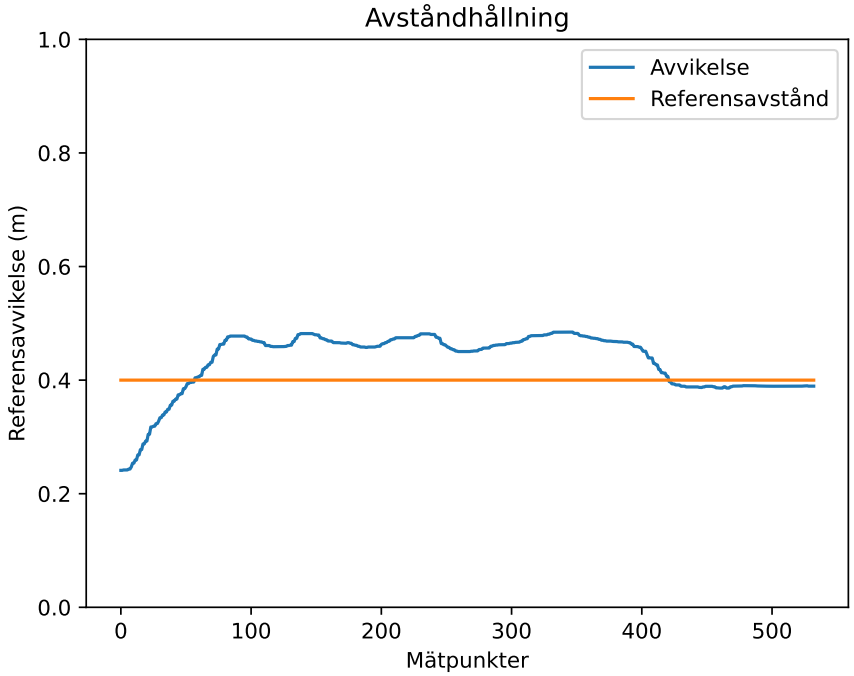
Figur 5.6: Bild över vägen som kördes i Figur 5.5.

Det är tydligt att reglersystemet uppvisar en förmåga att hantera snabba förändringar i omgivningen, vilket är avgörande för att säkerställa säker och smidig körning genom kurvor. Dock kan det vara värt att ytterligare analysera och eventuellt finjustera systemets respons för att minska eventuella överutslag och förbättra övergången mellan raksträckor och kurvor.

Genom dessa tester var det möjligt att utforska dollyns förmåga att hantera skarpa kurvor på ett fullskaligt sätt. I Figur 5.6 illustreras en kurva som mätts till 40 grader från den raka startsträckan. Vid fortsatta försök noterades att dollyn kunde navigera genom en kurva med en lutning på upp till 45 grader utan att avvika från vägbanan. Detta representerar den maximala lutning som har testats och visar på dollyns förmåga att hantera utmanande kurvor.

5.2 Fysisk avståndshållning

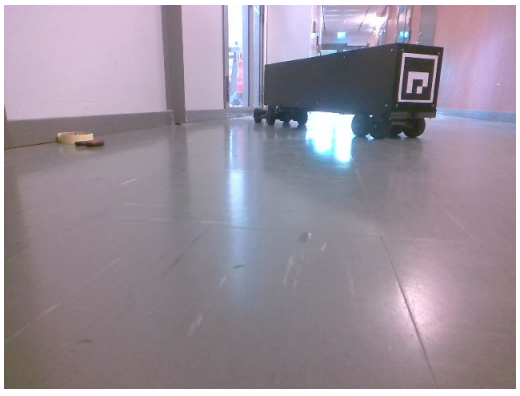
I den fysiska testningen av avståndshållning initierades ett objekt cirka 0,2 meter från dollyn. Dollyn accelererade med målet att kunna följa objektet med ett konstant avstånd för att till slut stanna på ett önskat avstånd (0,4 m) från objektet. Referensavvikelsen, som är skillnaden mellan det uppmätta avståndet och det önskade referensavståndet, illustreras i Figur 5.7. Denna avvikelse visar ett relativt konstant kvarstående fel på ca +0.1 m under körningen. Dock noteras att det önskade avståndet uppnås när dollyn slutligen stannar framför objektet.



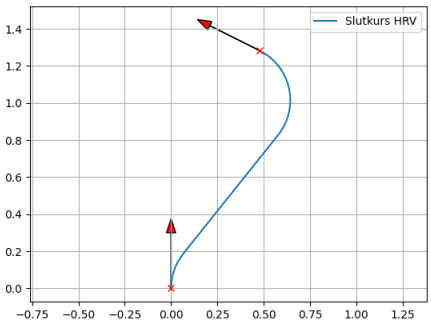
Figur 5.7: Graf över referensavvikelse i meter med P-regulator.

5.3 Dubins path

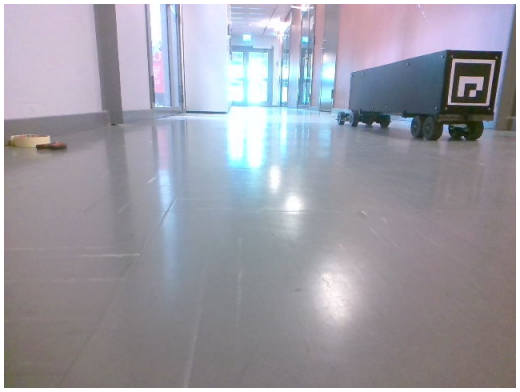
Simuleringar av Dubins path gjordes genom att sätta en ArUcomarkör på baksidan av ett av ekipagen. Sedan togs en bild med kameran på andra ekipaget. Två bilder togs som kan ses i Figur 5.8 och Figur 5.10 från olika position och vinkel för att se om en planerad vägbana kunde generaras med hjälp av Dubins path algoritmen. Från Figur 5.9 och Figur 5.11 kan man se den planerade vägbanan.



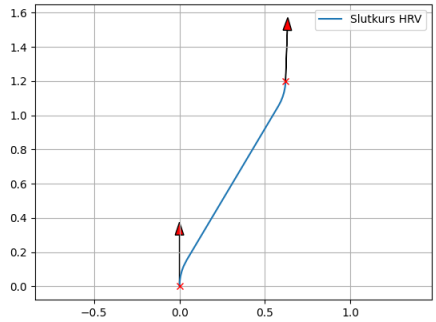
Figur 5.8: Bild 1 tagen från dollyn.



Figur 5.9: Genererad körbana av Dubins path utifrån Figur 5.8.



Figur 5.10: Bild 2 tagen från dollyn.



Figur 5.11: Genererad körbana av Dubins path utifrån Figur 5.10.

6

Diskussion

I detta kapitel diskuteras resultaten samt utmaningar som uppstått under projektets genomförande.

6.1 Vägföljning i Carla

Styrvinkeln beräknas genom att välja en punkt på referenslinjen, se Avsnitt 3.1.2. Att välja en punkt på referenslinjen som är längre ner i bilden hade också kunnat ge ett lägre utslag på styrvinkeln. Detta då eftersom avvikelsen mellan den valda punkten och kamerans mittaxel hade varit mindre än om punkten är vald längre upp i bilden och på referenslinjen. På så sätt kan denna förändring också resultera i en mindre vinkel på samma sätt som en förändring av skalfaktorn k. Däremot eftersom k är lättare att förändra och är mer intuitivt anses det vara ett bättre tillvägagångssätt.

Ett problemområde vid användning av Carla var att skapa en simulationsmiljö som emulerar den fysiska testmiljön. Kraven var att väglinjerna skulle vara vita och heldragna, vilket överraskande nog var en bristvara inom de färdiga stadskartorna i Carla. Idealt eftersträvas också en isolerad vägbana med heldragna vita linjer då det underlättar evalueringen av vägföljningsalgoritmen. Detta var något som tillslut identifierades, men antalet av dessa typer av miljöer anpassade efter projektets ändamål var extremt sällsynta, vilket resulterade i att den virtuella brandbilen bara simulerades i en specifik miljö.

En idé var att skapa egna kartor och testmiljöer med programmet RoadRunner som är direkt kompatibelt med Carla [45]. En konstruerad simulationsmiljö i RoadRunner kan importeras in och användas i Carla och kan därmed lägga en grund för en perfekt skräddarsydd testmiljö för ändamålet. Det hade också möjliggjort att kunna skapa mer varierande simuleringskontext och att mer i djupet kunna testa robustheten av vägföljningsalgoritmen i varierande underlag och ljus. Att kunna skapa en mindre bana hade också resulterat i mindre belastning på grafikkort och processor, som i Carla är essentiella hårdvarukomponenter för att effektivt kunna köra programmet. Problematiken låg i att det krävdes en licens som aldrig godkändes.

6.2 Analys av fysisk vägföljning

Trots de positiva resultaten från de varierade testerna av den fysiska vägföljningen finns det utrymme för diskussion och förbättringar. Vid implementeringen av Kalmanfiltret observerades en tydlig förbättring av signalen, där de flesta opålitliga insignalstörningar filtrerades bort. Dock kan man se i Figur 5.1 att den filtrerade signalen alltid ligger något efter den ofiltrerade signalen. Detta resulterar i ett litet fördröjning innan systemet kan reagera på eventuella fel. Den främsta orsaken till fördröjningen är att de initiala parametrarna, såsom mät- och systemfel, inte är korrekt initierade. Därför kommer systemet alltid att reagera trögt på förändringar i vägbanan, vilket kan påverka körningen negativt vid högre hastigheter.

Ett annat problem som identifierades under testerna av vägföljningen var att kameran inte alltid kunde detektera de markerade väglinjerna. Speciellt i kurvor och vid starttillfällen hade kameran svårigheter att hitta väglinjerna, vilket resulterade i att systemet antingen inte startade eller körde utanför vägbanan. Genom lättare tester där dollyn startade från olika höjder blev det lätt att se att detta berodde på att kamerans höjd var för låg. Detta kan kopplas till att när höjden på kameran konfigurerades fanns det inte samma krav på svängradie som det önskas vid nuvarande utvärdering. Genom att höja kamerans höjd skulle synfältet för dollyn öka, vilket möjliggör en bättre detektering av skarpa kurvor och förbättrad prestanda.

Vid körning på en rak väg, såsom illustrerat i Figur 5.3, är det tydligt att trots en väldimensionerad regulator fortsätter det att förekomma oscillationer i systemet. Denna observation kan härledas till fördröjningen i Kalmanfiltret och kameraströmmen. Eftersom alla fel når regulatorn med en viss fördröjning, försenas även systemets reaktion, vilket i sin tur resulterar i oscillationer. Detta fenomen betonar vikten av att inte bara dimensionera reglersystemet korrekt, utan också att minimera fördröjningar i sensordata och feedbackloopar för att uppnå önskad stabilitet och respons. Men det påverkas inte bara utav fördröjningarna, då väglinjerna var långt ifrån optimala påverkade de systemet avsevärt. Då de kunde variera i både vägbrädd och ibland bestå av mindre svängningar så hade dollyn svårare att hålla sig stabilt.

6.3 Simulering av avståndshållning

Från Figur 3.13 kan uttydas att det inte finns någon översväng av positionen mellan fordonen. Med det menas att följebilen aldrig befinner sig närmare än 20 enheter till ledarbilen vilket kan vara viktigt ut säkerhetssynpunkt och komfortsynpunkt för eventuella förare eller gods. Avståndet mellan ekipagen stabiliseras även kring 20 enheter vilket innebär att inget statiskt reglerfel finns i avståndshållningen. En mjuk reglering utan översvängar av avståndet mellan fordonen är fördelaktigt då detta sliter mindre på fordonet vilket ökar hållbarheten. Det kan dock innebära en viss fara då en oväntad situation kan uppkomma där en bil eller ett djur kan hamna

mellan fordonen vilket kräver ett snabbt agerande av styrningen vilket kan vara ett problem för systemet som det ser ut i nuläget. Ett korrekt avstånd mellan ekipagen är också viktigt då detta kan användas för att hålla säkerhetsavstånd utefter hastighet på fordonet

Modellen i Figur 3.11 är förenklad i det avseendet att den endast tar hänsyn till avståndshållning i en dimension, det vill säga i en riktning. Detta gör simuleringen enklare och också systemet som skall regleras mindre komplext. I systemet modelleras också motståndet som endast beroende på vind och friktion. Detta innebär att modellen inte tar hänsyn till om systemet skulle uppleva motstånd i form av att dollyn kör i en uppför- eller nedförsbacke. Är tanken att utveckla en avståndshållning för ett fordon som endast ska färdas åt ett håll på plant underlag kan det vara en bra approximation men är tanken att utveckla en avståndshållning för ett verkligt system kan det vara bra att även ta hänsyn till följning i sidled samt körning på ett lutande plan.

6.4 Analys av fysisk avståndshållning

Som noterat i Figur 5.2 förekommer det en kvarstående avvikelse ifrån önskat avstånd under körningen. Detta beror förmodligen på att en P-regulator endast tar hänsyn till det aktuella värdet av avvikelsen. Den saknar förmågan att ackumulera tidigare fel (som en I-komponent gör) eller att förutse framtida fel (som en D-komponent gör). För att kunna hålla ett konstant avstånd till dollyn framför hade därför en PID-regulator varit mer lämplig. Men på grund av tidsbrist i slutet av projektet, kunde en PID-regulator inte dimensioneras för den fysiska modellen. Därför användes en enkel P-regulator som lösning.

Trots att målet med konstant avståndshållning inte har uppfyllts med hjälp av P-regulatorn, kan dollyn ändå följa det framförvarande fordonet med ett avstånd som är godtagbart för de flesta trafiksituationer. Det är även värt att nämna att trots att dollyn ofta håller ett avstånd som är något längre än det önskade, är detta en betydligt säkrare situation än det motsatta scenariot. Om systemet istället hade tenderat att underskrida det önskade avståndet, hade det medfört en betydligt högre risk för kollision. Regulatorn garanterar även att dollyn alltid kommer att stanna 0.4 meter från ett objekt framför vilket visas i Figur 5.7.

Till denna regulator implementerades inte något Kalmanfilter. Den primära anledningen till detta var att avvikelsen, som visas i Figur 5.1, är betydligt mer brusig än avvikelsen i Figur 5.7. Men i mer oregelbundna situationer där sensorn registrerar felaktiga objekt eller misslyckas med att upptäcka något alls hade ett Kalmanfilter kunnat erbjuda signifikanta fördelar. Det hade ökat systemets robusthet genom att filtrera bort felaktiga signaler vilket är kritiskt när sensorn stöter på orelaterade objekt som inte ska påverka systemets beslut. Men på grund av tidsbrist bortprioriterades detta eftersom systemet i majoriteten av fallen fungerar väl utan detta och dimensioneringen av ett Kalmanfilter för linjeföljning ansågs vara mer kritiskt.

6.5 Autonom dockning med ArUco-markörer och Dubins path

Mätningarna utförda för att utvärdera positionsuppskattningens noggrannhet av ArUco-markörer visade på en imponerande precision. Horisontella mätningar uppvisade endast små avvikelser på ungefär 0.2 cm jämfört med kontrollmätningen, vilket indikerar en hög noggrannhet. Dessa avvikelser kan delvis förklaras av mätfel, särskilt då avståndet mättes med en linjal. Mätningarna i z-led (djup) visade något större mätfel med en avvikelse på 2 cm. För vinkelmätningarna observerades ett mätfel på endast 0.56 grader. Sammantaget visade resultaten en hög precision och tillförlitlighet vid identifiering av ArUco-markörerna, vilket är avgörande för att Dubins path-algoritmen ska fungera korrekt.

Simuleringen av Dubins path-algoritmen baserad på information från ArUco-markörerna, som åskådliggörs i Figur 5.8 till Figur 5.11, framhäver en lovande potential. Eftersom ingen fysisk implementation utfördes, begränsades möjligheterna till praktiska tester. Simuleringen fungerar i stället som en visualisering av algoritmens prestanda. Trots avsaknaden av direkta fysiska tester ger simuleringen viktig insikt i algoritmens beteende och dess förmåga att navigera baserat på de insamlade data från ArUco-markörerna. Denna visualisering utgör således ett första steg mot att bedöma algoritmens effektivitet och tillförlitlighet innan den eventuellt implementeras i en verklig miljö. Exempelvis kan det vara nödvändigt att finjustera parametrar såsom krökningen för att förbättra precisionen hos algoritmen. Detta beror på att många centrala delar av algoritmen är starkt beroende av just denna parameter. Genom att noggrant anpassa krökningsparametern kan man optimera algoritmens förmåga att generera mer exakta och realistiska vägbanor.

7

Vidareutveckling

I detta kapitel om vidareutveckling diskuteras framtida förbättringar och expansioner av de autonoma dollysystemen. Möjligheterna att utöka systemets rörelsefrihet samt integrera kommunikationstekniker, i kombination med implementeringen av Dubins path för förbättrade docknings- och koordinationsförmågor under körning, utforskas. Vidare undersöks implementeringen av ett globalt positioneringssystem för att stärka dollyernas navigationsförmåga på ett mer omfattande och tillförlitligt sätt. Genom dessa förbättringar strävas det efter att skapa en robustare och mer effektiv lösning för framtida arbeten, baserat på insikter som framkom under arbetets gång.

7.1 Simulering av avståndshållning

En möjlig utvidgning av systemet i Figur 3.11 skulle kunna innefatta att modellen får ytterligare en frihetsgrad för rörelse i sidled, utöver de befintliga framåt- och bakåtriktade rörelserna. Dessutom kan dynamiken för körning i lutning integreras i modellen. Detta skulle göra simuleringen mer representativ för verkliga scenarier, såsom körning i sluttningar och i kurvor, vilket är vanliga situationer på verkliga vägar.

För att undersöka hur bra systemet följer en referensbana och hur väl avståndet mellan olika enheter upprätthålls, kan en visualisering av en planerad rutt användas. Genom att plotta ledarfordonet och följefordonets rörelser kan vi studera både avståndet mellan fordonen och följarbilens förmåga att följa en referenslinje i sidled. Detta skulle ge läsaren en mer konkret uppfattning om systemets prestanda i avståndshållning och banföljning under realistiska förhållanden.

7.2 Kommunikation mellan dollys

För vidareutvecklingen av systemet är det avgörande att implementera kommunikation mellan dollyerna för att möjliggöra effektiv dockning och koordinerad körning när ekipagen är kopplade. För att uppnå realtidssynkronisering mellan ekipagen kan tekniker som Wi-Fi eller Bluetooth användas, där Raspberry Pi 5 redan har inbyggda moduler för detta.

7.3 Globalt positioneringsystem

I detta projekt har endast ett lokalt positioneringssystem med hjälp av en kamera undersökts. För att framgångsrikt navigera efter en planerad rutt skulle dock ett globalt positioneringssystem behövas. Genom att använda ett globalt positioneringssystem skulle dollyn kunna se hela banan som den ska manövrera, i kombination med det lokala systemet.

I början av projektet valdes det att ignorera möjligheten till ett globalt positioneringssystem eftersom det utgör en betydande teknisk utmaning och skulle kräva en betydande del av projektets resurser för att implementera. Emellertid har andra projekt, såsom kandidatgrupp EENX16-24-12 eller Inomhuspositioneringssystem, utforskat och utvecklat liknande positioneringssystem. Genom ett samarbete med dessa grupper skulle det vara möjligt att dra nytta av deras arbete och implementera ett globalt positioneringssystem i detta projekt. Med hjälp av ett sådant system skulle dollyn kunna planera sin rutt och navigera till valda positioner i utrymmet med hög precision och tillförlitlighet. Genom att integrera ett globalt positioneringssystem kan man möjliggöra en mer avancerad och effektiv autonom navigationsförmåga för dollyn.

7.4 Fysisk implementering av Dubins path

I vidareutvecklingen av det autonoma dockningssystemet kan nästa steg innefatta den fysiska implementeringen av Dubins path-algoritmen på dollyn. Genom att integrera algoritmen direkt på fordonet skulle man kunna utföra tester och utvärdera dess prestanda i verklig miljö. På så sätt kan eventuella utmaningar eller begränsningar som inte kan upptäckas i simuleringar identifieras och åtgärdas. Dessutom kan resultaten från dessa fysiska tester användas för att noggrant justera och optimera parametrar, vilket i sin tur kan förbättra algoritmens beteende och precision.

8

Sociala och Etiska Aspekter

Den tekniska utveckling som pågår i dagens samhälle har stort inflytande på hur omvärlden formas. Denna utveckling kan ge utslag i både positiva och negativa effekter. Den primära positiva effekten som eftersträvas är att höja människans levnadsstandard och öka tillgängligheten för tekniken som utvecklas. Ur ett mer negativt perspektiv blir tekniken mer avancerad och svårare att kontrollera för människan. Det kan bidra till att konsekvenserna blir stora då skaparen av tekniken inte längre har full kontroll över teknologins påverkan. I kapitlet sociala och etiska aspekter analyseras hur en autonom lastbil kan komma att påverka samhället.

8.1 Sociala aspekter

När utvecklingen av autonoma lastbilar kommer till stadiet att transporterna kan vara helt autonoma kommer samhället påverkas från flera håll. Förarlösa lastbilar kommer påverka den nuvarande marknaden som lastbilsförare befinner sig i. Jobbmöjligheterna kommer korrigeras om och bli mer mot ingenjörsinriktning och arbeten inom service för fordonen. Alla lastbilsförare kommer dock inte att förlora sina jobb [46]. Företag som testar autonoma lastbilar har för avsikt att hela tiden ha en förare i lastbilen. Det är för mycket som kan gå fel för att en förare inte ska sitta i lastbilen. Yrket som lastbilsförare kommer att genomgå förändringar i takt med utvecklingen. Framförallt kommer inte föraren köra lastbilen manuellt till lika stor grad, utan istället agera som en pilot genom att övervaka och se till så att den autonoma lastbilen framför sig korrekt.

I och med att lastbilarna blir mer autonoma så kan körningen planeras bättre. I en studie gjord av TuSimple blev resultat att den generella bränsleförbrukningen minskade med 11% vid autonom körning jämfört med manuell körning [47].

Nio av tio trafikolyckor uppstår pågrund av mänskliga fel och misstag [48]. De autonoma lastbilarna möjliggör att transporter och leveranser inte påverkas av den mänskliga faktorn. Dessutom kan de autonoma lastbilarna transportera sig dygnet runt då det inte finns någon förare som behöver stanna och vila, vilket gör att det blir mer effektiva leveranser. Det kan bidra till att minskad trängsel för andra trafikanter under dagen då större delen av transporterna kan ske under natten.

Tanken är att den autonoma lastbilen ska bestå av flera dollys som ska vara sammankopplade, vilket medför att längden på fordonet utökas. När lastbilarna blir längre

ökar parallellt riskerna för andra trafikanter i trafiken. Riskerna som medkommer är bland annat att både sikten och omkörningsmöjligheterna blir mer begränsade och till följd av detta blir omkörningstiden längre. Detta betyder dock inte att risken för kollision ökar, tvärtom, eftersom antalet lastbilar ute på vägarna minskar bidrar det till färre olyckor [49].

Det finns många tekniska utmaningar med autonoma fordon, men även säkerhetsfrågor och lagar som hindrar utvecklingen [50]. Vidare behöver den digitala infrastrukturen utvecklas för att implementera autonoma lastbilar på vägarna. Dessutom finns det andra säkerhetsaspekter, bland annat cybersäkerhet. Det är något som behöver ses över för att undvika eventuella attacker och skador som kan följa en cyberattack.

8.2 Etiska aspekter

Ur ett etiskt perspektiv så kommer säkerheten hos den autonoma lastbilen stå i centrum. När det är ett fordon som är förarlöst så blir det svårt att bestämma vem som har ansvar vid eventuell olycka. Vid första anblick tänker man att en autonom lastbil företräder säkerhet, men hur säker kan en autonom lastbil vara? Kan man få den 100% säker från olyckor? Hur säkert behöver fordonet vara för att det ska kunna framföras autonomt? Ja, det finns många frågor som måste bli besvarade innan man börjar testa autonoma lastbilar på allmänna vägar. Skulle en olycka inträffa, vem är det som bär ansvar? Är det tillverkaren av fordonet? Företaget som utvecklat systemet? Personen som äger fordonet? Personen som sitter i förarsätet?

En viktig aspekt är att det självkörande fordonet ska respektera mänskliga värdigheten och friheten att välja [51]. Skulle den autonoma lastbilen ställas inför ett dilemma där dödsfall är oundvikligt behöver fordonet kunna prioritera. Om scenariot är att antingen köra på ett äldre par eller krocka så att passagerarna i lastbilen omkommer, hur ska fordonet prioritera i ett sådant dilemma? Ska den prioritera minst antal dödsfall, ålder, kön eller social status?

Vid fortsatt utveckling av autonoma fordon är det av vikt att inkludera dessa etiska aspekter. Därutöver bör försiktighet och noggrannhet vidtas för att i största möjliga mån säkerställa utvecklingen av autonoma lastbilar.

9
Slutsats

I detta projekt har ett nyskapande arbete initierats med inspiration från tidigare års projekt, där målet var att utveckla en autonom dolly. Genom samarbete med grupp EENX16-24-05 möjliggjordes skapandet av en modulär fysisk dolly, vilken sedan användes för att implementera mjukvarulösningar för autonom körning. Projektets huvudsyfte var att utveckla mjukvarubaserade lösningar med hjälp av utvalda sensorer för att möjliggöra vägföljning och avståndshållning till framförvarande fordon. Möjligheten att koppla samman de två enheterna utforskades också. För att utvärdera algoritmernas effektivitet genomfördes relevanta tester.

Initialt fokuserades projektet på att skapa simuleringar för att visa att en fysisk implementering skulle vara möjlig, eftersom det inte fanns en tillgänglig fysisk dolly för testning. Dessa simuleringar visade sig vara framgångsrika i en simulerad miljö och möjliggjorde en snabb övergång till fysiska tester när dollyn blev tillgänglig.

Genom olika tester på vägföljningsalgoritmen, inklusive varierande vägförhållanden, kunde vi dra slutsatsen att systemet uppfyller den ursprungliga målsättningen. Trots detta finns det utrymme för vidareutveckling, särskilt inom navigation och robusthet i algoritmen. Framtida projekt bör adressera dessa områden, inklusive ruttplanering.

Implementeringen av avståndshållning visade också lovande resultat och möjliggjorde uppfyllandet av projektets mål, trots eventuella brister i implementationen. Även här finns det potential för förbättring, vilket bör utforskas vidare.

Avslutningsvis, även om tiden inte tillät en fysisk implementering av autonom dockning, visade simuleringar och beräkningar att tanken är genomförbar med hjälp av AruCo markörer.

Slutligen kan vi dra slutsatsen att projektet har visat att en autonom dolly är möjlig med hjälp av de utvecklade algoritmerna. Dock krävs ytterligare arbete för att säkerställa dess säkerhet och robusthet i en verklig miljö.

Referenser

[1] Trafa. "Transportarbete i Sverige 2020 - 2022," Trafikanalys. (11/04/2023), URL: https://www.trafa.se/globalassets/statistik/transportarbete/transportarbete-2022-%202023-10-04.pd (hämtad 2024-04-12).

[2] C. Badue, R. Guidolini, R. V. Carneiro m. fl., *Self-Driving Cars: A Survey*, 2019. arXiv: 1901.04407 [cs.RO].

[3] O. Pauca, C. F. Caruntu och A. Maxim, "Trajectory Planning and Tracking for Cooperative Automated Vehicles in a Platoon," i *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, 2020, s. 769–774. DOI: 10.1109/ICSTCC50638.2020.9259780.

[4] M. Csail. "Explained: Levels of autonomy in self-driving cars," MIT. (10/12/2020), URL: ttps://www.csail.mit.edu/news/explained-levels-autonomy-self-driving-cars (hämtad 2024-04-12).

[5] S. Zhiwei, H. Weiwei, W. Ning m. fl., "Map free lane following based on low-cost laser scanner for near future autonomous service vehicle," i *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015, s. 706–711. DOI: 10.1109/IVS.2015.7225767.

[6] E. N. advanced, Skoda, 10/12/2013. URL: https://www.euroncap.com/en/ratings-rewards/euro-ncap-advanced-rewards/2013-skoda-lane-assistant/.

[7] E. A. och Jakob Bergman och David Espedalen och Isak Kjelsson och Rasmus Mårdberg och Ibrahim Timraz, "Autonom styrning av en lasbil," Chalmers Tekniska Högskola, 2023.

[8] Z. Xu, X. Baojie och W. Guoxin, "Canny edge detection based on Open CV," i *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, 2017, s. 53–56. DOI: 10.1109/ICEMI.2017.8265710.

[9] "Find edges in 2-D greyscale image - MATLAB edge," The MathWorks Inc. (n.d.), URL: https://se.mathworks.com/help/images/ref/edge.html (hämtad 2024-03-26).

[10] A. R. "Hough Transforms in Image Processing," Scaler. (2023-06-23), URL: https://www.scaler.com/topics/hough-transform-in-image-processing/ (hämtad 2024-05-10).

[11] Wikipedia. "Pendulum," Wikipedia. (), URL: https://en.wikipedia.org/wiki/Pendulum (hämtad 2024-05-03).

[12] Wikipedia. “Kinematic model,” Wikipedia. (), URL: https://en.wikipedia.org/wiki/Kinematics (hämtad 2024-05-03).

[13] KTH. “Tillståndsmodeller för kontinuerliga system,” KTH. (2024), URL: https://www.kth.se/social/upload/4f509540f2765406df000001/for14-16-kap14.pdf (hämtad 2024-04-30).

[14] libretexts. “P, I, D, PI, PD and PID controls,” University of Michigan. (), URL: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/09%3A_Proportional-Integral-Derivative_(PID)_Control/9.02%3A_P%2C_I%2C_D%2C_PI%2C_PD%2C_and_PID_control (hämtad 2024-05-01).

[15] William Franklin. “Kalman Filter Explained Simply.” (2020-12-31), URL: https://thekalmanfilter.com/kalman-filter-made-easy-ebook/ (hämtad 2024-03-27).

[16] Melda Ulusoy. “Why Use Kalman Filters? | Understanding Kalman Filters, Part 1,” Mathworks. (2017-01-30), URL: https://se.mathworks.com/videos/understanding-kalman-filters-part-1-why-use-kalman-filters--1485813028675.html (hämtad 2024-03-27).

[17] Dyoxygen. “Detection of ArUco Markers,” OpenCV. (24/03/2024), URL: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (hämtad 2024-03-25).

[18] Dyoxygen. “Calibration with ArUco and ChArUco,” OpenCV. (24/03/2024), URL: https://docs.opencv.org/4.x/da/d13/tutorial_aruco_calibration.html (hämtad 2024-03-25).

[19] D. J. Yeong, G. Velasco-Hernandez, J. Barry och J. Walsh, “Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review,” *Sensors*, årg. 21, s. 2140, 2021-03. DOI: 10.3390/s21062140.

[20] M. C. Roshan, M. Isaksson och A. Pranata, “A geometric calibration method for thermal cameras using a ChArUco board,” *Infrared Physics & Technology*, s. 105 219, 2024.

[21] A. Marut, K. Wojtowicz och K. Falkowski, “ArUco markers pose estimation in UAV landing aid system,” i *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, 2019, s. 261–266. DOI: 10.1109/MetroAeroSpace.2019.8869572.

[22] Dyoxygen. “Camera Calibration and 3D Reconstruction,” OpenCV. (11/04/2024), URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d (hämtad 2024-04-12).

[23] S. G. Manyam, D. Casbeer, A. L. V. Moll och Z. Fuchs, “Shortest Dubins path to a circle,” i *AIAA Scitech 2019 Forum*. DOI: 10.2514/6.2019-0919. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2019-0919. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2019-0919.

[24] N. Developers. “numpy.polyfit,” Numpy. (), URL: https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html (hämtad 2024-04-07).

[25] N. Developers. “numpy.poly1d,” Numpy. (), URL: https://numpy.org/doc/stable/reference/generated/numpy.poly1d.html (hämtad 2024-04-07).

[26] A. Sakai, *Dubins path planning*, Available at https://atsushisakai.github.io/PythonRobotics/modules/path_planning/dubins_path/dubins_path.html (2022/05/17).

[27] “Raspberry pi 5.” (24/03/2024), URL: https://botland.store/img/cms/23905_9.jpg (hämtad 2024-03-25).

[28] “Datasheet för Raspberry pi,” Raspberry Pi Ltd. (2023-10), URL: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf (hämtad 2024-03-26).

[29] “Active cooler.” (24/03/2024), URL: https://www.raspberrypi.com/products/active-cooler/ (hämtad 2024-03-25).

[30] B. Croston. “Rpi.GPIO.” (2022-02-06), URL: https://pypi.org/project/RPi.GPIO/ (hämtad 2024-03-26).

[31] B. Nuttall, gpiozero. (), URL: https://gpiozero.readthedocs.io/en/stable/ (hämtad 2024-05-09).

[32] B. Nuttall, gpiozero. (), URL: https://gpiozero.readthedocs.io/en/stable/api_output.html#gpiozero.PWMOutputDevice (hämtad 2024-04-25).

[33] Raspberry Pi Ltd. (), URL: https://www.raspberrypi.com/documentation/computers/images/GPIO-Pinout-Diagram-2.png (hämtad 2024-04-25).

[34] C. Davidson-Pilon. (2024), URL: https://pypi.org/project/rpi-hardware-pwm/ (hämtad 2024-04-25).

[35] (2024), URL: https://forums.raspberrypi.com/viewtopic.php?t=367294 (hämtad 2024-04-25).

[36] “FIT0729 Datasheet,” DFRobot. (), URL: https://docs.rs-online.com/88e0/A700000008916437.pdf (hämtad 2024-05-02).

[37] W. J. M. Rankine, *A manual of civil engineering*. 1883, s. 651–653. URL: https://archive.org/details/amanualcivileng02rankgoog/page/650/mode/2up.

[38] D.328, *Clothoid curve, a type of transition curve*. 4/11/2008. URL: https://sv.wikipedia.org/wiki/Klotoid#/media/Fil:Clothoid_curve.svg.

[39] Elecfreaks. “Ultrasonic Ranging Module HC - SR04,” Elecfreaks. (), URL: https://www.electrokit.com/upload/product/41013/41013207/HC-SR04.pdf (hämtad 2024-03-26).

[40] Paralax. “28015 Datasheet by Parallax Inc.,” Parallax Inc. (2/4/2013), URL: https://www.digikey.com/htmldatasheets/production/1253021/0/0/1/28015.html (hämtad 2024-03-24).

[41] Raspberry Pi Foundation. URL: https://projects.raspberrypi.org/en/projects/physical-computing/12.

[42] A. Instructables. “3.3V Mod for Ultrasonic Sensors,” Autodesk Instructables. (), URL: https://www.instructables.com/Modify-Ultrasonic-Sensors-for-3-Volts-Logic-prepar/ (hämtad 2024-05-03).

[43] R. Pi. “Datasheet,” Raspberry Pi. (), URL: https://datasheets.raspberrypi.com/cm/cm1-and-cm3-datasheet.pdf (hämtad 2024-05-03).

[44] “Ziegler-Nichols Tuning Rules for PID,” Microstars Laboratories. (), URL: https://www.mstarlabs.com/control/znrule.html#Ref5 (hämtad 2024-05-02).

[45] CARLA. “Generate a New Map in CARLA,” CARLA. (), URL: https://carla.readthedocs.io/en/latest/tuto_M_generate_map/ (hämtad 2024-05-09).

[46] ATBS staff. “Self-Driving Trucks: Are Truck Drivers Out of a Job?” ATBS. (), URL: https://www.atbs.com/post/self-driving-trucks-are-truck-drivers-out-of-a-jo (hämtad 2024-01-31).

[47] TuSimple’s editorial team. “Self-Driving Trucks: Are Truck Drivers Out of a Job?” TuSimple. (n.d.), URL: https://www.tusimple.com/blogs/tusimple-demonstrates-autonomous-system-fuel-efficiency-improvements-through-on-road-maneuver-performance-study/ (hämtad 2024-03-26).

[48] “Huvudsakliga risker i trafiken,” Arbetsmiljöverket. (2024-01-03), URL: https://www.av.se/halsa-och-sakerhet/sakerhet-i-trafiken/huvudsakliga-risker-i-trafiken/ (hämtad 2024-03-26).

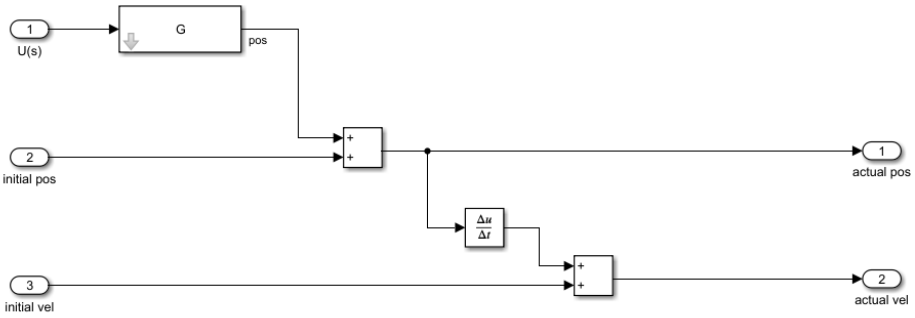
[49] “Grönt ljus för 34,5 meter långa lastbilar – minskar utsläppen med 4–6 procent,” Transportstyrelsen. (2023-12-01), URL: https://www.transportstyrelsen.se/sv/Nyhetsarkiv/2023/gront-ljus-for-345-meter-langa-lastbilar--minskar-utslappen-med-46-procent/ (hämtad 2024-02-05).

[50] I. Persson. “Framtidens lastbilar är förarlösa och eldrivna,” Vattenfall. (2021), URL: https://www.vattenfall.se/fokus/trender-och-innovation/sjalvkorande-lastbilar/ (hämtad 2024-03-26).

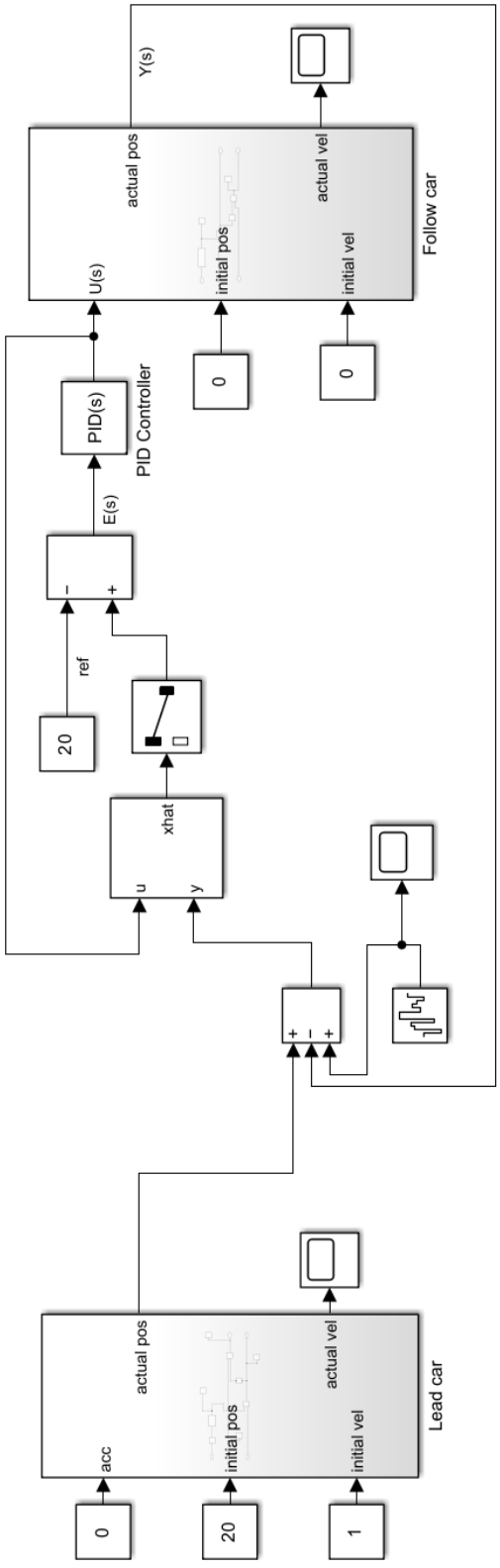
[51] “Självkörande bilar i EU: från science fiction till verklighet,” Europaparlamentet. (2019-01-15), URL: https://www.europarl.europa.eu/news/sv/headlines/economy/20190110STO23102/sjalvkorande-bilar-i-eu-fran-science-fiction-till-verklighet (hämtad 2024-03-26).

A

Appendix 1



Figur A.1: Simulink modell för följefordon. En liknande modell användes för att beskriva ledarforsdonet.



Figur A.2: Simulink modell för avståndshållning

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS Tekniska Högskola
Göteborg, Sverige
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY