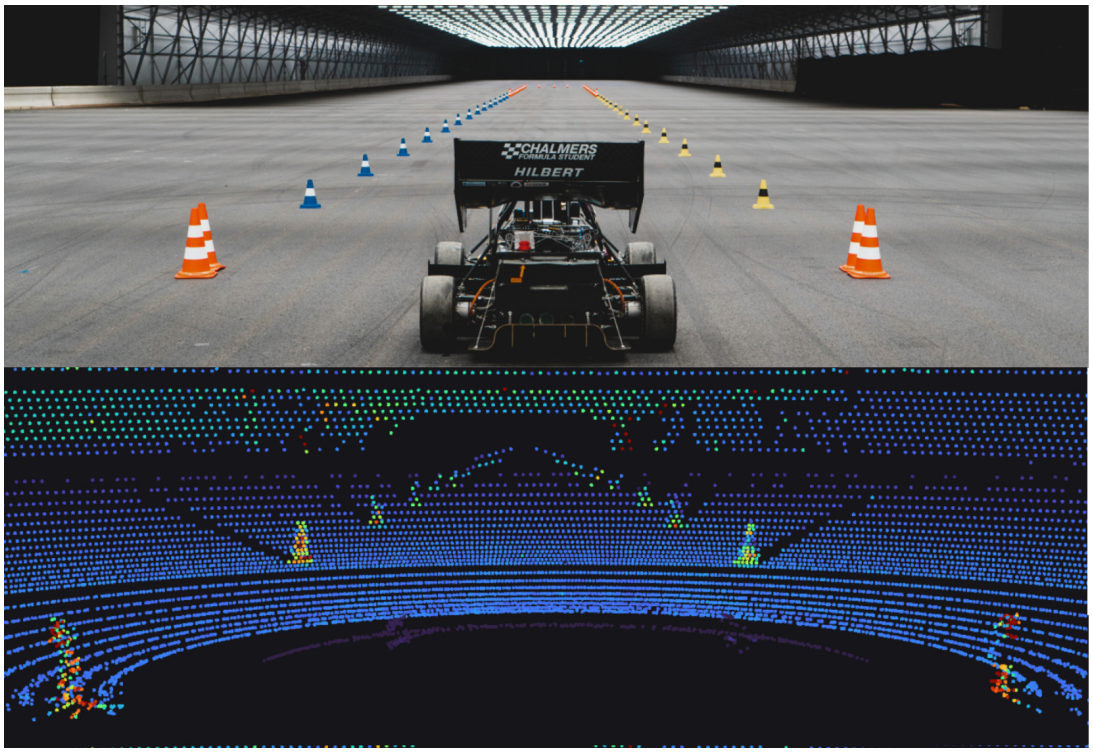
CHALMERS
UNIVERSITY OF TECHNOLOGY



Sensor fusion med LiDAR för antispinn till Chalmers Formula Students bil

Kandidatarbete vid institutionen för elektroteknik

Felix Dahlin, Felix Dahlström, Rafat Haque,
Adam Herbertsson, Noak Palander, Savinjith Walisadeera

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

Kandidatarbete 2025

Sensor fusion med LiDAR för antispinn till Chalmers Formula Students bil

Felix Dahlin, Felix Dahlström, Rafat Haque,
Adam Herbertsson, Noak Palander, Savinjith Walisadeera

CHALMERS

Institutionen för elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025

Sensor fusion med LiDAR för antispinn till Chalmers Formula Students bil
Felix Dahlin
Felix Dahlström
Rafat Haque
Adam Herbertsson
Noak Palander
Savinjith Walisadeera

Handledare: Professor Jonas Sjöberg, Chalmers Tekniska Högskola
Examinator: Docent Karinne Ramirez-Amaro, Chalmers Tekniska Högskola

Kandidatarbete 2025
Institutionen för elektroteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Telefon: +46 31 772 1000

Framsida: Ursprunglig bild från Chalmers Formula Student [1]

Skrivet i LaTeX, mall från Kyriaki Antoniadou-Plytaria
Göteborg 2025

Sammandrag

I detta arbete presenteras en implementation samt framtagande av ett antispinnsystem för autonomkörning av Chalmers Formula Students bil. Implementationen bygger på ett `ROS2`-nätverk som exekverar `python, C++` och `C`-kod i en integrerad dator. Metoden bygger på att skatta en markhastighet från LiDAR data, vilket bearbetas tillsammans med hjulhastigheten och IMU-data för att beräkna glidfaktorn. Detta skickas till ett reglersystem för att justera vridmomentet med målet att nå den optimala glidfaktorn för maximal acceleration. Resultaten visar att markhastighet kan estimeras från LiDAR-data och därefter beräkna glidfaktorn. Slutsatsen av projektet är att ett antispinnsystem kan baseras på LiDAR-odometri.

Abstract

In this work, an implementation and development of a traction control system for autonomous driving of Chalmers Formula Student's car is presented. The implementation is based on a `ROS2` network that executes `python, C++`, and `C` code on an integrated computer. The method is based on estimating a ground speed from LiDAR data, which is processed together with wheel speed and IMU data to calculate the slip level. This is then sent as a signal to a torque control system with the goal of finding the optimal slip ratio for maximum acceleration. The results show that ground speed can be estimated from LiDAR data. The conclusion drawn is that it is possible to base a traction control system on LiDAR odometry.

Förord

Denna rapport presenterar arbetet som gjordes under ett kandidatarbete 2025. Arbetet gjordes på elektroteknik institutionen E2 på Chalmers Tekniska Högskola. Projektet görs i samband med Chalmers Formula Student-projektet för 2025.

Vi vill tacka Arthur Alexandersson och William Almqvist, medlemmar i Chalmers Formula Student 2025, som hjälpte oss förstå existerande system och integrering av arbetet med årets mjukvara. Vi skulle även vilja medföra vår tacksamhet till Andreas Åberg och Anton Rosén för deras hjälp att få igång projektet och formulera problembeskrivning och syfte för kandidatarbetsförslaget. Till sist vill vi tacka vår examinator Docent Karinne Ramirez-Amaro och handledare Professor Jonas Sjöberg.

Felix Dahlin, Felix Dahlström, Rafat Haque,

Adam Herbertsson, Noak Palander, Savinjith Walisadeera,

Göteborg, Maj 2025.

Förkortningar

CFS	Chalmers Formula Student
CFS25	Chalmers Formula Students lag år 2025
IMU	Inertial Measurement Unit
LiDAR	Light detecting and ranging
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
RANSAC	Random sample consensus
EKF	Utökat Kalmanfilter (Extended Kalman filter)
KF	Kalmanfilter
MA	Moving-average
ROS	Robot operating system
RPM	Varv per minut
GSS	Markhastghetssensor

Nomenklatur

ξ	Glidfaktor [1]
v	Hastighet [m/s]
Δt	Tidssteg [s]
Δs	Sträcka [m]
ω	Hjulhastighet [rad/s]
$\alpha, \dot{\omega}$	vinkelacceleration [rad/s^2]
a,$\dot{v}$	Acceleration [m/s^2]
$\hat{\mathbf{x}}$	Skattad tillståndsvektor
$\mathbf{r}$	Ortsvektor [m, m, m]
r	Hjulradie [m]
$K(k+1\|k)$	Kalman-återkoppling (vid tidsstegsindex $k+1$ givet information från tidsstegindex k)
z_x	Omätbar friktionskraft
σ	Friktionsestimeringsparameter (LuGre)
v_s	Stribeck-hastighet [m/s]
v_{rel}	Relativ hastighet [m/s]
K_p	Proportionell term i regulator [Nm]
$\mathbf{u}$	Styrsignal (Vridmoment) [Nm, Nm, Nm, Nm]
e	Felterm [1]
$\boldsymbol{y}$	Sanna tillstånd
F_f	Friktionskraft [N]
F_{drag}	Luftmotstånd [N]
F_{lift}	Lyftkraft [N]
F_a	Accelerationskraft [N]
$\mathbf{w}$	Brusterm
G	Bilmodellssystem

(x, y, z)	Kartesiska koordinater [m, m, m]
(r, θ, φ)	Sfäriska koordinater [m, rad, rad]
P_t	Punktmoln vid tid t [m, m, m]
C_i	Kluster med index i [m, m, m]
N_d	Antal punkter i ett kluster [1]
ω_x	Pitch [rad/s]
ω_y	Roll [rad/s]
ω_z	Girhastighet (Yaw) [rad/s]
m	Bilens massa [kg]
I	Tröghetsmoment [kgm^2]
M_f	Friktionsmoment [Nm]
d	Spårvidd (avstånd mellan hjul) [m]
l_F	Avstånd från tyngdpunkt till framaxel [m]
δ	Styrvinkel för framhjul [rad]

Innehåll

Figurlista

Tabellista

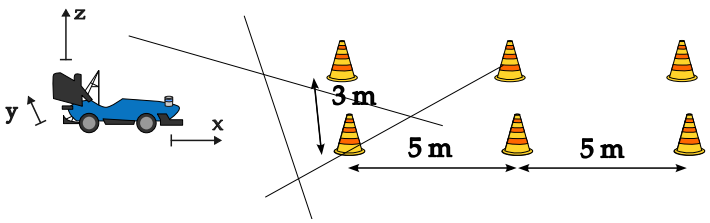
1

Introduktion

1.1 Bakgrund

Antispinn-system (engelska: *traction control*) skapades först på 1970-talet för att förhindra att förare förlorar vägfästet genom att tillföra för mycket vridmoment till drivhjulen. Idag finns systemet i alla nytillverkade personbilar som en viktig säkerhetsåtgärd.

Chalmers Formula Students (CFS) bil tävlar i både autonoma och manuella körevent. Ett av tävlingseventen är accelerationstävlingen där bilen kör i en raksträcka på 75 m från stillastående start (figur 1.1) [2]. Accelerationseventet motsvarar 75 poäng på en Formula Student-tävling vilket är 7.5% av den totala poängen. Designen av körsystemen på bilen ger också poäng på ingenjörsdesigneventet som motsvarar 150 poäng (15%) på tävlingen.



Figur 1.1: Fysiska omständigheter för uppställning av accelerationseventet.

I dagsläget kostar effektiva markhastighetssensorer för högprestanda-applikationer väldigt mycket. CFS25 samlade in data med en markhastighetssensor av Sensoric Solutions som var värd över 300 000 kr [3]. Fordonshastigheten är viktig för att kunna estimera andra delar av fordonsdynamiken som är viktigt i styrningen av ett autonomt fordon [4]. Hjulhastigheten mäter samtidigt rotationshastigheten av däcken vilket används för att förstå kraftresultanterna i fordonet. CFS25 har tillgång till en LiDAR, hjulhastighetssenorer och en accelerometer(IMU). Genom att kombinera de existerande sensorerna är det möjligt att skapa en markhastighetsestimering genom att endast använda mjukvara och spara stora kostnader.

Glidfaktorn (engelska: *slip*) ξ är det dimensionslösa förhållandet mellan hjulens ro-

tationshastighet och markhastigheten och beräknas enligt:

$$\xi = \frac{\omega r - v}{\omega r}, \tag{1.1}$$

där ω (rad/s) är hjulets vinkelhastighet, r (m) är hjulets radie och v (m/s) är markhastigheten. CFS25 har kommit fram till en optimal glidfaktor på $\xi_{\text{opt}} \approx 13\%$ för bilens däck [1]. I nuläget finns inget reglersystem för däckglid på CFS-bilen under varken manuellt eller autonomt körläge. Lösningen för detta är att skapa ett system som använder bilens existerande sensorer för att estimera glidfaktorn och optimera den genom att tillföra rätt mängd vridmoment till hjulen [5]. Rapporten beskriver framtagandet av ett sådant system för de redan existerande sensorerna. Ett antispinn-system skulle innebära en stor vinst då systemet reglerar vridmomentet efter vägfästet.

1.2 Syfte

Syftet med arbetet var att utvärdera om ett antispinn-system kan baseras på sensorfusion av hastighetsestimering från LiDAR data och hjulhastighetssensorerna. Projektet var att genom mjukvarulösningar till Chalmers Formula Student utvärdera ett system som estimerar hastighet, beräknar glidfaktor och reglerar vridmoment för att nå optimal glidfaktor ξ_{opt}.

1.3 Översiktlig systembeskrivning

Rapporten beskriver framtagandet av ett antispinn-system som med LiDAR-estimerad hastighet och hjulhastighet reglerar vridmomentet i hjulmotorn för att nå den optimala glidfaktorn. Projektet var uppbyggt i följande tre delar:

1. Ta ut kon-positioner med LiDAR-sensorn att beräkna fordonshastigheten v genom en algoritm. Samtidigt mäts hjulrotationshastigheten ω av en separat sensor. Detta behandlas i kapitel 3.

2. Därefter genomförs sensorfusion mellan hjul- och fordonshastighet för att estimera glidfaktorn ξ. Detta görs genom att jämföra fordonets hastighet v med hjulens rotationshastighet ω enligt ekvation 1.1. Detta behandlas i kapitel 4.

3. En regulator har därefter implementerats för att optimera vridmomentet till hjulen och därigenom reglera vridmomentet för att uppnå optimumvärdet $\xi_{\text{opt}} = 0.13$. Detta behandlas i kapitel 5.

1.4 Relaterade arbeten

Arbeten relaterade till LiDAR-estimering av hastigheter, odometri och antispinnsystem har gjorts tidigare. En undersökning av hastighet från LiDAR-data har gjorts av J. Zhang, W. Xiao, B. Coifman och J. P. Mills. De använde LiDAR-data och jämförde med tidigare tidssteg för att ta fram ändringen i sträcka och i sin tur föremålets hastighet [6]. De uppnådde en träffsäkerhet på 94 % med ett fel på 0.22 m/s. Denna undersökning gjordes med en statisk 3D-LiDAR som mätte fordon i rörelse.

I ett arbete av David J. Yoon, Keenan Burnett, Johann Laconte, Yi Chen, Heethesh Vhavle, Soeren Kammel, James Reuther, och Timothy D. Barfoot år 2023 undersöktes LiDAR-odometri med Doppler-effekten [7]. I denna undersökning används 'Iterative Closest Point' för att öka beräkningshastigheten. Metoden använder en 'Nearest Neighbour Search' för att välja ut statiska punkter inom en LiDAR-sensors räckvidd. I undersökningen beräknas den relativa sträckan genom att jämföra masscentrum-estimeringen av ett statiskt objekt mellan tidssteg.

För att acceleration ska kunna ske krävs en positiv kraftresultant och en relativ hastighetsskillnad mellan hjul- och fordonshastighet [8]. Vid ett specifikt värde för däckmodellen kommer denna resulterande kraft vara maximal enligt arbetet gjort av L. P. Guan och H Wang B. I arbetet försöker de hitta maximal friktion för att kunna maximera grepp och överföringen av vridmomentet i däcken till longitudinell kraft. Arbeten gjort av Kungliga Tekniska Högskolans Formula Student-lag KTH Formula student och IIT Delhi tog fram reglersystem för att maximera detta [9] [10]. I båda Formula student-lagen byggdes antispinnsystem för att effektivisera accelerationsfasen under accelerationseventet i Formula Student fram. En referensglidfaktor som är optimal för däckmodellen används. Arbetena jämför hur bilen accelererar för maximal gas med vridmomentsoptimeringen kontra `IPG Carmaker`-föraren. I reglersystemen används en PID-regulator i ett öppet styrt system för att ansätta optimalt vridmoment för varje hjulmotor. Undersökningarna fann att ett antispinnsystem med den optimala glidfaktorn för den korresponderande däckmodellen är bättre än en virtuell förare.

Avgränsningar

Projektet syftade endast till att undersöka accelerationsdeltävlingen eftersom det redan finns andra system som är optimerade till banorna med flera svängar. Projektet avgränsades på grund av:

- Under projektets tidsplan var CFS25-bilen ej konstruerad. Detta innebar att resultat baseras på återanvändning av existerande uppmätt data samt simuleringar och egna enhetstester.

- I projektet beaktas endast en plan rak körbana med torr asfalt som väglag. Detta beror på att inspelad data endast kommer från dessa förhållanden.
- Endast hastigheter som uppnås i accelerationseventet beaktas.

1.5 Samhälleliga och etiska aspekter

Utvecklingen av ett antispinnsystem bidrar till säkrare fordon. Detta kan också ge en falsk känsla av säkerhet, särskilt om systemet inte fungerar som det ska. Det är därför viktigt att noggrant testa och verifiera systemets funktion.

1.6 Rapportstruktur

I nästkommande del av rapporten presenteras teori för mjuk- och hårdvara på bilen som arbetet bygger på. I denna del beskrivs de existerande system som uppdaterades. I de tre efterföljande kapitlen beskrivs teori, implementering och resultat av de tre delarna av projektet som identifieras i 1.3. I rapportens näst sista del diskuteras det sammanställda resultatet av alla delmoment i de tidigare delarna. Till sist dras slutsatser av resultaten och förslag ges på vidare arbeten.

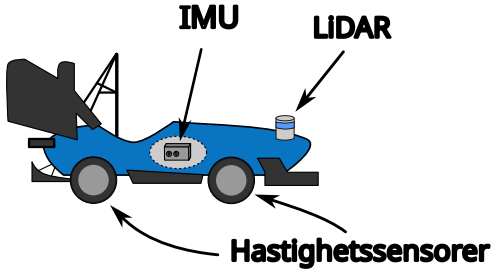
2

Existerande system och utgångspunkt

Arbetet utgår från CFS25-bilen vad gäller sensorer och uppbyggnad för att basera systemet på verkliga komponenter. Detta avsnitt beskriver bilens relevanta mjuk- och hårdvara vid utgångsläget för arbetet.

2.1 Sensorer

Nedan följer en beskrivning av relevanta sensorer i CFS-bilen för antispinnsystemet. Sensorernas placering illustreras i figur 2.1.

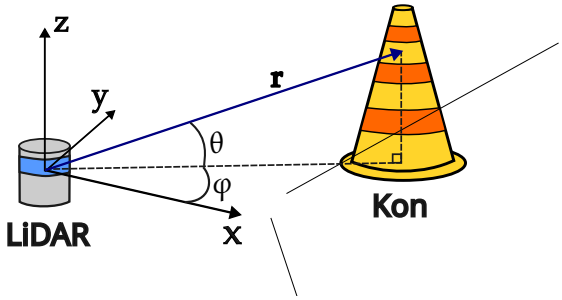


Figur 2.1: Bilens olika sensorers placering på bilen (notera ej skalenlig).

2.1.1 LiDAR

En LiDAR (Light Detection And Ranging) fungerar genom att skicka ut laserstrålar som reflekteras tillbaka från en träffyta [11]. Därefter bearbetas lasersignalerna som har reflekterats, där varje mätpunkt kopplas till ett avstånd mellan sensor och träffyta samtidigt som riktningen är känd (θ, φ), vilket illustreras i figur 2.2. Avståndet till träffytan är $|\mathbf{r}| = c\Delta t/2$ där c är ljusets hastighet och Δt är tiden till

att signaler återfås av sensorn [11]. Detta beräknas sedan till en koordinat i rummet $\mathbf{r} = (x,y,z) \in \mathbb{R}^3$ och återskapar en bild av omgivningen bestående av punkter på ytor. Dessa uträkningar utför LiDAR-modulen och skickas som en mängd med punkter som vid mottagning omvandlas till ett punktmolnsobjekt för `ROS2`. I CFS25 används en LiDAR av fabrikat HESAI modell Pandar 64 med uppdateringshastighet 20 Hz. samt en räckvidd på 200 m (se figur 2.4) [11].

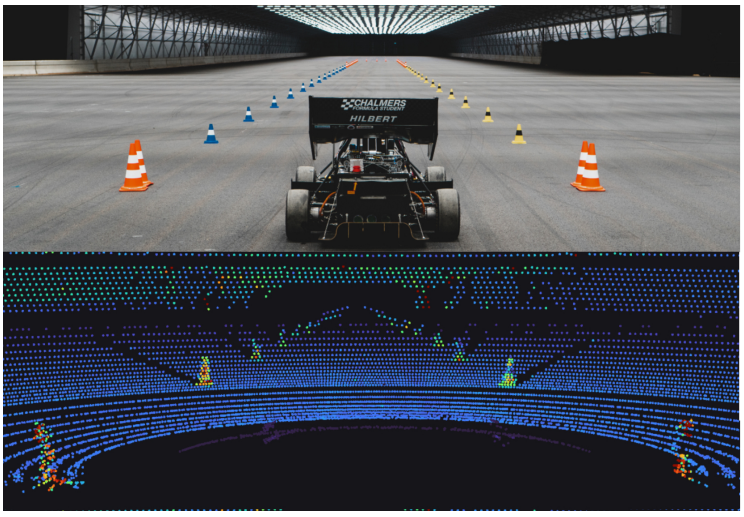


Figur 2.2: Koordinatsystemet för LiDAR-enhetens mätningar.

LiDAR-sensorns utdata är punktmoln vilket är en mängd positionsdata, tillsammans med en LiDAR-intensitet som beskriver reflektionsstyrkan som respektive punkt har. Ett punktmoln från sensorn vid tid t kan således beskrivas som

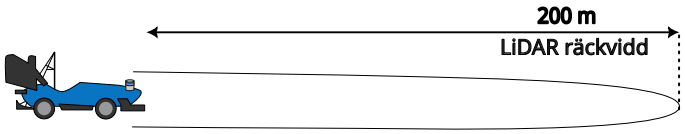
$$P_t = \{(x_{n,t}, y_{n,t}, z_{n,t}, I_{n,t}) \mid n = 1, \ldots, N\},$$

där N är det totala antalet punkter i punktmolnet och $I_{n,t}$ är intensiteten. För att beräkna avstånd mellan punkter och LiDAR:n behövs inte intensiteten och därför beaktas endast x-,y- och z-komponenterna av P_t.



Figur 2.3: Bild av körbanan och motsvarande punktmoln.

I punktmolnet identifieras närliggande punkter, vanligen kallat kluster, genom en algoritm baserad på Euklidisk klusterbildning [12] och DBSCAN [13]. Algoritmens



Figur 2.4: Räckvidd för LiDAR-enhet.

i:te identifierade kluster C_i är en mängd av punkter $\mathbf{p}_j \in P_t$ med följande villkor:

$$\|\mathbf{p}_j - \mathbf{p}_\ell\| \leq \varepsilon \, , \forall \mathbf{p}_j, \mathbf{p}_\ell \in C_i,$$

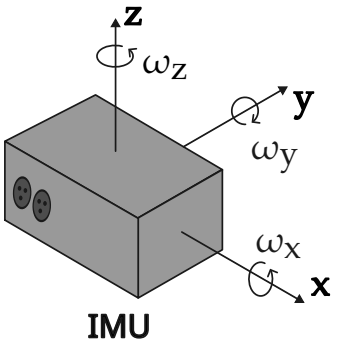
för något $\varepsilon > 0$. För C_i gäller då

$$C_i \cap C_j = \emptyset, \text{ för } i \neq j \text{ och } \bigcup_i C_i \subseteq P_t.$$

Algoritmen analyserar alla kluster och identifierar de som motsvarar koner. Från detta kan väldefinierade objekt skapas som bilen förhåller sig till. LiDAR-sensorn som används i CFS är speciell då HESAI Pandar kommer med egna drivrutiner [11].

2.1.2 Inertial Measurement Unit (IMU)

En annan sensor är *Inertial Measurement Unit* (IMU) och illustreras i figur 2.5. Den mäter accelerationer $(a_x, a_y, a_z) = (\ddot{x}, \ddot{y}, \ddot{z})$ samt rotationshastigheter $(\omega_x, \omega_y, \omega_z)$ (engelska: 'pitch, roll, yaw') i tre kartesiska dimensioner i 100 Hz [1]. CFS25 använder en Honeywell HG4930 [14].



Figur 2.5: Illustration av IMU som mäter accelerationer och rotationer.

2.1.3 Hjulhastighetssensorer

På CFS-bilen finns en hjulhastighetssensor vid varje hjul som mäter rotationshastigheten ω (rad/s). Mätfrekvensen för hjulhastighetssensorerna är 100 Hz. Hastigheten

beräknas i rotation per minut f_{wheel} (RPM) och översätts till en rotationshastighet genom en transformation [1]:

$$\omega = \frac{f_{\text{wheel}}\pi \cdot r_{\text{wheel}}}{390}$$

Sensorerna som används för varje hjul är Heidenhain ECI1119 [15].

2.2 Hjulmotor

CFS25 bilen är en helt fyrhjulsdriven elbil som har fyra hjulmotorer för att driva fordonet. Hjulmotorerna är gjorda av laget under våren 2025 [1] med maximalt varvtal på 20 000 RPM och nominellt vridmoment på 220 Nm. Reglerna i Formula Student begränsar effekten till motorerna till 80 kW [2].

2.3 Mjukvara

Efter att sensorerna har utfört en mätsekvens skickas och bearbetas detta med hjälp av mjukvara. Det finns olika sorters mjukvarudelar, vilket beskrivs vidare i följande delar. Mjukvaran är implementerad i en robotik-specificerad dator Nvidia Jetson AGX Orin [16].

2.3.1 `ROS2`-nätverket

CFS har utvecklat ett nätverk av `ROS2`-noder som möjliggör kommunikation mellan bilens mjukvaru- och hårdvarukomponenter via så kallade *topics* [17]. Dessa topics kan publiceras eller prenumereras på av noder, där varje nod representerar en specifik del av bilens system.

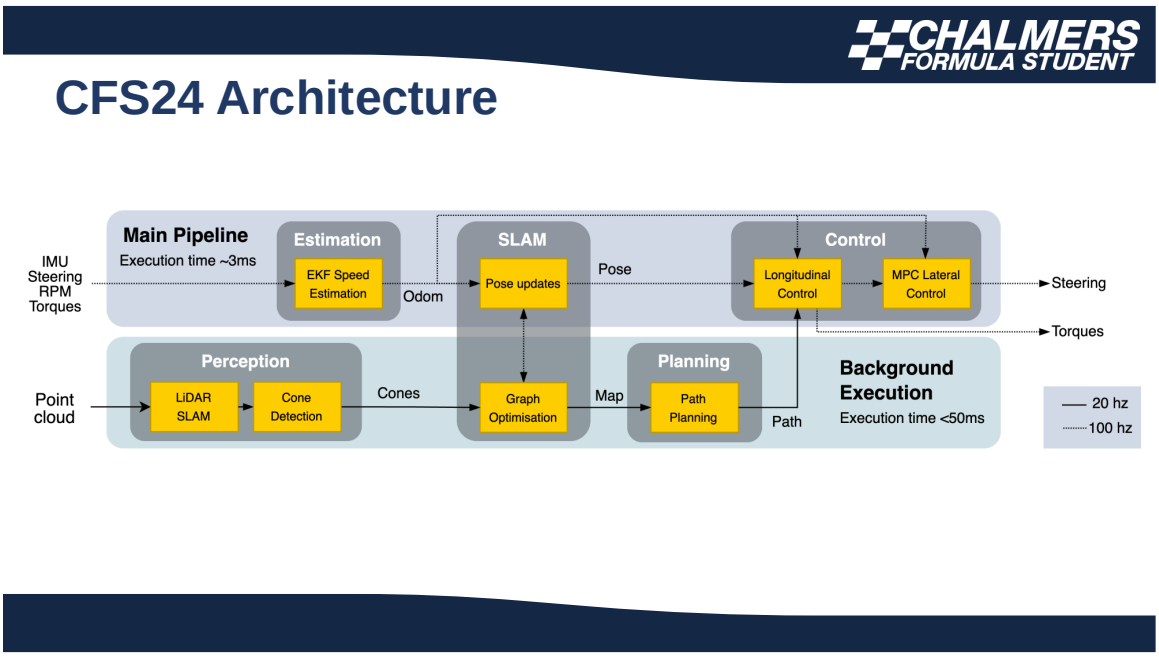
I det befintliga nätverket används LiDAR-sensorn endast för att detektera koner och planera körbanor, men utan hastighetsmätning. I den fysiska bilen skickas sensordata som topics, men systemet kan även köras i simuleringsläge där verklig sensordata ersätts av syntetisk. Det övergripande nätverket består av följande delsystem, som även illustreras i figur 2.6:

1. **Perception** - Detta delsystem bearbetar punktmoln från LiDAR-sensorn för att detektera koner. Processen inkluderar borttagning av markplanet, klusterbildning av punkter samt filtrering av objekt som inte motsvarar konformen.
2. **Estimation** - Tar in signaler från hjulhastighetssensorer och bilens IMU för att med hög noggrannhet uppskatta bilens hastighet, acceleration, tröghetsmoment och andra tillstånd med hjälp av Kalmanfiltrering.

3. **SLAM** (Simultaneous localization and mapping) - Används för att relatera objekt i punktmoln över tid och skapa en karta av körbanan. Systemet gör det möjligt för bilen att exempelvis hålla reda på redan identifierade koner och bestämma sin egen position i körbanan [18].

4. **Planning** - Används för att planera och optimera bilens rutt genom körbanan, med målet att minimera totaltid. Detta omfattar bland annat beräkning av högsta tillåtna hastigheter och planering av svängmanövrer.

5. **Control** - Denna del utgör bilens vridmomentreglering av hjulmotorerna. Ett delmål i projektet var att tillämpa ett eget reglersystem för accelerationseventet.



Figur 2.6: Översiktlig bild av CFS nuvarande system för autonom körning av bilen.

För våra ändamål behöver inte hela nätverket användas eftersom vi endast fokuserar på accelerationseventet. Till exempel behöver inte de noder som motsvarar Planning och SLAM användas eftersom vi inte behöver planera in några svängar. Däremot fungerar vår konfiltreringsalgoritm som presenteras i nästa kapitel ungefär som en förenklad version av SLAM där vi bara bryr oss om det relativa avståndet mellan en kon i två tidpunkter.

2.3.2 Inspelad Sensordata

CFS har genomfört ett flertal körningar av accelerationseventet och spelat in sensorsignaler med tidsstämpling. För varje tidpunkt kan det insamlade punktmolnet kopplas till samtidiga hjulhastigheter och IMU-data. Inspelningen har skett med hjälp av

`rosbags`, där signalerna lagras som meddelanden i olika topics. Detta möjliggör att utvecklade algoritmer och mjukvara kan testas på verklig sensordata. Under projektet användes körevent där CFS nyttjade en markhastighetssensor (engelska: *Ground Speed Sensor*, GSS) från Sensoric Solutions [3]. Under dessa event mäts markhastigheten med en precision på 0.2 mm/s och 1000 Hz uppdateringsfrekvens. En bild från accelerationstestet den dagen bifogas i figur 2.7.

Figur 2.7: Bild från bilens filmkamera från köreventet där arbetets resultat kommer ifrån.

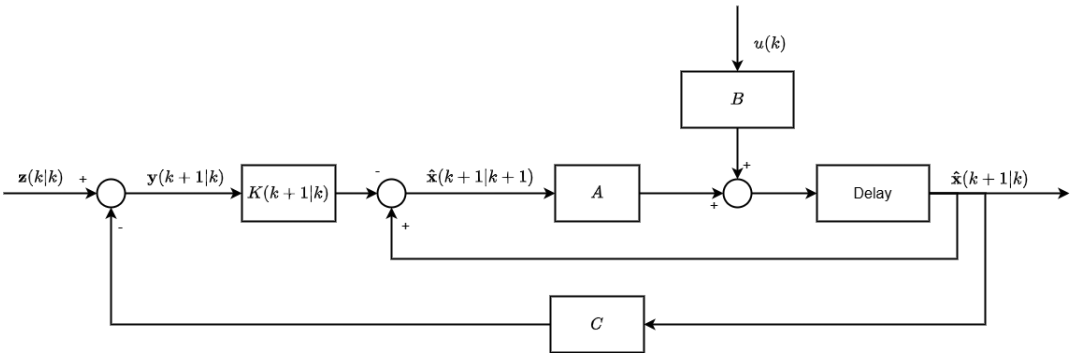
2.3.3 CFS25:s utökade Kalmanfilter

Det implementerade utökade Kalmanfiltret (EKF) i CFS25 är baserat på ett masterarbete av Nuno Alexandre de Almeida Salguiero [4]. Det finns 15 skattade tillståndsvariabler i modellen för $\hat{\mathbf{x}}$. Signalen $\mathbf{y}$ är skillnaden mellan sensordata z och skattningen. I CFS25-systemet estimeras friktionskrafterna från LuGre Tyre Dynamics enligt rapporten [4]. I modellen finns parametrar $\sigma_0, \sigma_1, \sigma_2$ som används som anpassningsbara värden för att beräkna den omätbara friktionen z_x. Stribeckhastigheten v_s är koefficient för 'slick-stick'-agerandet mellan däck och väglag. Friktionen i x-led bestäms genom:

$$
\begin{aligned}
v_{\text{rel}} &= \omega r - v_x \\
\dot{z}_x &= v_{\text{rel}} - \frac{|v_{\text{rel}}|}{v_s} \\
z_x &= \int \dot{z}_x \, \mathrm{d}t \\
F_f &= \sigma_0 z_x + \sigma_1 \dot{z}_x + \sigma_2 v_{\text{rel}}
\end{aligned}
\tag{2.1}
$$

Friktionsestimeringen är viktig för att kunna reglera vridmomentet i hjulmotorn under autonom körning [8]. Från friktionen estimeras den resulterande kraften för givet vridmoment. Detta görs genom beräkningar som hittas på sida 13 i Salguieros rapport. I den implementerade regulatorn används LuGre-modellen för att estimera hjulhastigheten ω från en modell givet en styrsignal $\mathbf{u}(k)$.

De tillstånd som ingår i CFS:s EKF är hjulhastighet för de fyra hjulen, däckkompressionsfaktorn i x- och y-led för alla 4 däck, hastigheten i x- och y-led och girhastighet ω_z. I $\mathbf{z}$ ingår sensordata från hjulhastighet, acceleration i x- och y-led samt girhastighet. Styrsignalerna $\mathbf{u}$ i tillståndsmodellen är vridmoment till varje hjulmotor. Matriserna A, B innehåller bilmodellen och utför estimeringsuträkningar för att skatta tillstånden i $\hat{\boldsymbol{x}}$. I figur 2.8 illustreras det implementerade Kalmanfiltret i CFS25-mjukvaran.



Figur 2.8: Blockschema över det existerande Kalmanfiltret.

Den predikterade kovariansmatrisen är P och process-kovariansmatrisen är Q och R är mätbrusets kovariansmatris. Matrisen C är en observationsmatris som används för att observera de tillstånd som finns i $\hat{\mathbf{x}}(k|k)$ men inte i $\mathbf{z}(k|k)$. Felet $\mathbf{y}(k|k)$ beräknas från de sanna tillstånden $\mathbf{z}$ och används för att beräkna Kalmanåterkopplingen $K(k+1|k)$. I det implementerade filtret finns sju uppmätta tillstånd: hjulhastigheterna, acceleration i x,y-led samt girhastigheten. I modellen är Kalmanåterkopplingen $K(k+1|k)$ beräknad från kovariansmatriser R, Q enligt ekvationerna 2.2:

$$
\begin{aligned}
P(k+1|k) &= AP(k|k)A^T + Q \\
S &= CP(k+1|k)C^T + R \\
K(k+1|k) &= PC^T S^{-1} \\
P(k+1|k+1) &= I - K(k+1|k)C + R \\
\mathbf{y}(k+1|k) &= \mathbf{z}(k|k) - C\hat{\mathbf{x}}(k+1|k) \\
\hat{\mathbf{x}}(k+1|k+1) &= \hat{\mathbf{x}}(k+1|k) + K(k+1|k)\mathbf{y}(k+1|k)
\end{aligned}
\tag{2.2}
$$

Glidfaktorn ξ i EKF:en beräknas enligt en komplex modell för en fyrhjulig bil från Salguieros rapport [4]. Den tar hänsyn till styrvinkel δ, girhastigheten ω_z lateral hastighet v_y och är ämnad för körevent med svängar och acceleration. Bilparametrarna l_f,d_r och d_f kan anpassas för att hitta en mer exakt. Hastigheten beräknas för varje enskilt däck innan glidfaktorn beräknas. Framdäckens glidfaktor ξ_{F} beräknas enligt:

$$
\begin{aligned}
v_{\mathrm{F}} &= (v_x - \omega_z \cdot \frac{d}{2}) \cdot \cos(\delta) + (v_y + l_f \cdot \omega_z) \cdot \sin(\delta), \\
\xi_{\mathrm{F}} &= \frac{\omega_{\mathrm{F}} \cdot r - v_{\mathrm{F}}}{v_{\mathrm{F}}}.
\end{aligned}
$$

Bakhjulens glidfaktor ξ_{R} beräknas enligt:

$$
\begin{aligned}
v_{\mathrm{R}} &= v_x - \omega_z \frac{d}{2}, \\
\xi_{\mathrm{R}} &= \frac{\omega_{\mathrm{R}} \cdot r - v_{\mathrm{R}}}{v_{\mathrm{R}}}.
\end{aligned}
$$

3

Markhastighet

Den första delen i antispinnsystemet är att estimera markhastigheten v med LiDAR-sensorn för att kunna beräkna glidfaktorn ξ. Hastighetsestimeringen från LiDAR-datan bygger på att det finns stationära nyckelpunkter i rummet som kan jämföras mellan olika mätningar [6]¡. Genom att jämföra positionsförändringen på dessa nyckelpunkter under ett visst tidsintervall kan en hastighet estimeras. Eftersom det enligt Formula Students regler finns koner utplacerade är det rimligt att använda dessa som nyckelpunkter.

3.1 Konfiltrering

För att förenkla markhastighetsberäkningen används en distansbaserad konfiltreringsalgoritm som selektivt väljer ut relevanta koner (nyckelpunkter) från deras LiDAR-kluster. Algoritmen har tre steg som upprepas vid varje mätning med en frekvens på 20 Hz:

1. Klustrets centrum (x,y) beräknas för varje kon enligt ekvation 3.1.
2. Distansen mellan varje kluster och LiDAR-sensorn beräknas enligt ekvation 3.2.
3. Ett självbalanserande träd konstrueras och de irrelevanta konerna filtreras bort.

Centrum av en kons kluster $\mathbf{c}$ bestäms av:

$$\mathbf{c} = \frac{1}{N_d} \sum_j \mathbf{p}_j, \text{ för } \mathbf{p}_j \in C_i, \tag{3.1}$$

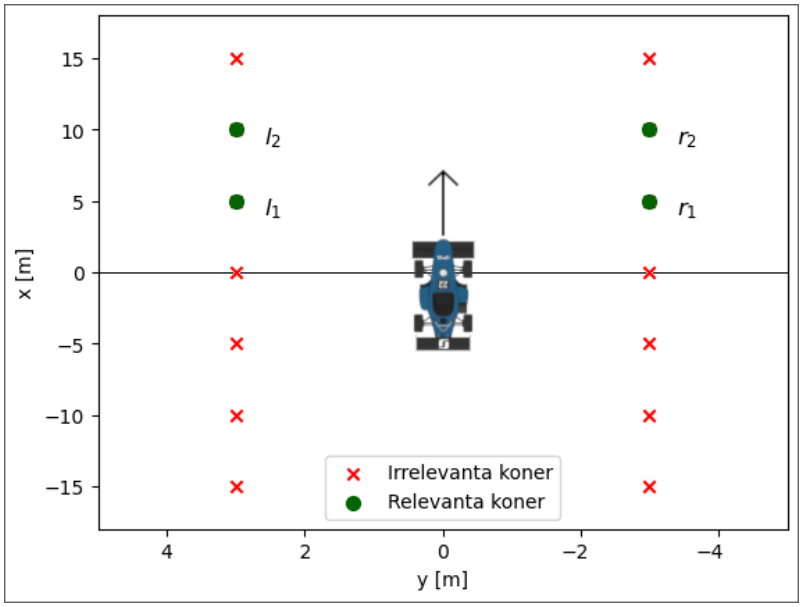
där C_i är konens LiDAR-kluster, N_d är antalet datapunkter tillhörande klustret, är klustrets centrum.

Avståndet mellan varje kon och LiDAR-sensorn bestäms med den euklidiska avståndsformeln:

$$d = \sqrt{c_x^2 + c_y^2}, \tag{3.2}$$

där d är det relativa avståndet mellan en kon och LiDAR-sensorn, där LiDAR-sensorns position är origo (0.0).

Koner räknas som irrelevanta om deras avstånd till LiDAR-sensorn är mindre än en meter framför sensorn, om de befinner sig bakom LiDAR-sensorn, eller om de inte är bland de två närmsta konerna på höger respektive vänster sida av sensorn (se figur 3.1). En kon identifieras som vänster om dess y-koordinat är positiv, respektive till höger då y-koordinaten är negativ. En kon är framför LiDAR-sensorn då dess x-koordinat är positiv och bakom om x-koordinaten är negativ (se figur 1.1).



Figur 3.1: Illustrering av relevanta samt irrelevanta koner i relation till LiDAR-sensorn.

Det självbalanserade trädet konstrueras som ett röd-svart träd med alla koners positioner från ekvation 3.2 i stigande ordning [19] [20]. Detta ger konstruktionen beräkningskomplexiteten $\mathcal{O}(n \log(n))$ där n är antalet koner. Sökningen av de relevanta konerna har beräkningskomplexiteten $\mathcal{O}(1)$ eftersom de närmaste konerna befinner sig högst upp i trädet.

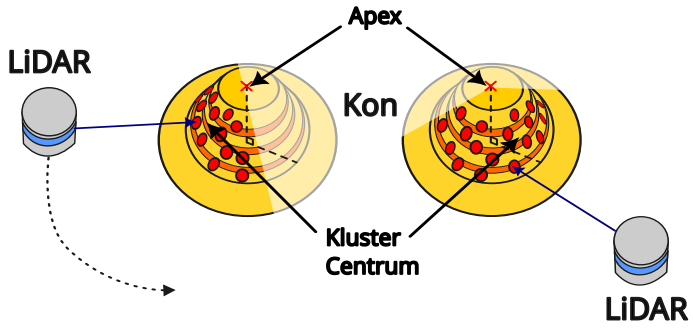
Resultatet av konfiltreringsalgoritmen är att enbart fyra koner finns kvar efter varje mätning. En pseudokodsimplementering av konfiltreringsalgoritmen presenteras i Appendix A.2.

3.2 Nyckelpunkter och konytanpassning

Efter koner är valda måste deras kluster konverteras till nyckelpunkter. Det finns minst två sätt att göra detta:

1. Använda klustrets centrum, se ekvation 3.1. Detta beräknas i konfiltreringen för att få en första ungefärlig position.
2. Anpassa klustret till en yta motsvarande en kon och extrahera apex (konens spets) som är masscentrum.

Den första naiva metoden bygger på approximationen att en kons punktkluster ej förflyttas under bilens rörelse. Detta är inte helt sant eftersom även om konen är stillastående i rummet så innebär LiDAR:ns perspektiv på konen att endast en sida kan mätas vilket gör att klustrets centrum är förskjutet mot LiDAR-enheten jämfört med konens faktiska medelpunkt. Detta gör att när bilen kör så ändras konens position i det stillastående referenssystemet, vilket syns i figur 3.2. Detta uppfattas som bilens hastighet av LiDAR-enheten.



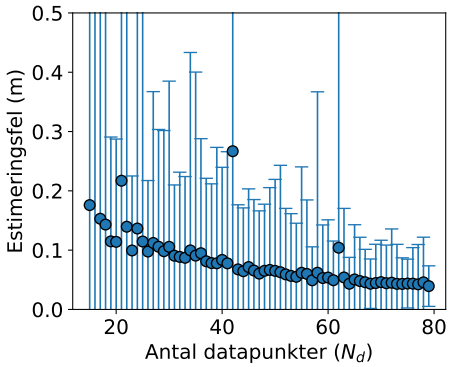
Figur 3.2: Denna bild visar en topp-vy över hur punktklustrets centrum förflyttas med LiDAR-enheten medan apex är stationärt. Detta är varför apex är att föredra som nyckelpunkt vid hastighetsberäkningar för att undvika felaktiga slutsatser om bilens förflyttning.

Den andra metoden är att anpassa en konyta till klustret och använda apex som nyckelpunkt och fördelen med detta är att apex är en stationär punkt. Nackdelen är att fel i mätningar från LiDAR:n propagerar genom anpassningen och kan ge en felaktig position som förändras mellan mätningar. Klusteranpassning till en känd geometrisk form kan göras med *Random Sample Consensus* (RANSAC)-algoritmen[21]. Det finns även parametrar för RANSAC såsom maximala antalet iterationer, föreslagna startvärden för konens geometri och tillåtna avvikelser från startvärden och klustrets form. I grunden är detta en iterativ algoritm för att estimera parametrar för en

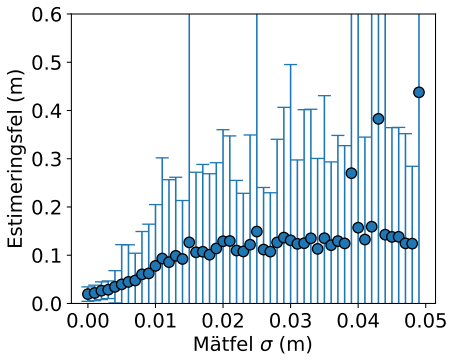
matematisk modell och i detta projekt används det för att anpassa en kon till ett punktmolnskluster genom att anpassa parametrar i konmodellen.

Det optimala är att testa detta direkt genom LiDAR-data mot en kon med känd medelpunkt och anpassa ytan och utvärdera felet. Eftersom tillgång till hårdvara saknas och denna uppsättning av data inte finns sedan tidigare används andra metoder. Punkter kan genereras slumpmässigt på en sida av en kon och sedan anpassas med RANSAC. I figur 3.3 och 3.4 syns resultatet från simuleringen i form av det fel som uppstår då en sådan anpassning utförs. RANSAC-algoritmens pålitlighet, det vill säga hur ofta den klarar att anpassa klustret till en konyta hittas i Appenix A.3.2 och simuleringskoden finns i Appendix A.3.1.

Figurerna visar en Monte Carlo-sampling [22] av fel vid konpositionsestimering genom anpassning av kon till konkluster genom att variera antalet datapunkter N_d eller felet σ för punkter. Slumpmässiga punkter på en konyta (en sida) $P = \{(x_n, y_n, z_n) \mid n = 1, \ldots, N_d\}$ genereras. Därefter skapas stokastiska fel $\epsilon_{i,n} \sim \mathcal{N}(0.\sigma^2)$, $\forall n \in \{1, \ldots, N_d\}$, $i \in \{1,2,3\}$ och sedan anpassas $P' = \{(x_n + \epsilon_{1,n}, y_n + \epsilon_{2,n}, z_n + \epsilon_{3,n}) \mid n = 1, \ldots, N_d\}$ till ytan varvid fel till centrum beräknas. För varje punkt i graferna utförs detta 1000 gånger varvid en standardavvikelse på estimeringsfelet kan extraheras. Konens dimensioner är 0.3 m hög med en radie 0.13 m.



Figur 3.3: Anpassning av konkluster med olika många datapunkter N_d till en konmodell. Beräkningar har $\sigma = 0.01$ m.



Figur 3.4: Anpassning av kon till konkluster med punkter som har ett visst mätfel. Beräknad med $N_d = 40$.

3.3 Relativ positionering

Vid varje mätning av nya koner, som sker med en frekvens på 20 Hz, sparas alla relevanta (se figur 3.1) koner i en buffert som innehåller de filtrerade konernas positionsdata från den senaste samt föregående mätningen. Avståndet från LiDAR-sensorn till de närmaste konerna på vänster respektive höger sida jämförs med avståndet från förra mätningen enligt:

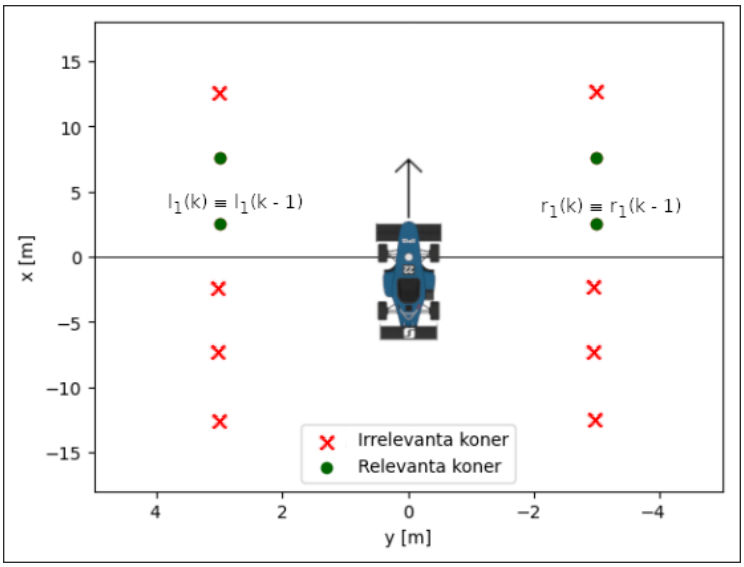
$$\Delta s_1 = \frac{(l_1(k-1) - l_1(k)) + (r_1(k-1) - r_1(k))}{2}, \tag{3.3}$$

där

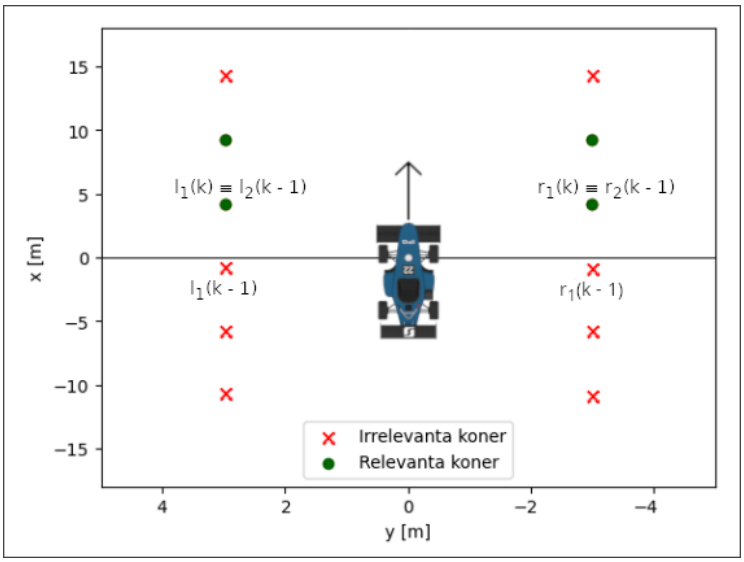
- Δs_1 är det relativa avståndet LiDAR-sensorn färdats mellan mätningarna,
- $l_1(k)$ och $l_1(l-1)$ är den närmaste konan åt vänster, nuvarande respektive föregående mätning, och
- $r_1(k)$ och $r_1(k-1)$ är den närmaste konan åt höger, nuvarande respektive föregående mätning.

Detta ger tre möjliga utfall:

1. De föregående närmsta relevanta konerna $l_1(k-1)$ samt $r_1(k-1)$, har förflyttats närmare sensorn och räknas fortfarande som relevanta i den senaste mätningen, som illustreras i figur 3.5. Detta ger att LiDAR-sensorns förflyttade avstånd mellan mätningarna är $\Delta s = \Delta s_1$.
2. De föregående närmaste relevanta konerna, $l_1(k-1)$ samt $r_1(k-1)$, har förflyttats bakom LiDAR-sensorn och räknas nu som irrelevanta, detta indikeras av att $\Delta s_1 < 0$ från ekvation 3.3. Då jämförs föregående mätnings näst närmsta koner med nuvarande mätnings närmsta koner enligt ekvation 3.4 och illustreras av figur 3.6. LiDAR-sensorns förflyttade avstånd mellan mätningarna ges av $\Delta s = \Delta s_2$ från ekvation 3.4.
3. Alla koner från föregående mätning $(k-1)$ har förflyttats bakom LiDAR-sensorn. Detta fall behandlas inte då det enbart är möjligt om LiDAR-sensorn förflyttar sig 6 meter mellan mätningarna (se figur 3.1), en hastighet motsvarande 120 m/s $\approx$ 432 km/h som anses vara orimligt högt under accelerationseventet [1].



Figur 3.5: Utfall 1, de närmsta konerna på höger respektive vänster sida identifieras som samma koner från föregående mätning med en uppdaterad relativ position.



Figur 3.6: Utfall 2, föregående mätnings närmsta konerna förflyttades bakom LiDAR-sensorn, nuvarande mätnings närmsta konerna identifieras som föregående mätnings näst närmsta koner.

$$\Delta s_2 = \frac{(l_2(k-1) - l_1(k)) + (r_2(k-1) - r_1(k))}{2}. \tag{3.4}$$

En beskrivning av ovanstående algoritm presenteras som pseudokod i Appendix A.2.

3.4 Markhastighetsestimering

Två metoder har evaluerats för att estimera markhastigheten från avståndsmätningarna. Den första uppskattar markhastigheten genom att försumma bilens acceleration för korta tidsintervall ($\Delta t = 50$ ms). Numerisk derivering ger då följande formel för hastighetsestimeringen v:

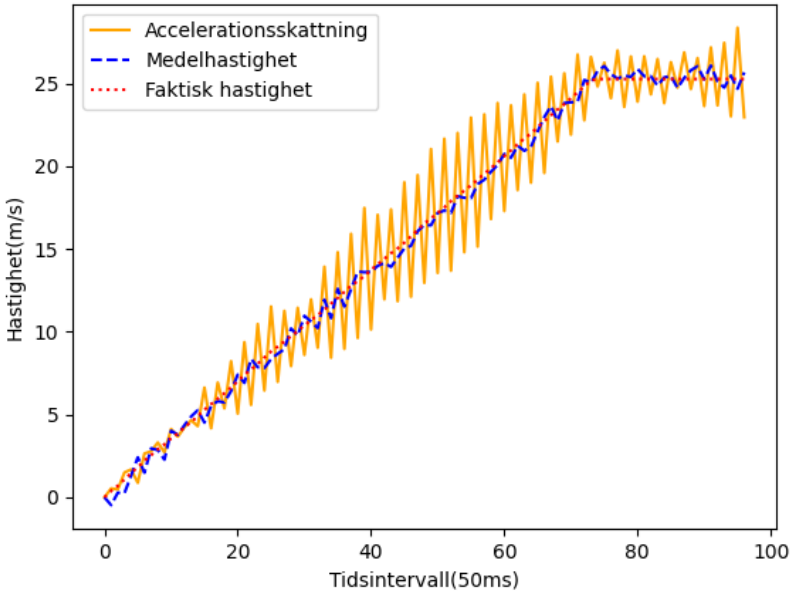
$$v = \frac{\Delta s}{\Delta t}. \tag{3.5}$$

I den andra metoden antas accelerationen vara konstant för korta tidsintervall och med hjälp av tidigare estimeringar av hastigheten beräknas accelerationen på följande sätt:

$$a = 2\frac{(\Delta s - v(k-1)\Delta t)}{\Delta t^2}. \tag{3.6}$$

Från detta kan hastigheten uppskattas med $v(k) = v(k-1) + a\Delta t$. Detta liknar arbete gjort av J. Zhang, W. Xiao, B. Coifman och J. P. Mills men i projektets fall är LiDAR:n i rörelse [6].

Hypotetiskt bör metoden som uppskattar acceleration fungera bättre då den tar hänsyn till att bilen accelererar under tidsintervallet. Dock är den första metoden enklare och då tidsintervallen är korta kan effekten från accelerationen vara relativt liten. Vid en acceleration på 8 m/s^2 motsvarar detta ett fel på 0.4 m/s. Dessa metoder valdes att evaluera då från LiDAR-mätningarna fås endast mätning av sträcka och då inte tillräckligt stora datamängder för att använda sig av prediktionsmodeller finns att tillgå. Båda metoder evalueras med hjälp av en `python`-simulering av ett accelerationsevent. I simuleringen är parametrar sm acceleration hämtade från data från tester genomförda av CFS25 [1]. Koden för `python`-simuleringen hittas i Appendix A.3.3.



Figur 3.7: Simulering av hastighetsestimeringsmetoder.

Figur 3.7 visar resultatet från en simulering av de två hastighetsuppskattningsmetoderna. Metoden då medelhastigheten beräknas i ekvation 3.5 fungerar bättre än metoden där acceleration estimeras i ekvation 3.6 när mätprecisionen från LiDAR-sensorn beaktas. Detta bekräftas av resultatet från upprepade simuleringar i tabell 3.1. Accelerationsestimeringsmetoden visade sig vara väldigt känslig för mätfel då den tenderade att börja självoscillera vid upprepade mätfel. Problemet med medelhastighetsmetoden var att den beräknar medelhastigheten istället för sluthastigheten för ett tidsintervall, men trots detta ger den bättre resultat än accelerationsestimeringsmetoden.

Tabell 3.1: Resultat för 1000 simuleringar

Metod	Genomsnittligt fel (m/s)	Varians för fel
Medelhastighet	0.417	0.000938
Accelerationsestimering	2.074	1.387

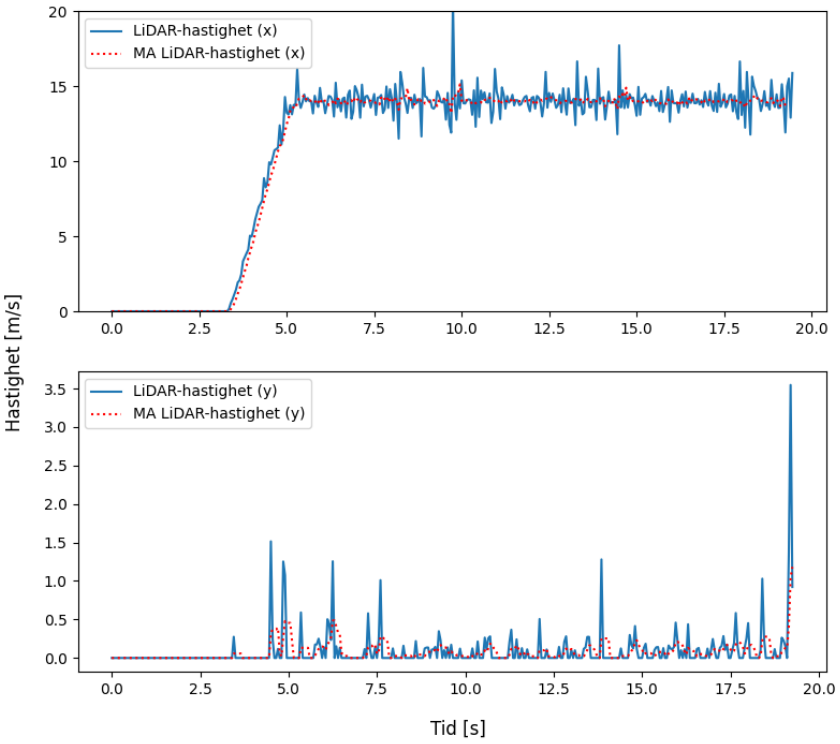
Tabell 3.1 visar att medelvärdesmetoden inte bara har ett mindre genomsnittligt fel, men även en liten varians för felet. Accelerationsestimeringen uteslöts därför som metod för hastighetsestimeringen. Det framgår också att medelvärdesmetoden påverkas relativt lite av mätfelen från LiDAR-sensorn då det genomsnittliga felet är väldigt nära det teoretiska felet på 0.4 m/s.

3.5 Hastighetsfiltrering

För att få en mindre brusig hastighetskurva så implementeras ett moving-average (MA) filter med en max-kapacitet på fem mätningar [23]. Detta ger en estimerad LiDAR-hastighet enligt:

$$v = \frac{1}{N} \cdot \sum_{k=0}^{N-1} b_k \tag{3.7}$$

där $N \in [1, 5]$, v är den filtrerade hastighetsestimeringen och b är bufferten med upp till de fem senaste hastighetsestimeringarna från LiDAR-sensorn.



Figur 3.8: Resultat av markhastighetsestimering från LiDAR jämfört med hjulhastighet

Resultatet av den estimerade markhastigheten från LiDAR-sensorn, med och utan filtrering i både under en riktig körning på ett accelerationsevent syns i figur 3.8.

4

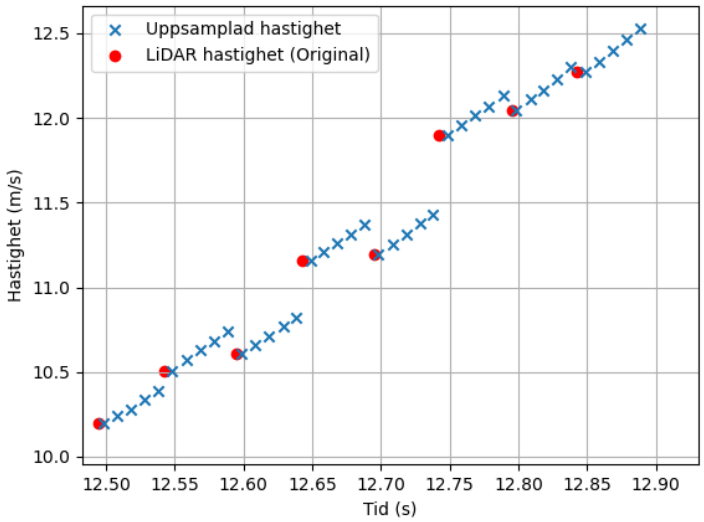
Sensorfusion

I sensorfusion-steget kombineras sensordata från de olika sensorerna från samma tidssteg [24]. Därefter kan nya tillstånd estimeras med hjälp av Kalmanfiltrering. I arbetet nyttjas hjulhastigheter, acceleration och LiDAR-hastigheten för att estimera bilens markhastighet och därmed glidfaktorn.

4.1 Uppsampling

För att få ett snabbt system används den högsta uppdateringsfrekvens 100 Hz hos Kalmanfiltret då det är den högsta mätfrekvensen från sensorerna. För att hantera de olika mätfrekvenserna uppsamplas LiDAR-odometrin som har en frekvens på 20 Hz till 100 Hz för att matcha IMU:ns mätfrkvens. Detta görs med hjälp av accelerationsdata som uppdaterar hastighetsestimeringen mellan LiDAR-mätningarna. Formeln för detta blir:

$$\hat{v}_{\text{uppsamplad}} = v + a_{\text{imu}} \cdot \Delta t$$



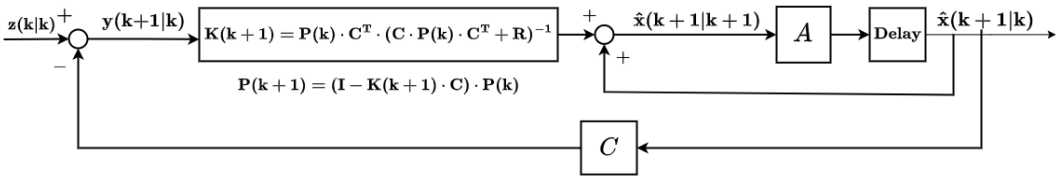
Figur 4.1: Illustration över hur uppsamplingsalgoritmen fungerar.

4.2 Kalmanfilter

För att få en optimal skattning av bilens markhastighet används ett Kalmanfilter för att kombinera de olika mätningar som påverkar bilens markhastighet[25]. Dessa mätningar är bland annat hastighetsestimering från LiDAR-data och accelerationsmätning från bilens IMU.
Kalmanfiltret fungerar på följande två steg [25]:

1. **Prediktion**:
   I detta steg används det tidigare tillståndet $\hat{x}(k|k)$ och en matris A som beskriver hur tillståndet förändras. Då erhålls en gissning över hur tillståndet borde vara.

2. **Uppdatering:**
   I detta steg jämförs prediktionen från det tidigare steget med mätningar från IMU:n och LiDAR:n. Kalmanfiltret justerar då gissningen om nästa tillstånd beroende på hur pålitlig modellen var från prediktionsteget och hur pålitlig mätningen är. Filtret väger ihop mätningen och modellen på ett optimalt sätt, där mätningen får mer vikt om modellen är osäker, och vice versa.



Figur 4.2: Blockschema över det implementerade Kalmanfiltret.

4.2.1 Modell

Kalmanfiltret bygger på antagandet av en linjär Gaussisk systemmodell [25]. Modellering av kalmanfiltrets tillståndsekvation innehåller ingen styrsignal, därmed ingen B matris. Anledningen för detta är antagandet om att estimeringen av markhastigheten skulle vara bättre då prediktionsteget sker oberoende av andra styrsignal. Dock har inte detta antagande verifierats genom att jämföra med ett kalmanfilter med påverkande på styrsignal. Filtret har dimensionerats enligt följande modell:

Tillståndsekvation (processmodell):

$$\hat{\boldsymbol{x}}(k+1|k) = A\hat{\boldsymbol{x}}(k|k) + \boldsymbol{w}(k).$$

Mätmodell (observationsmodell):

$$\hat{\boldsymbol{z}}(k) = C\hat{\boldsymbol{x}}(k+1|k) + \boldsymbol{v}(k),$$

där:

- $\boldsymbol{x}(k)$ är tillståndsvektorn vid tid k.
- A: Tillståndsövergångsmatris - beskriver hur systemet utvecklas.
- $\boldsymbol{z}(k|k)$ är mätvektorn.
- C: Observationsmatris.
- $\boldsymbol{w}(k) \sim \mathcal{N}(0, Q)$ är processbrus och Q är processbrusets kovariansmatris - osäkerheten i modell.
- $\boldsymbol{v}(k) \sim \mathcal{N}(0, R)$ är mätbrus och R är mätbrusets kovariansmatris - osäkerheten i sensor.

Kovariansmatriserna i filtret R och Q justerades i enlighet med kalmanfiltrets respons från tidigare inspelad data. P matrisen är ansatt till godtycklig startvärde och detta uppdateras i uppdateringssteget enligt figur 4.2. Filtret har modellerats till följande tillståndsvektor $\boldsymbol{x}$ och A-matris:

$$\boldsymbol{x} = \begin{bmatrix} v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \qquad A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Mätvektor $\boldsymbol{z}(k)$ och processbrusets kovariansmatris Q:

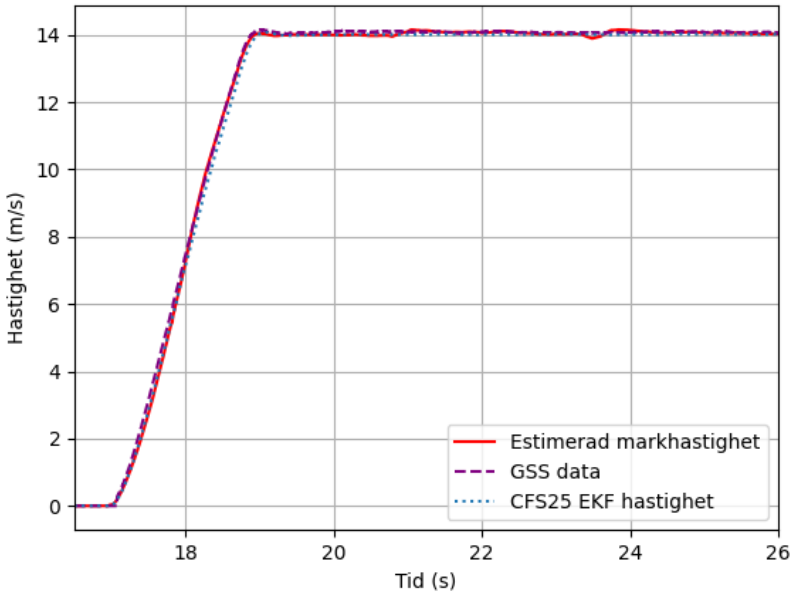
$$\boldsymbol{z} = \begin{bmatrix} v_x^{\text{LiDAR}} \\ v_y^{\text{LiDAR}} \\ a_x^{\text{IMU}} \\ a_y^{\text{IMU}} \end{bmatrix} \qquad Q = \begin{bmatrix} 0.008 & 0 & 0 & 0 \\ 0 & 0.008 & 0 & 0 \\ 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0.05 \end{bmatrix}$$

Mätbruskovariansmatris R, kovariansmatris P och mätmatrisen C:

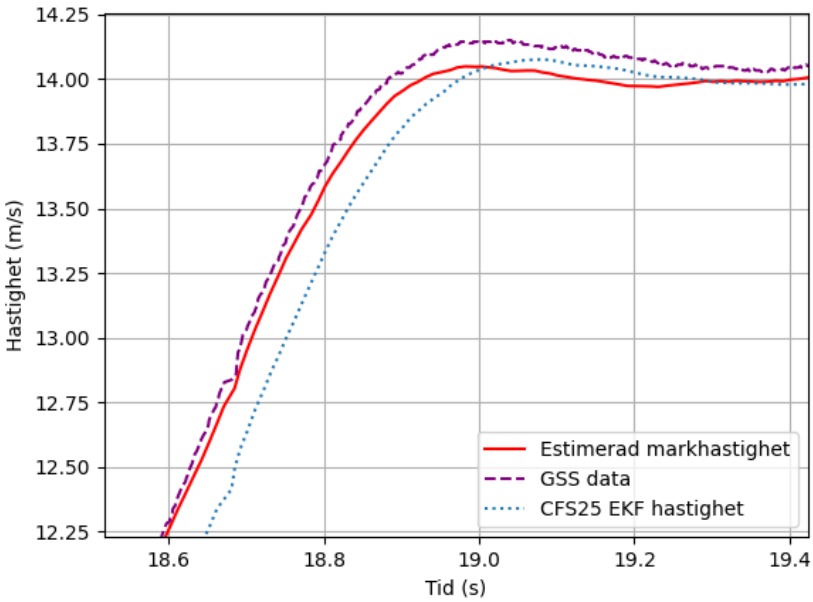
$$R = \begin{bmatrix} 150 & 0 & 0 & 0 \\ 0 & 150 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix} \qquad P = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.2.2 Resultat

Filtret har testats med data från tidigare körningar för att utvärdera responsen. I figur 4.3 visas den estimerade hastigheten jämfört med mätningar från en markhastighetssensor(GSS) och den nuvarande metoden med EKF.



(a) Hastigheterna under hela körningen.



(b) Inzoomad vy under slutet av accelerationsfasen.

Figur 4.3: Jämförelse av LiDAR-estimerad hastighet, GSS-data och CFS25:s EKF.

I tabellen 4.1 redovisas rotmedelkvadratavvikelsen (RMSE) och bias av hastihgetsestimering från det implementerade Kalmanfiltret (KF) och det existerande EKF. Dessa är jämförda med hastighetsmätningar v_{GSS} från en lånad markhastighetssensor (GSS) av av Sensoric Solutions [3].

$$e_{i,j} = \hat{v}_{i,j} - v_{GSS,j}, \quad i \in \{\text{KF,EKF}\},\, j = 1, \ldots, n,$$

$$\text{RMSE}_i = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(e_{i,j})^2},$$

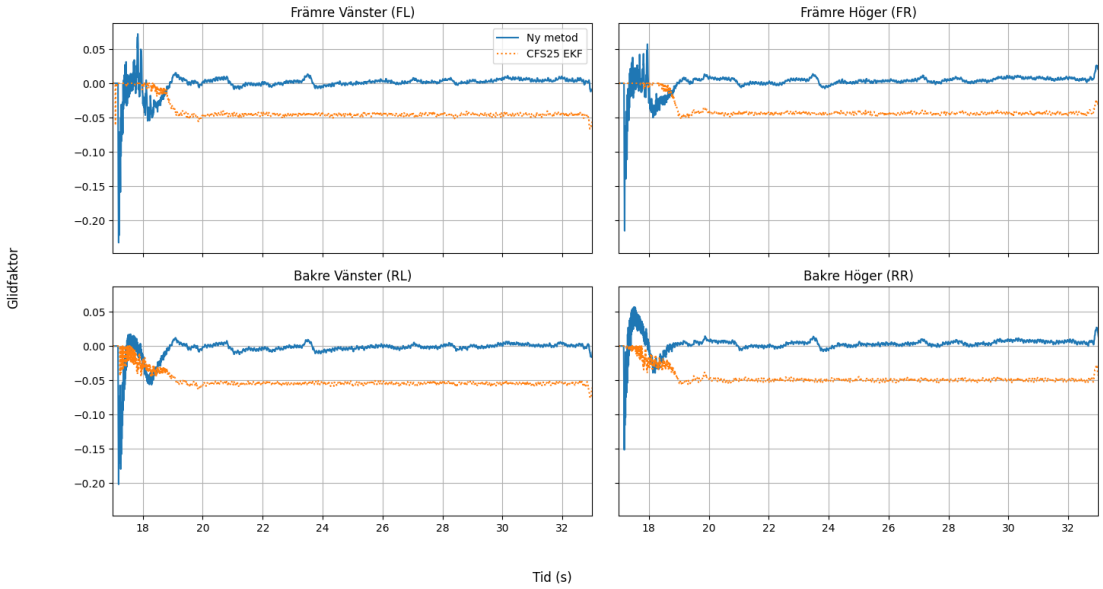
$$\text{Bias}_i = \frac{1}{n}\sum_{j=1}^{n} e_{i,j}.$$

	RMSE	Medelfel (bias)
KF	0.0628 m/s	–0.0192 m/s
EKF	0.0831 m/s	–0.0668 m/s

Tabell 4.1: Jämförelsetabell av det implementerade Kalmanfiltret och CFS25:s existerande EKF.

Kalmanfiltret som implementerades 4.2 visas ha ett lägre medelfel 4.1 jämfört med den redan implementerade EKF:en. En förbättring på $\approx 71.2\%$ i medelfel och $\approx 24.4\%$ förbättring i rotmedelkvadrat.

Efter markhastigheten har estimerats paras den ihop med hjulhastighetsdatan för att beräkna glidfaktorn ξ. I figur 4.4 visas glidfaktorn över hela körningen samt en jämförelse med glidfaktor-estiemringarna från CFS25 glidfaktorn. Ny metod i figuren legenden syftar på glidfaktorberäkning med ekvation 1.1 och med LiDAR-estimerad hastighet istället för CFS25 EKF v_x.



Figur 4.4: Jämförelse av den longitudinella glidfaktorn mellan CFS25 och den nya metoden.

I figur 4.4 syns det att värdena på glidfaktorn från CFS25:s EKF är negativa då bilen slutar accelerera i figur 4.3a. Detta är fel eftersom då bilen når konstant fart når den även kraftjämvikt vilket medför att den relativa hastigheten bör vara

$$\omega r - v \approx 0 \implies \xi \approx 0.$$

Om ett fordon har en glidfaktor $\xi \neq 0$ över tid är fordonet inte i kraftjämvikt.

4.2.3 Utvärdering av den nya metoden

Undersökning av graferna i 4.3b och resultaten i tabell 4.1 visar att både mätfelet och medelfelet av hastighetsestimeringen är mindre i den nya metoden. Figur 4.4 visar att glidfaktorn som beräknas med skattad hastighet från LiDAR-data och IMU-data är mer verklighetstrogen eftersom den relativa hastigheten, $v_{\text{rel}} = \omega r - v = 0$ då bilen slutar accelerera efter 19 s. Slutsatsen är att metoden för glidfaktorberäkningen borde ändras från CFS gamla metod till den nya under accelerationseventet.

Från analys av den nya metoden i figur 4.4 syns det att glidfaktorn ξ är brusig vilket har implikationer för styrsystem som baseras på metoden. Bruset kommer från hjulhastighetssensorn då hastigheten i figur 4.3 ej oscillerar.

5

Antispinnsystem

Glidfaktorn ξ mellan däck och väglag beräknas enligt ekvation 1.1. För att reglera glidfaktorn justeras vridmomentet från hjulmotorn. Reglering av detta kan göras på flera sätt. Ett av dem är att genom en regulator med proportionerligt värde till reglerfelet e, som benämns P-regulator. I systemet användes för varje hjulmotor och påverka varje däcks glidfaktor.

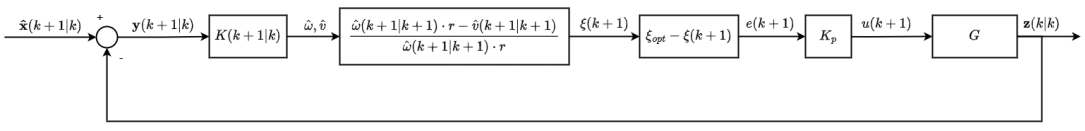
Metoden som valdes är att använda en proportionell regulator för varje hjulmotor. Det innebär att motorns vridmoment beräknas för varje enskilt däcks glidfaktor. P-regulatorn beräknar ett vridmoment u utefter reglerfelet e i ett slutet reglerschema [26].

Felet beräknas från den optimala glidfaktorn $\xi_{\text{opt}} = 0.13$. Beräkningarna för vridmomentet görs genom ekvation 5.1 och vridmomentet ansätts enligt ekvation 5.2.

$$e(k+1) = \xi(k+1) - \xi_{\text{opt}} \tag{5.1}$$

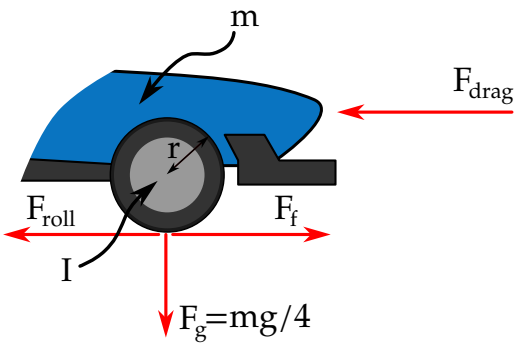
$$u(k+1) = K_p e(k+1) \tag{5.2}$$

Det proportionella värdet K_p styr hur mycket vridmomentet u ändras vid tidsteget k. Regleringen leder i sin tur till en accelerationkraft F_a på bilen och en ökning av hjulens vinkelhastigheter ω. Hjulhastighetssensorerna i bilen anger den aktuella hjulhastigheten ω vilket filtreras i CFS25:s EKF. Utsignalen från systemet G blir det sanna tillståndet i Kalmanfiltren och betecknas $\mathbf{z}$. Där beräknas felet från det skattade tillståndet i det tidigare tidssteget $\hat{\mathbf{x}}(k+1|k)$. Därefter appliceras Kalmanåterkopplingen enligt formlerna i ekvation 2.2. Det övergripande reglerschemat illustreras som ett blockschemafigur i 5.1.



Figur 5.1: Blockschema över glidregleringen.

Systemet G uppskattar hjulhastigheten och markhastigheten från dynamikformler för ett givet vridmoment som systemet får från regulatorn. En illustration på kraftsituationen visas i figur 5.2 [27]. Detta innebär att regleringen är MIMO (multiple input, multiple output) då både mark- och hjulhastigheten påverkar och påverkas av styrsignalen. Båda uppskattas genom den resulterande kraften mellan däck och väglag. Den accelererande kraften $F_a(k)$ drivs av vridmomentet u från hjulmotorn i bilen.



Figur 5.2: Friläggning av hjulet i fordonsdynamiken.

M_f betecknar friktionsmomentet, r betecknar hjulradien, m bilens massa, g gravitationskonstanen, A bilens area, och I är tröghetsmomentet. F_{roll} och F_{drag} är rörelseberoende resistiva krafter där C_r, C_d är rull- respektive luftmotståndkoefficienterna. Värden till alla dessa parametrar kan hittas i Appendix A.1. Fordonsmodellen för systemet G beskrivs av följande ekvationer:

$$\begin{aligned} M_f(k) &= F_f(k)r \\ \dot{\omega}(k) &= \frac{u(k) - M_f(k)}{I} \\ F_{\text{roll}} &= C_r mg \\ F_{\text{drag}}(k) &= C_d \frac{1}{2} \rho A v(k)^2 \end{aligned} \tag{5.3}$$

Det resulterande momentet bestäms i G enligt ekvation 5.3. Den resulterande kraften i ett däck $F_a(k)$ beräknas genom:

$$\begin{aligned} F_a(k) &= \frac{2u(k)m}{r(m + I/r^2)} - F_f(k) - F_{\text{roll}} - F_{\text{drag}}(k) \\ \dot{v}(k) &= \frac{F_a(k)}{m} \end{aligned} \tag{5.4}$$

Vinkelaccelerationen samt den longitudinella accelerationen integreras med Eulers

stegmetod för att hitta vinkelhastigheten ω och hastigheten v.

$$\begin{aligned} v(k+1) &= v(k) + \dot{v}(k) \\ \omega(k+1) &= \omega(k) + \dot{\omega}(k) \end{aligned} \tag{5.5}$$

Från de ovanstående formlerna framgår att systemet G beskrivs av de följande differentialekvationerna:

$$\begin{aligned} \dot{v}(k) &= \frac{2u(k)}{r(m + \frac{I}{r^2})} - \frac{F_f(k)}{m} - \frac{F_{\text{roll}}}{m} - \frac{F_{\text{drag}}(k)}{m} \\ \dot{\omega}(k) &= \frac{F_f(k)r - u(k)}{I} \end{aligned} \tag{5.6}$$

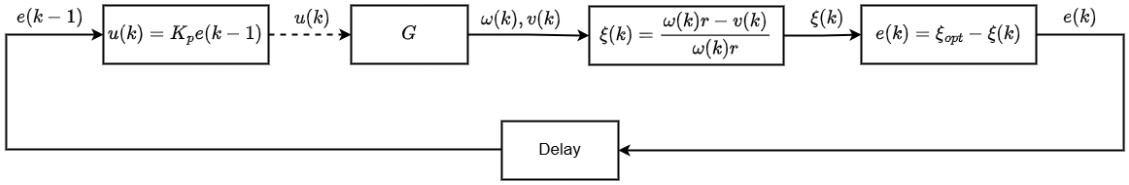
Med denna algoritm ansätts det vridmoment $u(k)$ som i sin tur leder till glidfaktor $\xi = \xi_{opt}$ och felvärdet $e(k) = 0$. Vridmomentet beräknas vid ett givet tidssteg k och begärs av hjulmotorn för nästa tidssteg $k + 1$. Figuren visar hur det begärda vridmomentet ändras efter felet e. Detta beräknas enligt ekvation 5.1. Glidfaktorn behöver enligt ekvation 1.1 en tröskel på hjulrotationshastighet ω_{rad} rad/s. Utan denna tröskel blir glidfaktorn $\xi = 1$ vid $\omega = v = 0$ eftersom gränsvärdet $\lim_{\omega \to 0} \xi = 1$. När detta sker fungerar inte systemet då det vill sänka vridmomentet när det redan är noll och därmed blir reglersystemet redundant.

5.1 Stabilitetsundersökningen genom simulering

Utvärdering av nya koncept testas först i förenklade miljöer där antalet parametrar som påverkar resultatet är kontrollerade. Reglersystemet som nyttjas är ett MIMO-system vilket gör att vanliga metoder för att undersöka stabilitet såsom Nyqvist-kriteriet eller Ruth-Hurwitz kriterium kräver antaganden kring konstant friktion för att bestämma överföringsfunktionen [28]. I en simulering kan styrsignalerna undersökas grafiskt för att utvärdera stabilitet för systemet. Därför utförs stabilitetsundersökningen genom ett enhetstest.

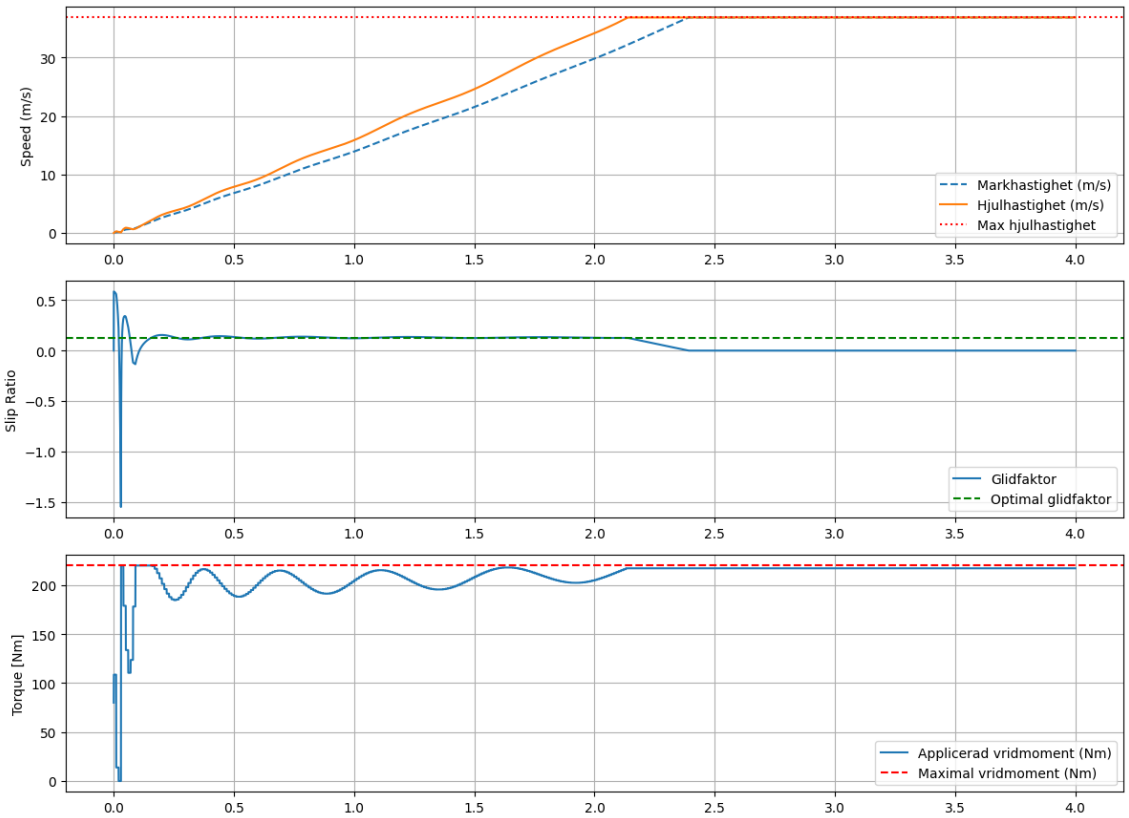
Testet körs enligt Formula Students regler för ett accelerationsevent och körning på raksträckan simuleras under 4.0 s. Bilmodellens statiska parametrar förenklas och det enda som undersöks är en hjulmotor med de fysiska egenskaperna hos bilen. De resterande hjulmotorer antas bete sig på samma sätt. Simuleringen görs i ett fall som liknar ett accelerationsevent utan sensorbrus. Under enhetstestning av antispinnsystemets regulator skapades ett skript för systemets fordonsdynamik som hittas i Appendix A.3.4. I simuleringen skickas en styrsignal $u(k)$ baserat på det tidigare reglerfelet e. Denna signal skickas till fordonsmodellen G beskrivet enligt ekvationer 5.6. Från systemet erhålls hjul- och markhastighet v, ω. Från detta beräknas glidfaktorn ξ enligt ekvation 1.1 och sedan beräknas felet e. Vid nästa iteration i

simuleringen $k+1$ skickas detta fel till början av simuleringsloopen där styrsignalen beräknas. Detta itereras i 100 Hz samtidigt som beräkningarna i systemet G utförs i 1000 Hz. Detta illustreras med den streckade pilen för styrsignalen, vilket uppdateras en tiondel av iterationerna. Simuleringen börjar med startparametrarna: $\xi(0), u(0), \omega(0) \approx 0$. Parametrar från hjulmotorerna införs enligt hjulmotorsbeskrivningen i 2.2 i rapporten. Blockschemat över simuleringen som utförs under ett enhetstest visas i figur 5.3.



Figur 5.3: Blockschema över beräkningarna vid varje steg i enhetstestet.

Accelerationen integreras i sin tur för att hitta hastigheten v och gildfaktorn ξ. Resultatet från simuleringen visas i figur 5.4.



Figur 5.4: Enhetstest med motordynamik och rörelseberoende resistiva krafter.

Från resultaten syns att vridmomentet u regleras för att minska felet e. Från detta enhetstest dras slutsatsen att regulatordesignen åtminstone håller i simulerade

miljöerna. Resultatet visar även att regleringen av styrsignalen sker stabilt utan stora oscillationer.

5.2 Simulering i CarMaker

CarMaker är ett bilsimuleringsprogram som användas för att utvärdera systemet och är industristandard [29]. En fristående version av programmet, CarMaker for Simulink användes då det har stöd för ROS2 med en toolbox vilket gör det möjligt att kontrollera simuleringen helt med ett ROS2-nätverk. I CarMaker finns även tillgång till sensoremulering av samtliga sensorer som CFS-bilen har. Programmet valdes för att simulera hela systemet, från LiDAR-detektion till vridmomentsuträkning och utskickandet av styrsignalen. Simuleringsmjukvaran tar även hänsyn till mer avancerade bilmodellsparametrar som däckmodell, aerodynamik och drivlina. En till skillnad mellan simulering i enhetstest och CarMaker är att laterala hastigheter införs och därmed ett behov av att styra bilen. Simuleringen görs fortfarande med avseende till accelerationsevent.

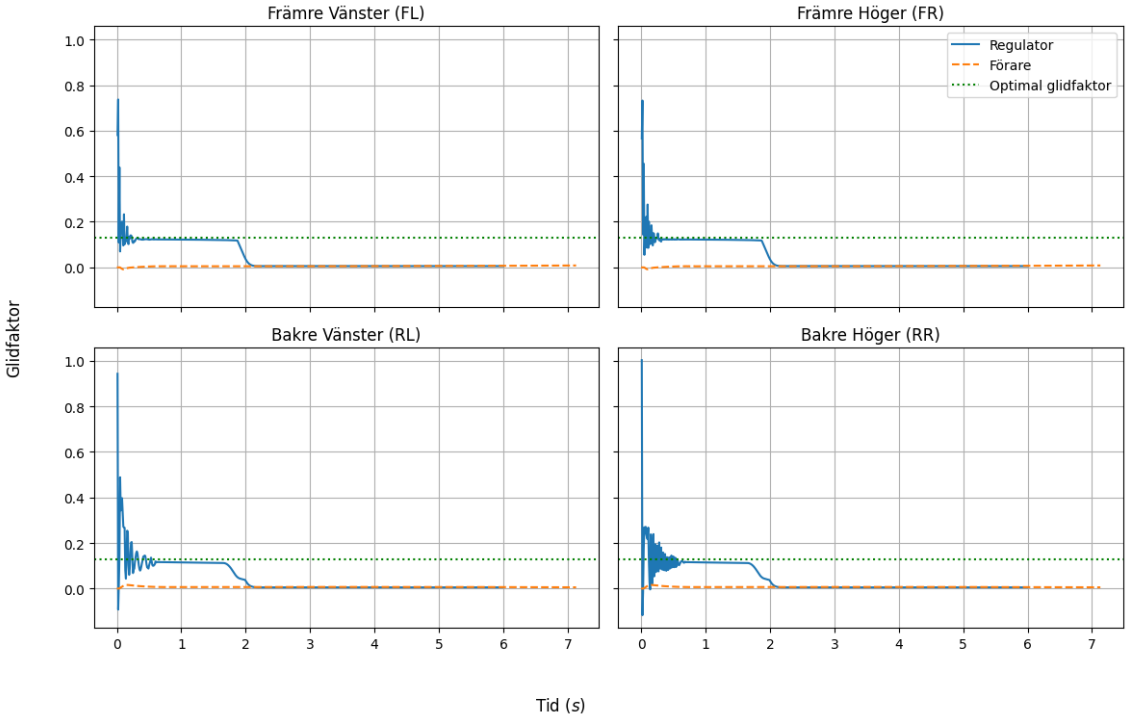
Metoden för att undersöka om reglersystemet gör någon skillnad i CarMaker bygger på CFS25-bilens parametrar i ett accelerationsevent enligt Formula Students regler [30]. För att utvärdera prestanda av antispinnsystemets koncept undersöktes glidfaktor, vridmoment i varje hjul, markhastigheten samt tiden det tar att nå mål. Eftersom LiDAR-sensorn inte kunde simuleras ersattes denna hastighetsmätning av CarMaker markhastighetssensorn. Glidfaktorn regleras på samma sätt som i enhetstestet (figur 5.1).

Resultaten för dessa simuleringar syns i tabell 5.1 vilket visar på en förbättring med reglersystemet jämfört med endast virtuell förare vilket överensstämmer med relaterade arbeten av KTH Formula student och IIT Delhi [9] [10]. I jämförelse med CFS24 var körtiden under ett autonomt accelerationsevent 3,848 s med en maxhastighet på 31,556 m/s [1].

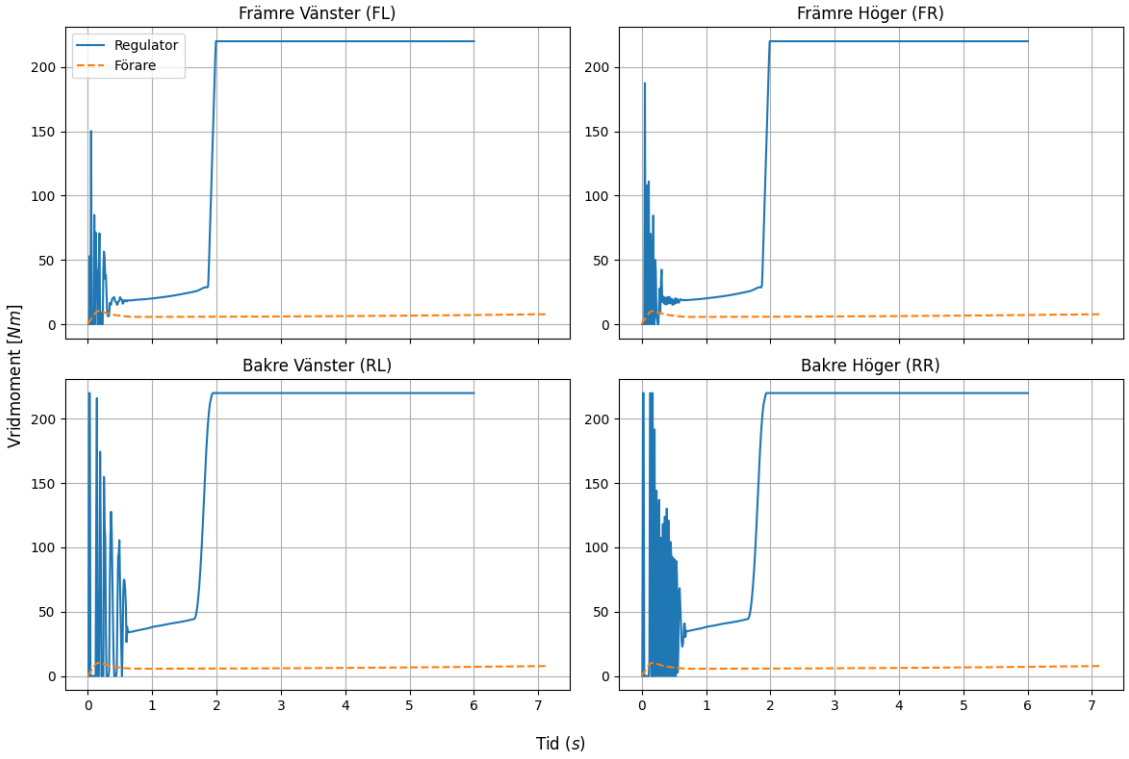
Tabell 5.1: Jämförelse av simulering av antispinnsystemet mot det autonoma systemet.

	Tid [s]	Maximal Hastighet [m/s]
Med Antispinnsystem	3.295	34.021
Virtuell Förare	7.131	21.167

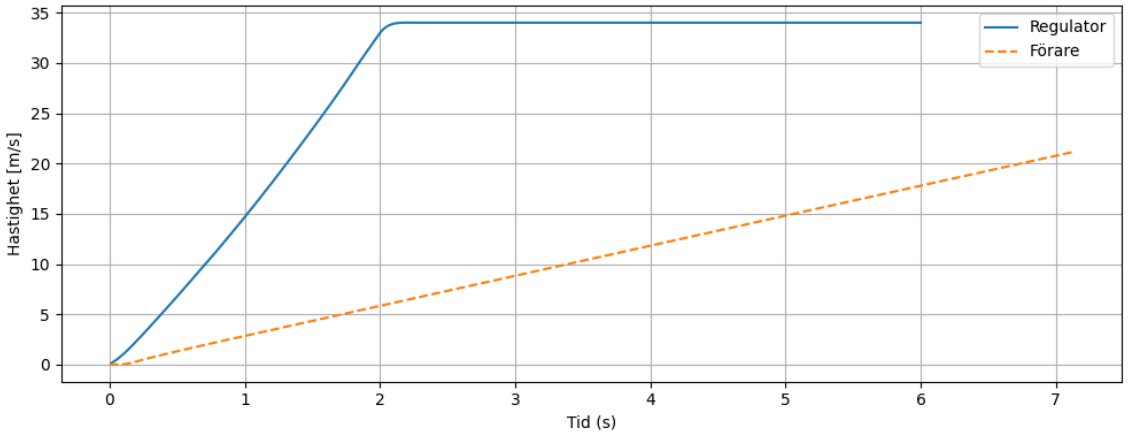
I figur 5.5 kan syns hur glidfaktorn på varje enskilt hjul när vridmomentet i figur 5.6 ändras. Hastigheten under köreventet syns i figur 5.7. Reglersystemet betecknas som regulator i figurerna nedan.



Figur 5.5: Jämförelse av glidfaktor ξ för varje däck under simuleringen. Den optimala glidfaktorn $\xi_{opt} = 0.13$ markeras som referens.



Figur 5.6: Jämförelse över det begärda vridmomentet i de olika hjulmotorerna för olika förare under simuleringen.



Figur 5.7: Hastighetsjämförelse mellan reglersystemet och den virtuella föraren.

5.2.1 Utvärdering av antispinnsystemet

Genom att analysera resultaten från `CarMaker` anses reglersystemet påverka accelerationen på ett positivt sätt. I jämförelse med den virtuella föraren och CFS24:s körningar bidrog systemet till en snabbare tid. Däremot syns det på både vridmoment och glidfaktor-resultaten att systemet agerar väldigt reaktivt med stora oscillationer. Figur 5.7 visar att hastigheten ökar stabilt vilket innebär att hjulhastigheten måste vara den instabila faktorn. I figur 5.5 syns att det bildas ett kvarstående fel i P-regleringen innan maximal hastighet nås. En integrerande del i regulatorn hade motverkat detta genom att öka noggrannheten. För att motverka den redan höga instabiliteten måste en derivatadel introduceras till regulatorn. En PID-regulator skulle i sin tur minska oscillationer, öka fasmarginalen och motverka bildandet av statiska fel. Från resultatet i sensorfusions-delen syns det i figur **??** att det finns signifikant brus vilket innebär att en PID-kontroller hade passat alla scenarion utan att riskera instabilitet. Detta kan inte säkerställas då simuleringarna inte har utförts över ett intervall av värden på K_p.

6

Helbilssimulering och diskussion

I denna delen av rapporten diskuteras arbetet i sin helhet, framförallt den tänkta sammansättningen av de tidigare delarna i `CarMaker` och vidare utvecklingspotential.

6.1 Helbilssimulering i `CarMaker`

Under arbetet simulerades inte hela antispinn-systemets alla tre delar samtidigt. Detta eftersom `CarMaker` kräver 'Ray Tracing' och NVIDIA-hårdvara [31]. Under hela projektets gång har gruppen jobbat med att få igång simuleringsmjukvaran med `ROS`-nätverket. Simuleringar har kunnat göras virtuellt med `ROS`-integration i `CarMaker` med antispinn-reglersystemet, men då har en markhastighetssensor använts istället för LiDAR-odometrin. Gruppen har haft två möten med IPG Sverige och CFS för att integrerat programmet, men inte lyckats. Under ett handledningsmöte med IPG Sverige visades ett felmeddelande som inte fanns i deras databas. Under sista instansen så avinstallerades grafikkortspaketen på en medlems dator och ominstallerades utan någon skillnad. Under projektets gång har över 100 timmar lagts på att få igång simuleringsverktyget utan framgång inom LiDAR-simulering.

För att LiDAR:n ska simuleras krävs också att den virtuella sensorn är kalibrerad på samma sätt som HESAI-sensorn som används. HESAI Pandar 64 sensorn kommer med specifika drivrutiner. Dessa har inte heller kunnat integreras med simuleringsverktyget för att få samma datatyper virtuellt som fysiskt.

6.2 Diskussion

Rapporten har behandlat framtagandet av ett antispinnsystem som är baserat på fordonshastighet v estimerad från sekvenser av LiDAR-punktmoln, IMU- och hjulhastighetssensormätningar. Utifrån dessa beräknades den longitudinella glidfaktorn ξ för att reglera vridmoment och hålla den optimala glidfaktorn $\xi_{\text{opt}} = 0.13$.

Algoritmen som tillämpas för att beräkna hastigheten bygger på att jämföra punktmoln från tidigare mätningar. LiDAR-hastigheten beräknas med en frekvens på 20 Hz och genom att kombinera detta med IMU-acceleration hittas en uppsamplad hastighet i 100 Hz med Kalmanfiltrering. Figur 4.3b visar att den estimerade hastigheten är väldigt nära den faktiska markhastigheten under accelerationsfasen. För att möjliggöra en effektiv vridmomentsreglering krävs att hastighetsuppmätningarna, för hjul och fordon, är precisa. Från figur 4.4 syns det att den nya metoden för att beräkna glidfaktor ξ är mer sanningsenlig än den gamla och borde ersätta CFS25 glidfaktorberäkningen under accelerationseventet. Detta fastslås genom det kvarstående felet som bildas i CFS25:s EKF-beräkning av glidfaktorn. I samma figur syns det kraftiga oscillationer i glidfaktorn vilket påverkar reglersystemet. Brusig data leder till oscillerande vridmomentsbegäran vilket skapar ett instabilt system. Överdrivet varierande vridmoment kan ha två negativa följder:

- Ökad säkerhetsrisk då bilen agerar reaktivt eftersom vridmomentsbegäran ändras kraftigt mellan tidsstegen. Detta kan leda till att bilen kör av banan eller till andra oförutsägbara följder.
- Dålig reglering av glidfaktorn leder till att acceleration blir suboptimal och bilens prestanda blir sämre. Detta leder i sin tur till sämre resultat i tävlingen.

Dessa problem kan åtgärdas genom att justera Kalmanfiltret för att motverka bruset, men det kan även leda till att systemet blir överberoende på IMU-accelerationen. Ett instabilt reglersystem kan åtgärdas med en större derivatadel i en PID-regulator. Derivatadelen motverkar oscillationer i feltermen. En annan lösning som åtgärdar instabilitet är att beräkna vridmomentet på ett sätt som tar hänsyn till fler variabler än mark- och hjulhastigheten. Denna metod kan minska säkerhetsrisken av brusig hjulhastighetsdata. För att nå hög prestanda av ett antispinnsystem krävs precis uppmätning av markhastigheten och därmed finns mer effektiva metoder än LiDAR-uppmätt hastighet. För mer djupgående slutsatser krävs att hela systemet simuleras och testas med antispinn-reglering.

7

Slutsats och vidare arbete

I detta kapitel avslutas rapporten med att ge förslag för vidare arbete och eventuella förbättringar.

Slutsatserna som dras från detta arbete är att det finns effektivare sätt att skapa antispinnsystem än med LiDAR-odometrin från systemet som presenteras i rapporten.

Arbetet genomfördes under ett kandidatarbete utan tillgång till hårdvaran. Nästa steg i arbetet är att testa de olika systemen i verkligheten och undersöka deras effektivitet. Endast efter att systemets funktion har testats i verkligheten kan man justera systemparametrarna.

Följande områden kan förbättras eller vidare arbetas med för att skapa ett system som kan köras under alla Formula Students körevent:

- Inom området av hastighetsestimering kan algoritmen för att beräkna markhastigheten utökas till att inkludera andra deltävlingar än accelerationseventet. Den implementerade algoritmen bygger på att det alltid finns minst fyra koner, två till vänster och två till höger, framför LiDAR-sensorn. I svängar med liten radie kan det uppstå situationer då algoritmen inte hittar koner på ena sidan och då kan en hastighet inte beräknas. Algoritmen behöver alltså anpassas till att hitta statiska punkter på annat sätt i en sådan situation. Andra hastighetsestimeringsalgortimer, som prediktionsmodeller, kan också undersökas men dessa behöver datamängder som inte fanns tillgänglig vid arbetes utförande [32].
- Ett vidare arbete är att utvärdera om ett sammanslaget utökat Kalmanfilter (EKF) hade skattat hastigheten mer sanningsenligt än ett hastighetsbaserat Kalmanfilter. Detta innebär att lägga till en sanntillståndsuppmätning i EKF:en för hastighet i x- och y-led och utföra samma tester.
- Ett annat vidare arbete är att utveckla Kalmanfiltret från metoden. Det hade inneburit att komplettera tillståndsekvationen med en styrsignal $u(k)$ och fordonsmodellen G, som beskrivs i kapitlet om antispinnsystemet 5.
- Undersöka om en mer avancerad regulator eller optimeringsalgoritm är bättre

än den klassiska P-regulatorn som endast reglerar vridmomentet efter glidfaktor. Ett arbete kring modellering av ett optimeringsproblem som reglerar mot optimal glidfaktor och minimerar faktorer som girning, v_y och möjligtvis roll och pitch.

- RANSAC-parametrar kan optimeras såsom till exempel antalet iterationer som tillåts, eller vilka avvikelser från den kända kongeometrin som godkänns. Detta möjliggör att kunna dra nytta av konanpassning och utföra detta på ett pålitligt sätt vilket kan förbättra konpositioneringen.

- Utfall 3 som berörs i den relativa positioneringen i kapitel 3.3 i relation till väldigt hög markhastighet hade även kunnat uteslutas genom att en LiDAR-sensor med högre uppdateringsfrekvens använts. Detta hade gjort systemet mer flexibelt för höga hastigheter som ännu inte uppnås, men kan eventuellt uppnås i framtida Formula Student-bilar. Ett annat alternativ är att undersöka om en solid-state LiDAR med högre uppdateringsfrekvens är bättre lämpad för ändamålet.

Litteratur

[1] Proprietary Team Data Sheet Chalmers Formula Student 2025,

[2] Formula Student Germany. "FSG Rules." Hämtad: 2025-05-09. (2025), URL: https://www.formulastudent.de/fsg/rules.

[3] Sensoric Solutions. "OMS Race – 2-Axis Sensor for Motorsport." Hämtad: 2025-05-10. (2025), URL: https://www.sensoric-solutions.com/products/sensors/oms-race-biaxial-sensor-for-motorsport/.

[4] N. A. de Almeida Salgueiro, "Torque Vectoring Control of an Electric Vehicle with In-Wheel Motors," examensarb., Universidade de Lisboa, 2021, s. 4–18.

[5] C. V. Manna R Gonzalez D, "Control Challenges for High-Speed Autonomous Racing: Analysis and Simulated Experiments," i *SAE International Journal of Connected and Automated Vehicles-V131-12EJ*, 2022.

[6] B. C. J. Zhang W. Xiao och J. P. Mills, "Vehicle Tracking and Speed Estimation From Roadside Lidar," i *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 13*, 2020.

[7] J. L. David J. Yoon Keenan Burnett och T. D. Barfoot, "Need for Speed: Fast Correspondence-Free Lidar-Inertial Odometry Using Doppler Velocity," i *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[8] L. P. Guan H Wang B, "Identification of maximum road friction coefficient and optimal slip ratio based on road type recognition," i *Chinese Journal of Mechanical Engineering, 27(5)*, 2014.

[9] F. Colin, "Traction Control for KTH Formula Student," i *Dissertation*, 2020.

[10] A. Biswas och R. Yadav, "Modeling and Simulation of Launch Control System for Formula Student Electric Vehicle," i *IEEE Transportation Electrification Conference (ITEC-India)*, 2021.

[11] HESAI Technologies, *Pandar64 User Manual*, https://www.hesaitech.com/wp-content/uploads/2025/04/Pandar64_User_Manual_640-en-250410.pdf, Hämtad: 2025-05-11, 2025.

[12] Point Cloud Library (PCL), *Cluster Extraction*, Hämtad: Mar. 6, 2025, 2025. URL: https://pcl.readthedocs.io/projects/tutorials/en/master/cluster_extraction.html.

[13] M. Ester, H.-P. Kriegel, J. Sander och X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," i *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, Portland, OR, USA: AAAI Press, 1996, s. 226–231.

[14] Honeywell International Inc., `https://aerospace.honeywell.com/us/en/products-and-services/products/navigation-and-sensors/navigation-systems/hg4930-mems-inertial-measurement-unit`, Hämtad: 2025-05-12.

[15] Dr. Johannes Heidenhain GmbH, `https://www.heidenhain.com/fileadmin/pdf/en/01_Products/Produktinformationen/PI_ECI1119_EQI1131_FS_ID1246793_en.pdf`, Hämtad: 2025-05-12.

[16] NVIDIA, `https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/index.html`, Hämtad: 2025-05-12.

[17] ROS - Robot Operating System, `https://www.ros.org/`, Hämtad: 2025-05-09.

[18] L. A. N. C. Lopes, "SLAM Method for the Formula Student Driverless Competition," examensarb., TÉCNICO LISBOA, 2021.

[19] R. Sedgewick och K. Wayne, *Algorithms*, 4th. Boston, MA 02116: Addison-Wesley, 2011, pp. 432–447, ISBN: 978-0-321-57351-3.

[20] ISO/IEC JTC1/SC22/WG21. "Working Draft, Standard for Programming Language C++, Section 23.4.6: std::set." Hämtad: 2025-03-06. (2025).

[21] R. K. R. Schnabel R. Wahl, "Efficient RANSAC for Point-Cloud Shape Detection," i *Computer Graphics Forum: Volume 26, Issue 2*, 2007.

[22] C. P. Robert och G. Casella, *Monte Carlo Statistical Methods*, 2. utg. Springer, 2004, Second edition, ISBN: 978-0-387-21239-7. URL: `https://doi.org/10.1007/978-1-4757-4145-2`.

[23] T. Verbeure, *Moving Average and CIC Filters*, `https://tomverbeure.github.io/2020/09/30/Moving-Average-and-CIC-Filters.html`, Hämtad: 2023-10-01, 2020.

[24] H. B. Mitchell, "Sensor and Data Fusion Concepts and Applications," i *Proceedings of SPIE*, vol. 7692, SPIE, 2010.

[25] A. Becker, *Kalman Filter from the Ground Up*. Scribd, 2017.

[26] L. L. Torkel Glad, *Control Theory and Multivariable and Nonlinear Models*. Taylor & Francis, 2000.

[27] K. J. Å. C. Canudas de Wit H. Olsson, "A New Model for Control of Systems with Friction," i *IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 40, NO. 3, MARCH 1995*, 1995.

[28] B. Lennartsson, *Reglerteknikens grunder*. Studentlitteratur, 2002.

[29] IPG CarMaker. Hämtad: 2025-05-09. (2025), URL: `https://www.ipg-automotive.com/en/products-solutions/software/carmaker/`.

[30] *Formula Student Rules 2024. Section D.5.1*, `https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.1.pdf`, Hämtad: 2025-04-08.

[31] IPG CarMaker, *User Guide*, 2025.

[32] Q. B. Zhang L. Liu W., "Combined Prediction for Vehicle Speed with Fixed Route," i *Chinese Journal of Mechanical Engineering volume 33*, 2020.

A

Appendix

A.1 Bilparametrar

De bilparametrar som används i simuleringsmiljöer för att skapa CFS25-bilmodellen.

Tabell A.1: CFS25 bilparametrar

Variabel	**Värde**	**Beskrivning [Enhet]**
ξ_{opt}	0.13	Önskad glidfaktor [-]
u_{max}	220	Maximalt vridmoment [Nm]
m	207	Fordonsmassa (kvartfordon) [kg]
r	0.2286	Hjulradie [m]
J	2.84	Hjulens tröghetsmoment [kg·m^2]
C_{rr}	0.010	Rullmotståndskoefficient [-]
C_d	0.375	Luftmotståndskoefficient [-]
C_l	0.9225	Lyftkraftskoefficient [-]
A	0.29	Frontarea [m^2]
RPM_{max}	20 000	Maximal RPM
P_{max}	80 000	Maximal Effekt [W]

A.2 Markhastighetsestimering

```
// Calculates the center of a cone
function center(Cone cone) {
    return sum(cone.positions) / cone.position.length
}

// Verifies whether a cone position is in view
function in_view(Point p, Direction dir) {
    if (dir == Left && p.x > 0 && p.y > 0) {
```

```
9           return true
10      }
11      else if (dir == Right && p.x > 0 && p.y < 0) {
12          return false
13      }
14
15      return false
16  }
17
18  function filter(Cones cones, Direction dir) {
19      Set filtered; // red black tree ordered by euclidian distance
20      for (cone in cones) {
21          position = center(cone);
22          if (in_view(position, dir) && position.x > 1) {
23              filtered.add(position);
24          }
25      }
26
27      return filtered[0], filtered[1];
28  }
```

Listing A.1: Pseudokod

```
function relative_distance(cones, t, dt) {
    // The closest cone on the left for this and the previous
   measurement
    l1_old = cones.first_left[t - dt]
    l1_new = cones.first_left[t]

    // The closest cone on the right for this and the previous
   measurement
    r1_old = cones.first_right[t - dt]
    r1_new = cones.first_right[t]

    distance = ((l1_old - l1_new) + (r1_old - r1_new)) / 2

    // Handle 2nd case, closest cones are now invalid
    if distance < 0 {
        // The 2nd closest cone on the left for this and the
   previous measurement
        l2_old = cones.second_left[t - dt]
        l2_new = cones.second_left[t]

       // The 2nd closest cone on the right for this and the
   previous measurement
        r2_old = cones.second_right[t - dt]
        r2_new = cones.second_right[t]

        distance = ((l2_old - l2_new) + (r2_old - r2_new)) / 2
    }

    return distance
}
```

Listing A.2: Pseudokod för relativ distansförflyttning av LiDAR

A.3 Enhetstester

Återskappandet av enhetstester kan göras genom att tillämpa en styrsignal som ett vridmomentsbegäran i fordonsdynamiksimuleringen som görs i `python` nedan. Alla enhetstester har bifogats för att läsaren ska kunna regenerera undersökningen och utvärdera resultaten.

A.3.1 RANSAC simulering

koden i `C++` kompileras med kommandot

```
g++ fitting.cpp -std=c++17 -o fitting -I/usr/include/pcl-1.15
-I/usr/include/eigen3 -I/usr/include/vtk -lpcl_common -lpcl_search
-lpcl_features -lpcl_sample_consensus -L/usr/lib -lvtkCommonCore
```

-lvtkIOCore -lvtkFiltersCore -lvtkRenderingCore -lvtkRenderingOpenGL2 -lvtkInteractionStyle -lvtkRenderingFreeType

```
#include <pcl/features/normal_3d.h>
#include <pcl/sample_consensus/sac_model_cone.h>
#include <pcl/sample_consensus/ransac.h>

#include <fstream>
#include <sstream>
#include <iostream>
#include <cmath>

// Function to load point cloud from a CSV file
pcl::PointCloud<pcl::PointXYZ>::Ptr loadCSV(const std::string& filename) {
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Could not open file " << filename << std::endl;
        return cloud;
    }

    std::string line;
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        float x, y, z;
        char comma;
        ss >> x >> comma >> y >> comma >> z;
        cloud->points.emplace_back(x, y, z);
    }

    file.close();
    return cloud;
}

int main() {
    // Load point cloud from CSV
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud = loadCSV("cone_test_data.csv");

    if (cloud->empty()) {
        std::cerr << "No points loaded!" << std::endl;
        return -1;
    }

    // Create a normals object to be used with the cone model
    pcl::PointCloud<pcl::Normal>::Ptr normals(new pcl::PointCloud<pcl::Normal>);

    // Estimate normals (you can use a normal estimation method, e.g., pcl::NormalEstimation)
    pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
    ne.setInputCloud(cloud);
    pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new pcl::search::
```

```
    KdTree<pcl::PointXYZ>);
48   ne.setSearchMethod(tree);
49   ne.setRadiusSearch(0.05);  // Adjust based on point density
50   ne.compute(*normals);
51
52   // Check if normals are computed successfully
53   if (normals->empty()) {
54       std::cerr << "Failed to compute normals!" << std::endl;
55       return -1;
56   }
57
58   // Define the cone model (with both point and normal types)
59   pcl::SampleConsensusModelCone<pcl::PointXYZ, pcl::Normal>::Ptr
    model(
60       new pcl::SampleConsensusModelCone<pcl::PointXYZ, pcl::
    Normal>(cloud));
61
62   // Set the normals for the model
63   model->setInputNormals(normals);
64
65
66   // Force the axis to the z-axis (0, 0, 1)
67   Eigen::Vector3f axis(0, 0, -1); // z-axis
68   model->setAxis(axis);  // This sets the axis of the cone to be
    aligned with the z-axis
69   model->setEpsAngle(10*M_PI/180);
70   //model->setMinMaxOpeningAngle((24.5-5)*M_PI/180, (24.5+5)*M_PI
    /180);
71   //model->setMinMaxOpeningAngle(0.49, 0.49);
72
73
74   // RANSAC fitting
75   pcl::RandomSampleConsensus<pcl::PointXYZ> ransac(model);
76   ransac.setDistanceThreshold(0.001);  // Adjust threshold based
    on noise
77   //ransac.setMaxIterations(1000);
78   ransac.computeModel();
79
80   // Check if RANSAC found a model
81   std::vector<int> inliers;  // To store inlier indices
82   ransac.getInliers(inliers);
83
84   if (inliers.empty()) {
85       std::cerr << "RANSAC could not find a model!" << std::endl;
86       return -1;
87   }
88
89   // Get the cone parameters
90   Eigen::VectorXf coefficients;
91   ransac.getModelCoefficients(coefficients);
92
93   // Extract parameters
94   Eigen::Vector3f apex(coefficients[0], coefficients[1],
    coefficients[2]); // Apex position
95   Eigen::Vector3f newaxis(coefficients[3], coefficients[4],
    coefficients[5]); // Axis direction
```

```
    float angle = coefficients[6]; // Opening angle (in radians)

    // Print results
    //std::cout << "Apex: " << apex.transpose() << std::endl;
    //std::cout << "Axis: " << newaxis.transpose() << std::endl;
    //std::cout << "Opening angle: " << angle * 180.0 / M_PI << "
    degrees" << std::endl;

    std::cout << coefficients[0] << ";" << coefficients[1];

    return 0;
}
```

Listing A.3: Simulering för RANSAC

Detta kombineras sedan med `python` scriptet nedan för att evaluera felet, standardavvikelsen på felet och medeltalet misslyckade anpassningar.

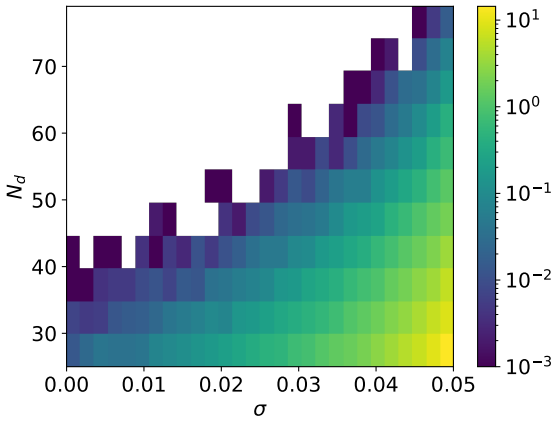
```
import numpy as np
from random import uniform
import subprocess

x0 = 0
y0 = 0
z0 = 0
R = 0.13
h = 0.3

def err_test(runs=10, Nd=50, sigma=0):
    errs = []

    avg_fails = []
    for _ in range(0,runs):
        fails = 0
        while True:
            data = []

            for i in range(0,Nd):
                while True:
                    x = uniform(x0-R, x0+R)
                    y = uniform(y0-R, y0)
                    z = (1-np.sqrt((x-x0)**2 + (y-y0)**2)/R)*h + z0

                    if np.sqrt((x-x0)**2 + (y-y0)**2) < R:
                        x += np.random.normal(0, sigma)
                        y += np.random.normal(0, sigma)
                        z += np.random.normal(0, sigma)

                        data.append(np.array([x, y, z]))
                        break

            df = pd.DataFrame(data, columns=["x", "y", "z"])
            df.to_csv("cone_test_data.csv", index=False)

            try:
```

```
38                  p = subprocess.Popen("./fitting", stdout=subprocess
    .PIPE)
39                  result, _ = p.communicate()
40                  result = result.decode("utf-8")
41
42                  x_res = float(result[:result.find(';')])
43                  y_res = float(result[result.find(';')+1:])
44
45                  err = np.sqrt((x_res-x0)**2 + (y_res-x0)**2)
46                  errs.append(err)
47                  break
48              except:
49                  fails+=1
50                  print('RANSAC failed!')
51                  pass
52
53          avg_fails.append(fails)
54
55      np.save(f'data/data{str(Nd)}-{str(sigma)}-{runs}', [np.mean(
    errs), np.std(errs), np.mean(avg_fails)])
```

Listing A.4: Simulering för RANSAC

A.3.2 RANSAC pålitlighet

Om punktklustret är för brusigt eller om det innehåller för få punkter kan RANSAC helt misslyckas med att anpassa konmodellen till ytan. Medelvärdet för antalet misslyckade anpassningar av RANSAC för de 1000 simuleringarna för varje punkt syns i figur A.1. Den är genererad på samma sätt som figur 3.3 och 3.4. Vit färg innebär att inga felkörningar påträffades.



Figur A.1: Misslyckade anpassningar för olika N_d och σ.

A.3.3 Hastighetsestimeringsalgoritmer

```
import numpy as np
from findCones import *
from updateState import *
from speedEstimation import *

cones = [[-1.5,0],[1.5,0],[-1.5,5],[1.5,5],
        [-1.5,10],[1.5,10],[-1.5,15],[1.5,15],
        [-1.5,20],[1.5,20],[-1.5,25],[1.5,25],
        [-1.5,30],[1.5,30],[-1.5,35],[1.5,35],
        [-1.5,40],[1.5,40],[-1.5,45],[1.5,45],
        [-1.5,50],[1.5,50],[-1.5,55],[1.5,55],
        [-1.5,60],[1.5,60],[-1.5,65],[1.5,65],
        [-1.5,70],[1.5,70],[-1.5,75],[1.5,75],
        [-1.5,80],[1.5,80],[-1.5,85],[1.5,85]]

bigErr = []
n=1000
for i in range(1,n):

    carPos = (0,0)
    carSpeed = (0,0)
    finnished = False
    accel = 6.83
    maxSpeed=25.27
    frequency = 20

    newCones = findCones(carPos,cones)
    oldCones = 0

    speedEst = [[0,0]]

    err = 0
    counter = 0

    while finnished == False:
        oldCones = newCones
        carSpeed, carPos = updateState(accel,carSpeed,carPos,
   maxSpeed,frequency)
        newCones = findCones(carPos,cones)
        speedEst.append(accelSpeedEstimation(newCones,oldCones,
   frequency,speedEst[-1]))
        if carSpeed[1]<25:
            err += abs(speedEst[-1][1]-carSpeed[1])
            counter += 1

        if carPos[1]>75:
            finnished=True

    bigErr.append(err/counter)

    print(f"Average Error: {err/counter}")

print(f"average error for {n} runs: {sum(bigErr)/len(bigErr)}")
print(f"Variance for {n} runs: {np.var(bigErr)}")
```

Listing A.5: Simulering av hastighetsestimeringsalgoritmer

```
import numpy as np

def findCones(carPos,cones,r=20):
    foundCones = []
    for i in cones:
        if i[1]>carPos[1]:
            dist = np.linalg.norm(np.subtract(i,carPos))
            if (dist<=r):
                foundCones.append(np.subtract(i,carPos))
    return foundCones
```

Listing A.6: Funktion för att hitta koner

```
import numpy as np
import random

def updateState(a,currentSpeed,currentPos,maxSpeed,frequency):
    a+=random.uniform(-1,1)
    newSpeed = [0,currentSpeed[1] + a/frequency]
    if newSpeed[1] < 0:
        newSpeed[1] = 0
    if newSpeed[1] > 25.27:
        newSpeed[1] = 25.27
    if np.linalg.norm(newSpeed)>maxSpeed:
        newSpeed = (0,maxSpeed)
    newPos = [0,0]
    newPos[0] = currentPos[0]
    newPos[1] = currentPos[1] + currentSpeed[1]/frequency+a/(2*
   frequency**2)
    return newSpeed, newPos
```

Listing A.7: Funktion för att uppdatera bilens tillstånd

```
import numpy as np
import random


def SpeedEstimation(newCones,oldCones,frequency,oldV,allowedDist=2)
   :
    toClose=False
    for i in oldCones:
        if(i[1]<allowedDist):
            toClose=True
    #print(oldCones)
    #print(newCones)
    counter = 0
    distance = (0,0)
    t = 1/frequency
    if toClose:
        speedEst = 0
        for i in newCones:
            for j in oldCones:
```

```
                if np.linalg.norm(i-j)<allowedDist:
                    distance = (j-i)
    else:
        distance = (oldCones[0]-newCones[0]+oldCones[1]-newCones
    [1])/2
        #print(distance)
    #print(distance)
    distance[1] += random.uniform(-0.04,0.04)

    speedEst = [0,distance[1]/t]


    return speedEst


def accelSpeedEstimation(newCones,oldCones,frequency,oldV,
    allowedDist=2):
    toClose=False
    for i in oldCones:
        if(i[1]<allowedDist):
            toClose=True
    #print(oldCones)
    #print(newCones)
    counter = 0
    distance = (0,0)
    t = 1/frequency
    if toClose:
        speedEst = 0
        for i in newCones:
            for j in oldCones:
                if np.linalg.norm(i-j)<allowedDist:
                    distance = (j-i)
    else:
        distance = (oldCones[0]-newCones[0]+oldCones[1]-newCones
    [1])/2
        #print(distance)
    #print(distance)
    distance[1] += random.uniform(-0.02,0.02)
    a = 2 *(distance[1]*frequency-oldV[1])
    #speedEst = [0,np.sqrt(2*distance[1]*a+oldV[1]**2)]
    #speedEst = [0,oldV[1]+a]
    #speedEst = [0,2*distance[1]/t-oldV[1]]
    #speedEst = [0,distance[1]/t]
    speedEst = [0,distance[1]/t+a/2]
    #print(distance[1]-oldV[1]*t)
    #print(speedEst)

    return speedEst
```

Listing A.8: Funktion för att uppskatta hastigheten från avståndsmätningar

A.3.4 Dynamiska krafter

```
import numpy as np
import matplotlib.pyplot as plt

# === Constants ===
lambda_desired = 0.13
max_torque = 220  # [Nm]
mass = 207 / 4  # [kg]
wheel_radius = 0.2286  # [m]
inertia = 2.84 / 4  # [kgm^2]
mu_peak = 1.3
g = 9.81  # [m/s^2]
dt = 0.0001  # [s]
total_time = 4.0  # [s]
steps = int(total_time / dt)
C_rr = 0.010  # Rolling resistance coefficient
C_d = 1.5 / 4  # Drag coefficient
C_l = 3.69 / 4 # Lift coefficient
A = 1.16 / 4  # Frontal area [m^2]
rho = 1.225  # Air density [kg/m^3]

# == Tuneable parameters ==
sigma0 = 96876.66
sigma1 = 440
sigma2 = 1.5
v_s = 10
z = 0.0
K_p = 111
K_i = 0
K_d = 1
power_limit = 80000  # [W]

# === NEW: Hard RPM Limits ===
MAX_RPM = 20000  # Hard cap on wheel RPM [rev/min]
MAX_WHEELSPEED = MAX_RPM /30 * np.pi /13 * wheel_radius  # [m/s]

def PID_regulator(e, prev_e, integral=0, dt=0.01, K_d=K_d, K_i=K_i)
    :
    integral += e * dt
    de = (e - prev_e) / dt
    return K_p * e + K_d * de + K_i * integral

# === Simulation Setup ===
vehicle_speed = [1e-5]
omega = [1e-5 / wheel_radius]
torque = [80]
time = [0.0]

# Force tracking lists (INITIALIZE ALL HERE)
error_list = [0]
F_drag_list = []
F_roll_list = []
F_friction_list = []
F_car_list = []
F_net_list = []
```

```
F_downforce_list = []
F_resistive_list = []
slip = []
acceleration_list = []
eta_list = []
distance = [0.0]
rpm_list = []
power_list = []
wheelspeed_mps_list = []  # NEW: Track wheel speed in m/s
F_car_hold = None

# === Main Simulation Loop ===
for step in range(steps):
    v = vehicle_speed[-1]
    w = omega[-1]
    t = torque[-1]
    current_time = time[-1]

    # === Calculate wheel and vehicle speeds ===
    current_rpm = w * (60 / (2 * np.pi))
    current_wheelspeed = w * wheel_radius  # [m/s]

    # Store for plots
    rpm_list.append(current_rpm)
    wheelspeed_mps_list.append(current_wheelspeed)

    # === Slip calculation ===
    wheel_speed_effective = min(current_wheelspeed, MAX_WHEELSPEED)
     # Clip wheel speed for slip calc
    current_slip = (wheel_speed_effective - v) / max(
   wheel_speed_effective, 1e-5)
    slip.append(current_slip)

    # === Current power calculation ===
    current_power = t * w if current_rpm <= MAX_RPM else 0
    power_list.append(current_power)

    # === PD Torque control every 0.01s ===
    error = lambda_desired - current_slip
    error_list.append(error)

    if step % 100 == 0:
        u = t + PID_regulator(error, prev_e=error_list[-2])
        u = np.clip(u, 0, max_torque)
    else:
        u = t

    # === Forces ===
    v_rel = wheel_speed_effective - v
    z_dot = v_rel - (abs(v_rel) / v_s) * z
    z += dt * z_dot
    F_friction = sigma0 * z + sigma1 * z_dot + sigma2 * v_rel
    F_drag = 0.5 * rho * C_d * A * v**2
    F_roll = C_rr * mass * g
    F_downforce = -0.5 * rho * C_l * A * v**2
    F_resistive = F_drag + F_roll + F_friction + F_downforce
```

```
108
109     # === Driving force ===
110     if current_power < power_limit:
111         F_car =  mass * 2 * u / (wheel_radius * (mass + inertia /
    wheel_radius**2))
112     else:
113         F_car = 0
114
115     # === BIG CONSTRAINT BLOCK ===
116     if (current_rpm >= MAX_RPM) or (current_wheelspeed >=
    MAX_WHEELSPEED):
117         # RPM limit reached
118         new_w = MAX_WHEELSPEED / wheel_radius  # Freeze wheel speed
119         u = t  # Freeze torque (do not update anymore)
120         alpha = 0  # No more angular acceleration
121         if F_car_hold is None:
122             F_car_hold = F_car  # Capture F_car at first moment you
     reach max speed
123         F_car = F_car_hold
124     else:
125         # Normal motion update
126         alpha = (u - F_friction * wheel_radius) / inertia
127         new_w = w + alpha * dt
128
129     F_net = np.abs((F_car) - F_resistive)
130     a_vehicle = F_net / mass
131
132     if v >= MAX_WHEELSPEED:
133         F_net = 0
134         new_v = v  # Freeze vehicle speed
135     else:
136         new_v = v + a_vehicle * dt
137
138     # Store forces
139     F_drag_list.append(F_drag)
140     F_roll_list.append(F_roll)
141     F_friction_list.append(F_friction)
142     F_downforce_list.append(np.abs(F_downforce))
143     F_car_list.append(F_car)
144     F_net_list.append(F_net)
145     acceleration_list.append(a_vehicle)
146     F_resistive_list.append(F_resistive)
147     # === Update states ===
148     omega.append(new_w)
149     vehicle_speed.append(new_v)
150     time.append(current_time + dt)
151     distance.append(distance[-1] + new_v * dt)
152     torque.append(u)
```

Listing A.9: P-regulator in python

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

CHALMERS