Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Rendell Cale

Modeling and Control of a Four-wheel Autonomous Agricultural Robot

Master's thesis in Cybernetics and Robotics
Supervisor: Torleiv Håland Bryne
June 2021

Rendell Cale

Modeling and Control of a Four-wheel Autonomous Agricultural Robot

Master’s thesis in Cybernetics and Robotics
Supervisor: Torleiv Håland Bryne
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of Science and Technology

Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

MSC THESIS DESCRIPTION SHEET

Name:	Rendell Walter Cale
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Modellering og regulering av en firehjuls autonom landbruks robot
Thesis title (English):	Modeling and Control of a four-wheel autonomous agricultural robot.

Thesis Description: The project thesis work is given in collaboration with AutoAgri, a local start-up company in Fosen making small and lightweight autonomous agricultural implements carrying robot. The goal of the robot is that to reduce cost for the farmer, reduce soil compaction and reduce greenhouse gas emissions compared to standard tractors of today.

The following tasks should be considered:

1. Perform a short literature review on
   a. Vehicle and cars kinematics and dynamics
   b. Wheel modeling and control
   c. Wheel friction
   d. High-level control strategies
2. Redesign and implement a simulator for the robot from the specialization project if deemed necessary. Present and justify any assumptions and design decisions made.
3. Develop a hierarchical control system architecture for the robot
   a. Define and design the control levels and define appropriate signals flows and time-scale separation of the different control system levels.
   b. Design and implement a low-level control strategy for each wheel.
   c. Design and implement a high-level controller for the robot
4. Present and discuss your results using relevant case studies.
5. Conclude your results and suggest further work.

Start date:	2021-01-18
Due date:	2021-06-14
Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Associate professor Torleiv H. Bryne, Dept. of Eng. Cybernetics, NTNU

Abstract

This master thesis develops a hierarchical control system for a 4-wheel-individual-steering (4WIS) vehicle, which is being created by the Norwegian agricultural startup AutoAgri. A novel controller which allows simultaneous control of course and yawrate is proposed. This controller uses the unique features of 4WIS vehicles, and improves upon the existing 4WIS controllers by unifying standard 4WIS driving-modes vehicles into a single framework. By unifying the driving-modes, it removes the need for discountinuous mode switching and mode-selection logic. A robust steering angle controller is developed using sliding-mode control (SMC). The control systems are developed and integrated using Robot Operating System 2 (ROS2), and they are tested using Gazebo to simulate the vehicle. To demonstrate the functionality of the system, we developed guidance and manual control systems. The guidance system uses waypoints to generate a continuous curvature path, and applies vector field guidance to control the course of the vehicle. In the manual control system, the vehicle is controlled by a human operator using a joystick input device.

The proposed control system is capable of handling a wide variety of cases and some unmodelled disturbances. There is a problem with chatter in the steering angle SMC which needs to be addressed before the control system can be applied in practice, but several strategies to reduce this are discussed.

Sammendrag

Denne masteroppgaven utvikler et hierarkisk kontrollsystem for en 4-hjul-individuelt-styrt (4WIS) bil, som blir laget av det norske landbruksoppstartsselskapet AutoAgri. Et nytt type kontrollsystem som tillater simultan kontroll av kurs og yawrate blir foreslått. Denne kontrolleren bruker egenskaper unikt for 4WIS biler, og forbedrer eksisterende kontrollere ved å forene standard kjøremoduser inn i ett rammeverk. Ved å forene kjøremodusene, fjernes behovet for diskontinuerlig modus-bytting og modus-seleksjonslogikk. En robust hjulstyringsvinkel-kontroller blir utviklet ved å bruke sliding-mode kontroll (SMC). Kontrollsystemene er utviklet og integrert med Robot Operating System 2 (ROS2), og de blir testet med Gazebo for å simulere bilen. For å demonstrere funksjonaliteten til systemet, så utviklet vi guiding- og manuell-kontrollsystemer. Guiding systemet bruker veipunkter for å generere en bane med kontinuerlig kurvatur, og vektorfelt for å styre kursen til bilen langs banen. I manuell kontroll blir bilen styrt av en menneskelig operatør ved hjelp av en joystick.

Det foreslåtte kontrollsystemet er i stand til å håndtere et variert antall tilfeller og noen umodellerte forstyrrelser. Det har et problem med chatter, altså høyfrekvente oscillasjoner, i hjulstyringsvinkel-kontrolleren som må bli addressert før kontrollsystemet kan bli tatt i bruk, men flere strategier for å redusere det blir diskutert.

Preface

This master thesis is written as the final part of my Master of Science degree in Technical Cybernetics at NTNU. It represents a full semester worth of work and builds upon five years of dedicated study.

I would like to thank my supervisor Torleiv Håland Bryne for giving instrumental feedback, and trusting me with a lot of freedom in my work. I would also like to thank AutoAgri for sharing information about their product and creating an interesting use-case for the tools I have acquired during my studies. This work was conducted during the COVID-19 pandemic, and while access to the university campus was restricted, I am grateful that NTNU allowed reading rooms to stay open throughout the semester.

Contents

List of Figures

List of Tables

Acronyms

4WIS four-wheel-individual-steering. 1, 2, 5, 17, 21, 27, 31, 39, 55, 59, 92, 101

API application programming interface. 36, 37

DDS data distribution service. 23

ENU East-North-Up. 6

GC guidance and control. 1, 21, 28, 81, 82

GNC guidance, navigation and control. 21

GUI graphical user interface. 1, 3, 48

ICR instantaneous center of rotation. 17, 18, 19, 56

IDL interface description language. 23, 24

ODE Open Dynamics Engine. 24, 31, 33

PI proportional-integral. 58

PID proportional-integral-derivative. 40, 60

QGC QGroundControl. 66, 96, 97, 98

ROS2 Robot Operating System 2. 2, 22, 23, 24, 36, 37, 95

SMC sliding-mode control. 41, 43, 46, 49, 50, 51, 62, 63, 103, 105

UAV Unmanned Aerial Vehicle. 92, 96

URDF unified robot description format. 34

1 Introduction

Agriculture is on the verge of an automation revolution with many companies dedicating their resources to it [21]. This is timely, since according to the UN, the global population will hit 9.7 billion by 2050 and the food production needs to increase by 70 % to accomodate this [4]. AutoAgri is a Norwegian company developing a novel agricultural vehicle designed with autonomy in mind. It boasts several advantages over typical agrictural vehicles, like reduced cost via autonomous operations, reduced soil compaction via lower mass and better load distribution, and reduced carbon footprint via a hybrid drivatrain [1]. The AutoAgri vehicle is a drop-in replacement for mid-sized agrictural vehicles. It weighs about 2 metric tonnes and is capable of carrying an additional 2 metric tonnes in payload. A unique feature, is that it is a four-wheel-individual-steering (4WIS) vehicle, meaning that each wheel can be steered and driven independently of the others. It is designed to be fully autonomous, which allows the farmer to operate it when conditions are ideal, as opposed to when the farmer is available. The vehicle can operate in the field and give live feedback to the farmer about the progress and any potential issues. It is equipped with collision avoidance sensors and software, so that if any unexpected animals walk into the field, they will not get hurt. A fully autonomous agricultural vehicle can increase the productivity, reduce the need for pesticide and manual labour. But the road to full autonomy is long and expensive, and the vehicle needs to prove its capabilities without autonomous features first. Currently the vehicle is being developed for manual control, but it is equipped with sensors to measure its surroundings and over time more and more autonomous features will be added.

1.1 Main contributions

The main contributions of this work are given below.

Developed a hierarchical control system

This work develops a hierarchical control system insired by guidance and control (GC). We demonstrate it by creating two applications; path-following and manual control. In the path-following application, the vehicle is controlled via waypoints which are specified through a graphical user interface (GUI). In the manual control mode, the vehicle is controlled using joystick inputs.

Robust sliding-mode control of wheel steering angle

A sliding-mode controller is designed and implemented to robustly control the steering angle of the wheels. Assuming we can put an upper bound on the friction resistance experienced by the wheel, it is able to control the steering angle precisely under unknown friction conditions. It does, unfortunately, have a problem with chatter which could be problematic for real-life applications.

Novel no-slip based course-yawrate controller

This work presents a novel control system for 4WIS vehicles which controls course and yawrate indepedently. This allows the vehicle to drive in any direction with any orientation. It unifies the mode-based approach often employed with 4WIS vehicles and is able to describe all no-slip based driving manouvers unique to 4WIS vehicles.

Other contributions

In addition to the contributions listed above, this work presents several minor contributions. We

- developed a 4WIS vehicle simulator using Gazebo,
- implemented continuous curvature path-smoothing and applied it to vehicle path-following,

To create the software in modular, loosely coupled components, we used Robot Operating System 2 (ROS2) as a software framework.

1.2 Structure of report

The report is roughly divided into five parts; vehicle modeling, system design, simulator, motion control systems, and applications.

Chapter 2 deals with modeling the vehicle. It presents equations of motion for various components of the system, and it defines core concepts that will be used throughout the text.

Chapter 3 presents the design of the system and the software engineering considerations that went into it. It also explains the third party tools we have used and what they contribute.

Chapter 4 presents the simulator that is used to evaluate the motion control systems. It is based on Gazebo, and in this chapter we explain how the AutoAgri vehicle was implemented inside Gazebo.

Chapters 5 and 6 presents the motion control systems that control the vehicle. This includes the guidance-system for path-following, and the control system that is responsible for setting actuator torques based on the reference signals. Results from case studies are presented throughout these chapters as we build the control system from the ground up.

Chapter 7 presents two example applications demonstrating how the system may be used in practice. The first example is manual control where we control the vehicle via a Playstation controller. The second example is autonomous control where we use a GUI to specify waypoints on a map, and use the motion control system to follow those waypoints.

We wrap up the thesis by discussing the limitations, points of improvement, and suggestions for feature work in Chapter 8.

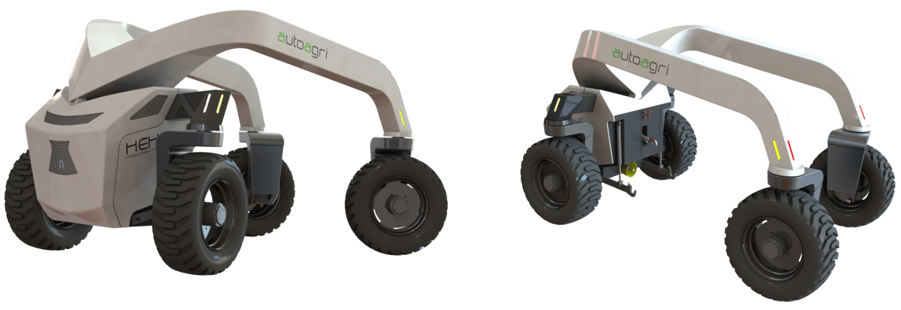
2 Vehicle Modeling

This chapter presents equations of motion for 4WIS vehicles. This chapter is in large part based on Ch. 2 and 3 of the specialization project report [6]. We first give an overview of the system, then we cover the relevant reference frames and vector notation. After that, we present simplified equations of motion for specific aspects of the system. We note that [6] gives a more complete description of dynamical equations, but they have not been used in this work.

2.1 System overview

The AutoAgri vehicle is shown in Fig. 2.1a. It is a 4WIS vehicle, meaning that each wheel is independently steered and driven. The wheels are driven by electric motors, which are powered either by an onboard lithium battery or a diesel generator. Unlike many 4WIS vehicles, the wheels can be rotated 360 degrees, which opens up many intersting movement patterns such as sideways driving and rotations on the spot.

The weight of the vehicle is 2380 kg, and it is capable of carrying an additional 2000 kg load in between the wheels. The distance between the front and rear wheels, called the wheelbase, is 2830 mm, and the distance between the left and right wheels, called the track, is 2000 mm. All the wheels are identical and have diameter 1010 mm and a width of 400 mm.



(a) Front view. (b) Rear view.

Figure 2.1: AutoAgri illustrations.

2.2 Reference frames and vector notation

To model the vehicle it is convenient to be able to describe positions and velocities in different reference frames. The three main reference frames we use are illustrated in Fig. 2.2, and they are the inertial, body, and wheel frames.

Inertial frame - $\{i\}$: The inertial frame has an arbitrary origin and is stationary with respect to the Earth. In accordance with the litterature on automotive vehicles (e.g. [17, 19]), we use an East-North-Up (ENU) convention for the inertial system. Note that it is a Cartesian coordinate system, so the east and north components represent distance from an origin, and not latitude and longitude as angles relative to the Earth. Due to the rotation and curvature of the Earth, this system is strictly speaking not an inertial reference frame, but since the vehicle will operate over small distances compared to the size of the Earth, we believe it is a good approximation.

Body frame - $\{b\}$: By rotating and translating the inertial reference frame to the center of mass of the vehicle, we get the body frame. Alternatively, we can say that the body frame defines the pose of the vehicle. If we assume the vehicle operates on flat terrain, then its position can be described by two numbers (E, N) representing distance in east and north directions from the origin. The orientation of the vehicle is descibed by the yaw or heading, which is denoted ψ. To denote the

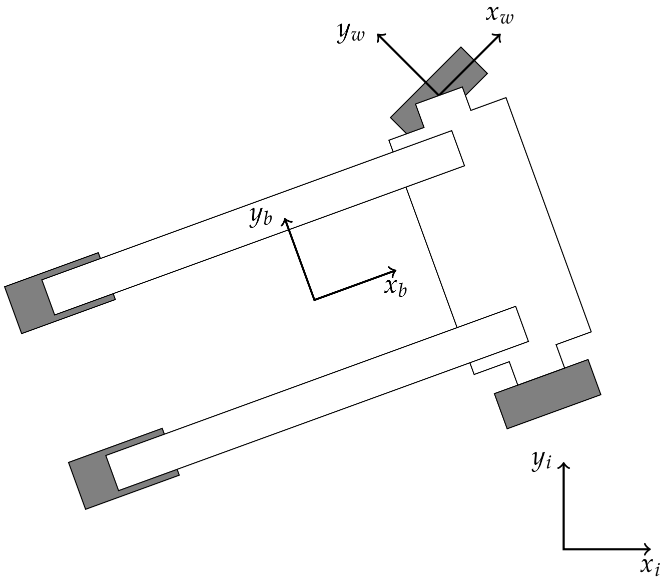


Figure 2.2: Reference frames viewed from top.

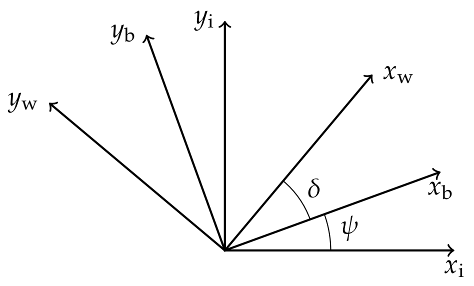


Figure 2.3: Reference frame orientations. Note that they are shown with the same origin for illustrational clarity.

position of the body frame relative to the inertial frame, we use the vector notation

$$\mathbf{p}_b^i = \begin{bmatrix} E \\ N \end{bmatrix} \tag{2.1}$$

In general the subscript gives a description of what the vector represents, and the superscipt describes which frame it is decomposed in. There exists more ellaborate vector notations, for instance as presented by [12], but we have not found it necesarry here.

Wheel frame(s) - $\{w\}$: In addition to the body frame, each wheel is given a reference frame which is centered on the wheel and rotated to align with the positive driving direction of the wheel, as illustrated in Fig. 2.2. The wheels are denoted by subscripts ij where $i \in \{\text{front}, \text{rear}\} = \{F, R\}$ and $j \in \{\text{left}, \text{right}\} = \{L, R\}$. Wheel ij is positioned at $\mathbf{p}_{ij}^b$ relative to the body frame. It is steered with an angle δ_{ij} relative to the body frame. For notational clarity, we will in many cases drop the subscript ij from the steering angle and other wheel states, and then it is implied that the statements apply to all wheels equally. The orientation of the reference frames are shown in Fig. 2.3.

2.2.1 Coordinate transformations

To express a vector from one coordinate frame in another, we need to apply a coordinate transformation. The theory behind this is covered rigourosly in [12], and we used it more actively in the specialization project [6]. The main tool we need are rotation matrices. To rotate a two-dimensional (2D) vector, we use the rotation matrix

$$\mathbf{R}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \tag{2.2}$$

Likewise, three-dimensional (3D) vectors are rotated about the z-axis using

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Note that (2.2) and (2.3) are essentially equivalent if we consider the 2D rotations in (2.2) to be rotations about the z-axis. Using these definitions and the theory covered in [34], we showed in the specialization project [6] that the rotation matrices between the reference frames are given by

$$\mathbf{R}_i^b = \mathbf{R}(-\psi) \Leftrightarrow \mathbf{R}_b^i = \mathbf{R}(\psi) \tag{2.4}$$

$$\mathbf{R}_b^w = \mathbf{R}(-\delta) \Leftrightarrow \mathbf{R}_w^b = \mathbf{R}(\delta) \tag{2.5}$$

This means that to rotate a body-frame vector $\mathbf{x}^b$ to the inertial frame, we would use

$$\mathbf{x}^i = \mathbf{R}_b^i \mathbf{x}^b = \mathbf{R}(\psi)\mathbf{x}^b \tag{2.6}$$

Note that to express positions from one frame in another frame, we need to include the translation component. For instance, say that the vehicle uses onboard sensors to detect an obstacle at $\mathbf{p}^b$ relative to the vehicle. The obstacle's position can then be transformed to the inertial frame using

$$\mathbf{p}^i = \mathbf{R}_b^i \mathbf{p}^b + \mathbf{p}_b^i \tag{2.7}$$

Reference frame transformations, especially with time-derivatives is covered in depth in [12], so we defer there for more details.

2.3 Wheel modeling

In this section we develop equations of motion for the wheel states. Each wheel can be steered and driven independently by two electric motors. To keep the model simple, we do not model the motors in any detail. It is assumed that each motor can be commanded to give a torque, and that torque is transfered immediately to the wheel. There are two wheel states that we want to model. The angular velocity ω and the steering angle δ. To distinguish between the wheels later, we use subscripts ij to reference a specific wheel, but when no subscript is given, the equations can be assumed to apply to all wheels.

The main external force that affects the wheels is friction between the tire and ground. The specialization project report [6] devotes a chapter to exploring this in detail, but here we give a brief summary of the results we found to be useful. After that we derive a set of models for the equations of motion.

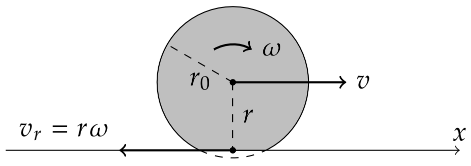


Figure 2.4: Wheel slip figure.

2.3.1 Slip

Consider a wheel driving in straight line as illustrated in Fig. 2.4. The load causes the wheel tires to compress giving a radius r. The angular velocity of the wheel is ω. If the tire perfectly sticks to the ground, then the velocity of the wheel will be the rotational equivalent velocity $v_r = r\omega$, but due to resistance, frictional lag, and other factors the actual velocity v may be different. The fact that the wheel's rotation may not be matched to the wheel's velocity, gives rise to a phenomon known as slip. There are several mathematical ways of defining slip. In [19], they argue that slip only has a physical meaning if it's limited to be within -1 and 1, so they use the following defintion.

$$s = \begin{cases} \frac{r\omega - v}{r\omega} & \text{for} \quad r\omega \geq v \quad \text{(driving)} \\ \frac{r\omega - v}{v} & \text{for} \quad r\omega < v \quad \text{(braking)} \end{cases} \tag{2.8}$$

Note that they always divide by the larger velocity, but if we allow ω and v to take on negative values, then this definition fails to limit the slip. An alternative and simpler definition is used by [28] to study wheel dynamics. They use

$$s = \frac{r\omega - v}{r|\omega|} \tag{2.9}$$

The differences between the slip definitions are subtle, but it is important to remember that slip is a construct, meaning that it is a property of the system that has over time been shown to be useful. There is no true definition of slip, but we need to be careful not to apply results obtained with one definition without using the same definition ourselves, atleast not without careful consideration.

2.3.2 Friction

The study of tire-road friction forces is a field in itself. It is particularly relevant for control systems in autonomous vehicles, since research has shown that it has difficult to control systems with friction without incorporating the friction model into the controller [37]. This was also experienced in the specialization project, where we found that we were only able to control the steering angle of the vehicle in low-friction environments [6].

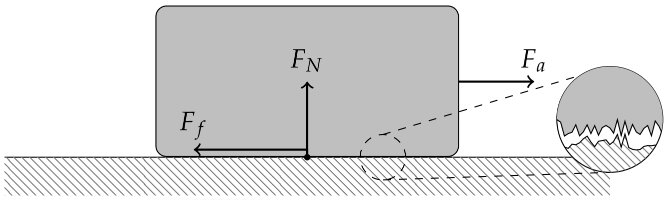


Figure 2.5: Static friction caused by asperities.

We begin by describing the two simplest friction models, namely static and kinetic. After that we review Burckhardt friction, which is a model specifically designed for tire-road friction.

Static and kinetic friction model

Static friction is a force that arises between two surfaces in contact with no relative velocity. It is caused mainly by asperities on each surface which cause the surfaces to "lock" together, as illustrated in Fig. 2.5. If we apply a force F_a, then the static friction force resisting motion is $F_f = F_a$ as long as the applied force does not exceed the breakaway force. The breakaway force is the maximum static friction, which is related to surface properties and weight. Summarizing the surface properties into a parameter of static friction μ_s, we write

$$F_f = \text{sat}\left(F_a, \mu_s F_N\right) \tag{2.10}$$

where the saturation function is

$$\text{sat}\left(x, M\right) = \begin{cases} -M & \text{when} \quad x < -M \\ x & \text{when} \quad |x| \leq M \\ M & \text{when} \quad x > M \end{cases} \tag{2.11}$$

If the applied force is bigger than $\mu_s F_N$, then the surfaces will move relative to each other, giving rise to *dry* friction. One of the simplest models of dry friction, is the Couloumb model which is commonly known as kinetic friction. It gives the friction force as

$$F_f = \mu_k F_N \text{sgn}\left(v\right) \quad \text{for} \quad v \neq 0 \tag{2.12}$$

Couloumb friction is similar in structure to static friction, but the parameter μ_k is typically smaller than μ_s. This is illustrated in Fig. 2.6.

Burckhardt semi-empirical friction model

A friction model that improves upon the static/kinetic model presented above is Burckhardt friction. Burckhardt is a fundamentally different type of model, as it is

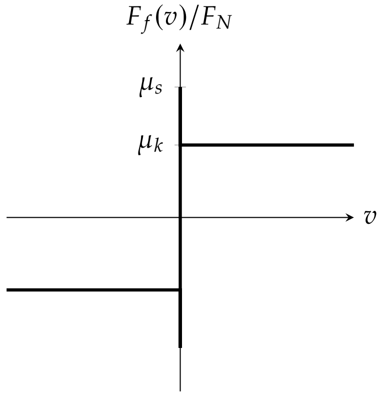


Figure 2.6: Combined static and kinetic friction.

Table 2.1: Example parameters for the simple Burckhardt model [5].

	c_1	c_2	c_3
Asphalt, dry	1.2801	23.99	0.52
Asphalt, wet	0.857	33.822	0.347
Snow	0.1946	94.129	0.0646
Ice	0.05	306.39	0

a semi-empirical model designed for tire-road friction estimation. It was proposed in [5] and is used extensively throughout the litterature (e.g. [19, 28]).

In Burckhardt's model, the friction parameter is modeled as a function of wheel slip. There are several versions of Burckhardt's model, but the simplest is

$$\mu(s) = c_1\left(1 - e^{-c_2 s}\right) - c_3 s \tag{2.13}$$

The parameters c_i have to be determined experimentally and will vary between applications. Burckhardt and Reimpell [5] gives a table of typical values, which we restate in Table 2.1. A plot of the model is given in Fig. 2.7.

Linear friction model

Note that around $s = 0$ in Fig. 2.7, the Burchkhardt model gives an almost linear relationship between μ and s. This is useful, since many vehicle applications will operate with low slip, which means we can approximate the friction parameter using a linear function

$$\mu = ks \quad \text{for small } s \quad . \tag{2.14}$$

If we have determined the Burckhardt parameters, then k can be determined by linearizing the Burckhardt model around $s = 0$. What we get then is

$$k = c_1 c_2 - c_3 \tag{2.15}$$

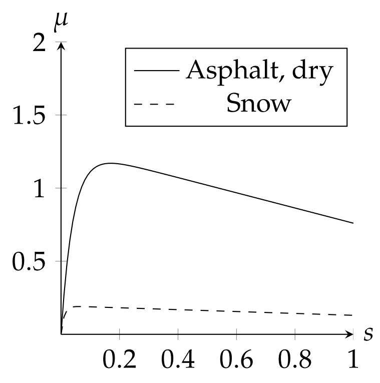


Figure 2.7: Burckhardt friction.

This simplification is only valid in a small region around the origin. It should in particular not be used if the slip exceeds the peak of the Burchkhardt model, which is given by

$$s_{\text{peak}} = \frac{1}{c_2} \ln \left(\frac{c_1 c_2}{c_3} \right), \tag{2.16}$$

since beyond the peak, the friction force changes drastically as we move from a static to a kinetic region.

First order dynamical friction model

It is known that static friction models, that is friction models without dynamical behaviour, are incapable of capturing effects like pre-sliding displacament and frictional lag (e.g. [10, 12, 23]). It was shown in [25] that the friction force can be modeled with a first order dynamical system

$$T\dot{F}_f = F - F_f \tag{2.17}$$

$$T = \frac{l}{r|\omega|} \tag{2.18}$$

where l is the relaxation length, and F is for instance the Burckhardt model. The relaxation length is a tire parameter related to rubber stiffness. It can be assumed to be in the same order of magnitude as the tire radius.

2.3.3 Equations of motion

In this section we present equations of motion for the wheel states. The wheel states are the angular velocity ω and the steering angle δ. The equations of motions were developed in the specialization project [6], and we restate them here as

$$J_y \dot{\omega} + rF_f = \tau_d \tag{2.19}$$

$$J_z \ddot{\delta} + M_f = \tau_s \tag{2.20}$$

J_y and J_z are the moments of inertia for the wheel about the y- and z-axes, respectively. r is the radius of the wheel. τ_d is the driving torque, and τ_s is the steering torque. Friction creates a force F_f on the angular velocity, and a steering resistance M_f on the steering angle.

To determine the friction force F_f we can use the Burckhardt friction model as given in Section 2.3.2 assuming we have access to the above ground velocity $\mathbf{v}$ of the wheel. The above ground velocity of the wheel is the velocity of the wheel relative to the ground, which can be computed as the velocity of the wheel frame. Determining the steering resistance M_f is unfortunately more challenging. In the specialization project, we estimated M_f offline as

$$M_f = \frac{1}{3}\text{sgn}\left(\dot{\delta}\right) w\mu_k F_Z \tag{2.21}$$

where w is the width of the tire, but we found that using this offline estimate in the control system yielded poor performance, which indicates that it is an imprecise estimate. While the exact value of M_f may not correspond to (2.21), we believe the structure of the estimate is correct. Based on this we argue there exists an upper bound $M = k\mu_k F_n$ such that the inequality

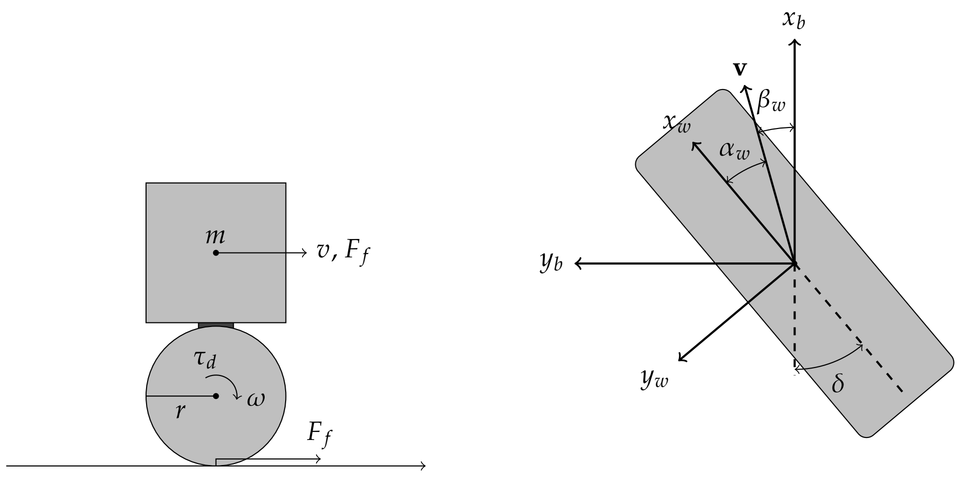
$$\left|M_f\right| \leq M = k\mu_k M \tag{2.22}$$

is always satisfied for some fixed value of k. While this does not provide a basis to simulate steering resistance, this property is used to derive a robust control system in Chapter 5.

2.3.4 Quarter vehicle model

Using the equations of motion for a single wheel, we can create one of the simplest vehicle models, namely the quarter vehicle model. We present two variations of the quarter vehicle model, the first one allows two dimensional movement and is used to illustrate a few core concepts. The second one is constrained to move only in one direction and is useful for analysis and control systems design.

2D quarter vehicle model The 2D quarter vehicle model has a single wheel and a mass, as illustrated in Fig. 2.8a. The wheel is moving horizontally in the xy-plane, and has a heading δ and angular velocity ω governed by (2.19) and (2.20). The angle α_w is called the tire side-slip angle, and is the angle between the wheel velocity $\mathbf{v}$ and the wheel heading. A related angle is the wheel sideslip angle,



(a) Side view. (b) Top view.

Figure 2.8: Quarter vehicle model.

denoted β_w, which is the angle between the wheel velocity and the vehicle body's x-axis. These angles are illustrated in Fig. 2.8b. To compute these angles, we can decompose the velocity vector in either the wheel frame or the body frame, and use

$$\alpha_w = \text{atan2}(v_y^w, v_x^w) \tag{2.23}$$

$$\beta_w = \text{atan2}(v_y^b, v_x^b) \tag{2.24}$$

We note that it is possible relate these quantities to a slip based friction model to create a complete dynamical model for the quarter vehicle model, but we do not need any more details than are given above for the current work.

1D quarter vehicle model A variation of the quarter vehicle model is the 1D quarter vehicle model that was used by [28] to study vehicle dynamics and wheel simulation stability. The model is illustrated by Fig. 2.8a, but we ignore the steering angle. This means that it can only move in one direction.

The equations of motion are obtained by Newtonian mechanics using the dynamic friction model of Section 2.3.2 in combination with a linear friction model. What we get then is

$$m\dot{v} = F_f \tag{2.25}$$

$$J\dot{\omega} = -F_f + \tau_d \tag{2.26}$$

$$\frac{l}{r|\omega|}\dot{F}_f = k\frac{r\omega - v}{r|\omega|} - F_f \tag{2.27}$$

Note that we use (2.9) as a slip definition. This allows us to rewrite and get

$$\dot{v} = \frac{1}{m}F_f \tag{2.28}$$

$$\dot{\omega} = -\frac{r}{J}F_f + \frac{1}{J}\tau_d \tag{2.29}$$

$$\dot{F}_f = k\frac{r\omega - v}{l} - \frac{r|\omega|}{l}F_f \tag{2.30}$$

This wheel model is interesting because it is almost linear, yet it still includes essential features of the system like friction forces. If we consider $u = \tau_d$ to be the input and $y = \omega$ to be the output, then it is a passive system. To see this we use the storage function

$$V = \frac{m}{2}v^2 + \frac{J}{2}\omega^2 + \frac{l}{2k}F_f^2 \tag{2.31}$$

Its time derivative is given by

$$\begin{aligned}
\dot{V} &= mv\dot{v} + J\omega\dot{\omega} + \frac{l}{k}F_f\dot{F}_f && (2.32)\\
&= vF_f - r\omega F_f + \omega\tau_d + F_f\left((r\omega - v) - \frac{r|\omega|}{k}F_f\right) && (2.33)\\
&= \omega\tau_d - \frac{r|\omega|}{k}F_f^2 && (2.34)\\
&= yu - \phi(\omega, F_f) && (2.35)
\end{aligned}$$

Since $\phi(\omega, F_f) \geq 0$ for all wheel states, we can write $uy \geq \dot{V}$, which means the system is passive [18]. One might expect the system to be strictly passive in practice, since if we stop applying torque then the wheel should slow down over time and come to a stand still. This modeling inaccuracy stems from the fact that we have not included any dampening effects.

2.4 Load transfer

When accelerating, the load experienced by each wheel will change. This effect is called load transfer, and in the specialization project [6] we derived a set of equations that describe this. For completeness, we will restate those results here.

Consider a vehicle with horizontal acceleration $\mathbf{a}^b = \begin{bmatrix} a_x & a_y \end{bmatrix}$ as illustrated in Figs. 2.9 and 2.10. Kiencke and Nielsen [19] defines inertia signals for the chassis as the negative acceleration

$$\mathbf{a}^b_{\text{ch}} = -\mathbf{a}^b. \tag{2.36}$$

In the specialization project we found that the load transfer equations are given by a superposition of the unloaded and loaded configuration.

$$F_{Zij} = F_{Zij,\text{unloaded}} + F_{Zij,\text{loaded}} \tag{2.37}$$

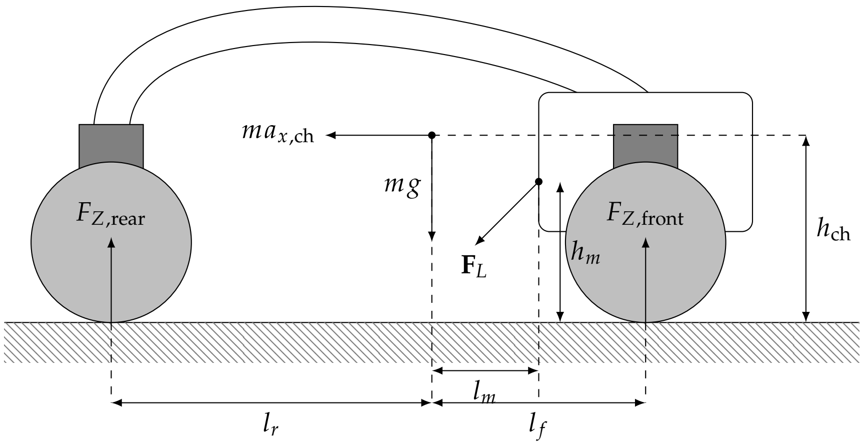


Figure 2.9: Load transfer setup, viewed from side.

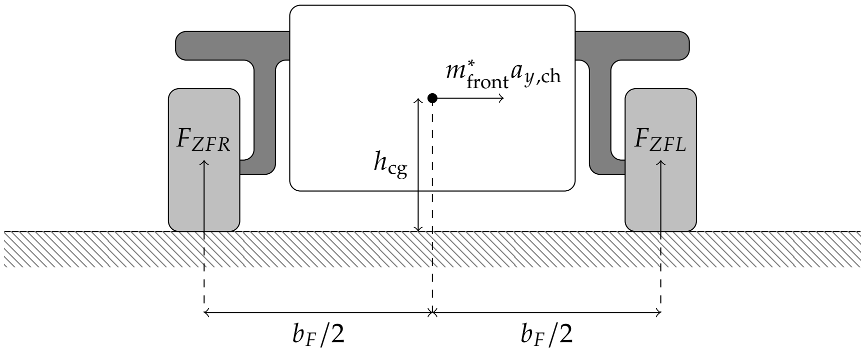


Figure 2.10: Load transfer setup, viewed from front.

Where the unloaded configurations are given by

$$F_{ZFL,\mathrm{unloaded}} = mg\left(\frac{l_R}{l} - \frac{h_{\mathrm{cg}}a_{x,\mathrm{ch}}}{lg}\right)\left(\frac{1}{2} - \frac{h_{\mathrm{cg}}a_{y,\mathrm{ch}}}{b_F g}\right), \tag{2.38}$$

$$F_{ZRL,\mathrm{unloaded}} = mg\left(\frac{l_F}{l} + \frac{h_{\mathrm{cg}}a_{x,\mathrm{ch}}}{lg}\right)\left(\frac{1}{2} - \frac{h_{\mathrm{cg}}a_{y,\mathrm{ch}}}{b_R g}\right), \tag{2.39}$$

$$F_{ZRR,\mathrm{unloaded}} = mg\left(\frac{l_F}{l} + \frac{h_{\mathrm{cg}}a_{x,\mathrm{ch}}}{lg}\right)\left(\frac{1}{2} + \frac{h_{\mathrm{cg}}a_{y,\mathrm{ch}}}{b_R g}\right), \tag{2.40}$$

$$F_{ZFR,\mathrm{unloaded}} = mg\left(\frac{l_R}{l} - \frac{h_{\mathrm{cg}}a_{x,\mathrm{ch}}}{lg}\right)\left(\frac{1}{2} + \frac{h_{\mathrm{cg}}a_{y,\mathrm{ch}}}{b_F g}\right), \tag{2.41}$$

and the loaded configuration are given by

$$F_{ZFL,\text{loaded}} = \left(\frac{l_R + l_m}{l} F_{L,z} - \frac{h_m}{l} F_{L,x}\right)\left(\frac{1}{2} - \frac{h_{\text{cg}} a_{y,\text{ch}}}{b_F g}\right), \tag{2.42}$$

$$F_{ZRL,\text{loaded}} = \left(\frac{l_f - l_m}{l} F_{L,z} + \frac{h_m}{l} F_{L,x}\right)\left(\frac{1}{2} - \frac{h_{\text{cg}} a_{y,\text{ch}}}{b_R g}\right), \tag{2.43}$$

$$F_{ZRR,\text{loaded}} = \left(\frac{l_f - l_m}{l} F_{L,z} + \frac{h_m}{l} F_{L,x}\right)\left(\frac{1}{2} + \frac{h_{\text{cg}} a_{y,\text{ch}}}{b_R g}\right), \tag{2.44}$$

$$F_{ZFR,\text{loaded}} = \left(\frac{l_r + l_m}{l} F_{L,z} - \frac{h_m}{l} F_{L,x}\right)\left(\frac{1}{2} + \frac{h_{\text{cg}} a_{y,\text{ch}}}{b_F g}\right). \tag{2.45}$$

2.5 Instantaneous center of rotation

The instantaneous center of rotation (ICR) is the point that every point on the vehicle rotates about, and is illustrated in Fig. 2.11. For typical front-wheel steered vehicles with Ackermann mechanisms, it will lie on a line collinear to the rear axle, but 4WIS vehicles are able to control it to arbitrary positions. The main reason we are interested in it, is because during normal driving all wheel will point orthogonally to the ICR, and when the wheel direction is not orthogonal to the ICR, the wheel will slip and potentially create large breaking forces. If we want to develop an optimization based control strategy, then the ICR provides a natural way to constrain the wheel states to each other. Using the body states, we can determine the ICR of the vehicle relative to the body frame.

Assume the vehicle has horizontal velocity $\mathbf{v} = [v_x, v_y, 0]^T$ and yawrate $\dot{\psi}$. Then the ICR relative to the body is

$$x_{\text{ICR}} = \frac{-v_y}{\dot{\psi}} \quad \text{and} \quad y_{\text{ICR}} = \frac{v_x}{\dot{\psi}} \tag{2.46}$$

This can be proved using rigid body kinematics. The velocity of the ICR can be written as a sum of the velocity of the body and the angular velocity about the

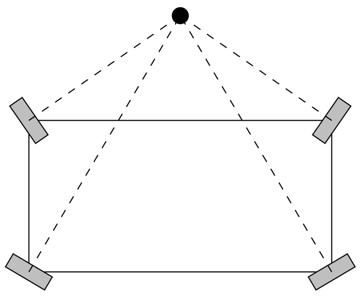


Figure 2.11: All wheels pointing toward the ICR.

ICR. By definition, the velocity of the ICR is zero, so we get

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix}v_x\\v_y\\0\end{bmatrix} + \begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix} \times \begin{bmatrix}x_{\mathrm{ICR}}\\y_{\mathrm{ICR}}\\0\end{bmatrix} \tag{2.47}$$

Solving for $x_{\mathrm{ICR}}, y_{\mathrm{ICR}}$ gives the above results, which we write as

$$\mathbf{p}^b_{\mathrm{ICR}} = \frac{1}{\dot{\psi}} \begin{bmatrix}-v_y\\v_x\\0\end{bmatrix} \tag{2.48}$$

2.6 No-slip conditions

No-slip conditions, or no-slip assumptions, are assumptions that greatly simplify the vehicle model, because they let us move from a dynamical vehicle model to a kinematic vehicle model. There are two no-slip assumptions:

1. The tire sideslip angle α_{ij} (see Fig. 2.8b) of each wheel is zero. This implies that the wheel velocity direction is aligned with the wheel direction.

$$\alpha_{ij} = 0 \Leftrightarrow \beta_{ij} = \delta_{ij} \quad \text{for all wheels} \quad . \tag{2.49}$$

2. The angular velocity of each wheel is matched to the ground velocity of the wheel.

$$r\omega_{ij} = v_{ij} \quad \text{for all wheels} \tag{2.50}$$

Note that these assumptions are idealized approximations, and all wheels exhibit slip when driving. This is because without slip, the wheels generate no force, which can be seen by plugging $s = 0$ into the Burckhardt model. A problem with the no-slip assumptions is that they constrain the wheel states in highly non-linear ways, which make it hard to use in optimization. This is mainly a problem if we want to map from wheel states to vehicle states, because most wheel state combinations do not satisfy the no-slip assumption. These nonlinear constraints are related to the ICR of the vehicle. If the ICR of the vehicle is $\mathbf{p}_{\mathrm{ICR}}$ and the position of wheel ij is $\mathbf{p}_{ij}$, then the wheel drives around the ICR with an instantaneous radius of

$$R_{ij} = \left\| \mathbf{p}_{\mathrm{ICR}} - \mathbf{p}_{ij} \right\| \tag{2.51}$$

No-slip assumptions give that the speed of the wheel is $v_{ij} = r\omega_{ij}$, which means that the wheel "orbits" the ICR with an instantaneous orbital period $T_{ij} = 2\pi R_{ij}/r\omega_{ij}$. Since all wheels must orbit the ICR with the same period, the wheels states are constrained by $T_{FL} = T_{RL} = T_{RR} = T_{FR}$. Writing this out and simplifying since all wheels have the same radius, we get

$$\frac{R_{FL}}{\omega_{FL}} = \frac{R_{RL}}{\omega_{RL}} = \frac{R_{RR}}{\omega_{RR}} = \frac{R_{FR}}{\omega_{FR}} \tag{2.52}$$

This constraint is only useful if we know the ICR, but the ICR is determined by atleast two of $\{\delta_{FL}, \delta_{RL}, \delta_{RR}, \delta_{FR}\}$. These non-linear constraints make it difficult to apply optimization techniques to no-slip models, especially if we want to optimize over the wheel state space $(\delta_{ij}, \omega_{ij})$.

3 System Design

Implementing vehicle simulation and control systems is a big software development task. For this project, we have used the lessons we learned in the specialization project, and rebuilt most of what was made there from scratch. As we'll see, this has allowed us to take advantage of tools and software available in the robotics community, and to create a simulator which is easier to maintain and more extensible for future work. In this chapter we describe the overall architecture of the system, and the third party tools that we have used.

3.1 Design goals

The design goals of the system specify what we want the end-product to be capable of. The primary goal of this work is to develop and evaluate a hierarchical control system for 4WIS vehicles, so the design of the system must facilitate that. Due to the ongoing pandemic, it has not been feasible to test the developed systems on the physical AutoAgri vehicle as was originally intended, so a simulator is used in its place. Ideally, the simulator would be a perfect substitute for the AutoAgri vehicle, emulating onboard sensors and actuator interfaces. But developing a simulator with such accuracy is a big task, and since we knew from early on that we were not going to perform field tests, we decided it was not worth the cost of implementing such an accurate simulation.

The end-system should be capable of controlling a simulated version of the AutoAgri vehicle, and it should provide both autonomous and manual modes of control.

3.2 System overview

The system is divided into four main modules with distinct tasks; simulator, state estimator, motion control, and operation. The simulator module does physics simulation and sensor emulation. The state estimator module does sensor fusion and state estimation. The motion control module controls the vehicle, translating high-level motion commands into actuator torques. Finally the operation module implements high-level operations that the vehicle can perform. This includes providing an interface for an operator to control the vehicle either manually via joystick commands or autonomously via waypoints. The top-level signal flow is illustrated in Fig. 3.1.

The main focus in this project has been developing a robust and capable motion control module. The other modules are minimally implemented, but we add them to the architecture to illustrate what a more complete system may look like. The system architecture is heavily inspired by guidance, navigation and control (GNC) as presented in [13]. The GC part is contained in the motion control

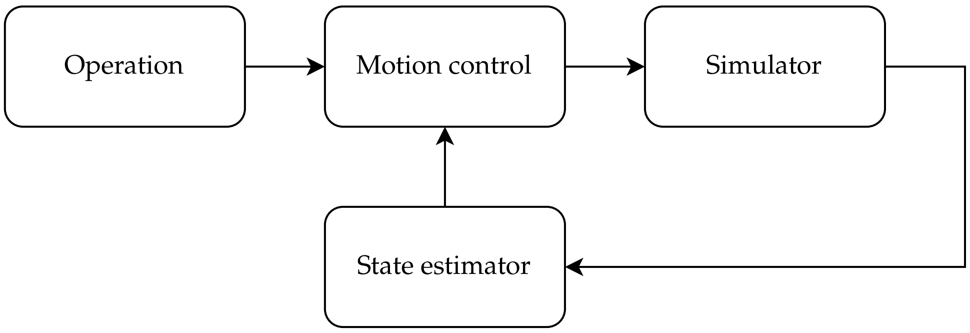


Figure 3.1: System overview.

module. An overarching design principle is that the modules should not need to change if we port this to a real vehicle. All that is needed is to swap out the simulator module with a real vehicle. The reason we strive for this is that it allows us to test the software extensively in simulation before deploying it to the vehicle. From a software engineering standpoint, this is preferable because simulator tests can then test the actual implementation that is used on the vehicle, before it is deployed. It saves us from having separate prototype and production implementations, which makes the software development process more robust. And the closer the simulator emulates the real vehicle, the more we can be confident that simulator results will carry over to the real vehicle.

3.3 Software tools

To build the system, we decided it would be beneficial to use third party software tools. The benefit of this is that it allows us to develop faster and better comply with standard tools used by the robotics community. The two main tools we use are ROS2 for building software, and Gazebo for physics simulation.

3.3.1 ROS2

ROS2 [30] is a framework for robotics applications, which has seen extensive use in robotics research and some industrial applications [29]. For our purposes it acts as a "software glue" that allows us to decompose the system into many smaller modules, and glue them togheter with typechecked messages. We note that there are other alternatives to ROS2 like ZeroMQ and Protocol Buffers, but we chose ROS2 primarily due to prior experience with it. ROS2 aims to address many of issues that have been found in its predecessor over the years, such as high communication overhead, security options, embedded systems, real time requirements, and much more. This means that ROS2 is not merely a research tool. It is also a robotics framework made for the industry.

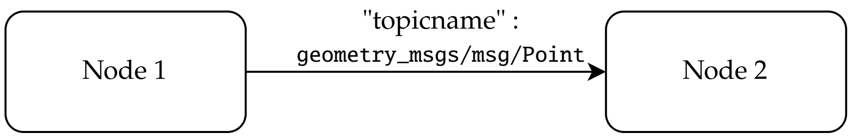


Figure 3.2: ROS2 example application.

```
float64 x
float64 y
float64 z
```

Listing 3.1: geometry_msgs/msg/Point

```
#include <geometry_msgs/msg/point.hpp>

geometry_msgs::msg::Point point;
point.x = ...
point.y = ...
point.z = ...
```

Listing 3.2: ROS2 message example in C++.

For our purposes, the two most important things it provides are

1. a way of dividing large software into smaller components called nodes, and
2. a message based communication layer for sending data between the nodes.

In addition to these features, ROS2 provides many development tools that allow us to introspect and debug running applications. An example application is shown in Fig. 3.2. In it we have two executables running in parallel, called nodes. Node 1 *publishes* a message of type `geometry_msgs/msg/Point`, and node 2 *subscribes* to it. The message is available globally in the system, so any number of nodes can decide to subscribe to it. To subscribe, it needs to know the address of the message, called the *topic*, which is represented by a string. Sending and recieving messages is handled by a data distribution service (DDS) and it can be configured to work even when nodes are running on different devices.

For data integrity reasons, the messages are always typed. The type of the message is specified in langauge independent interface description language (IDL) files. For instance the message used above, `geometry_msgs/msg/Point`, is defined in Listing 3.1. It consists of three 64-bit floating point numbers representing the x-, y-, and z-components of a point. The ROS2 build system compiles these files into language specific code. For instance, Listing 3.2 shows how ROS2 messages can be accessed in C++. ROS2 comes with a large library of message types for different purposes, and we can also create our own to represent specific aspects of the system. When we denote a message, we will refer to it using `<package-name>/msg/<type-name>`. A list of the message types we have used and created are given in Appendix B.

3.3.2 Gazebo

Gazebo [20] is a popular open source physics simulator, designed for robotics research. Using Gazebo for vehicle simulation and integrating it with ROS2 requires some setup, which we detail in Chapter 4.

3.4 Modules & interfaces

This section expands upon the system overview presented in Section 3.2. We present all the modules that we have implemented and give the message types that they use to communicate. To present the interfaces, we describe their physical meaning and show the corresponding IDL files.

3.4.1 Simulator

The simulator module implements a simulation model, which is the most accurate and detailed model we have of the vehicle. In the specialization project, we found that implementing the equations of motion and getting good simulator performance in a wide variety of scenarios was difficult. To mitigate this we decided to use Open Dynamics Engine (ODE) to simulate the system instead. In this project, we go one step further and delegate the entire simulation task to Gazebo.

Input The input that the simulator allows are wheel commands. Specifically, each wheel has two torque inputs, one for driving and one for steering. These signals are wrapped in a `vehicle_interface/msg/WheelCommand`, which is given in Listing 3.3.

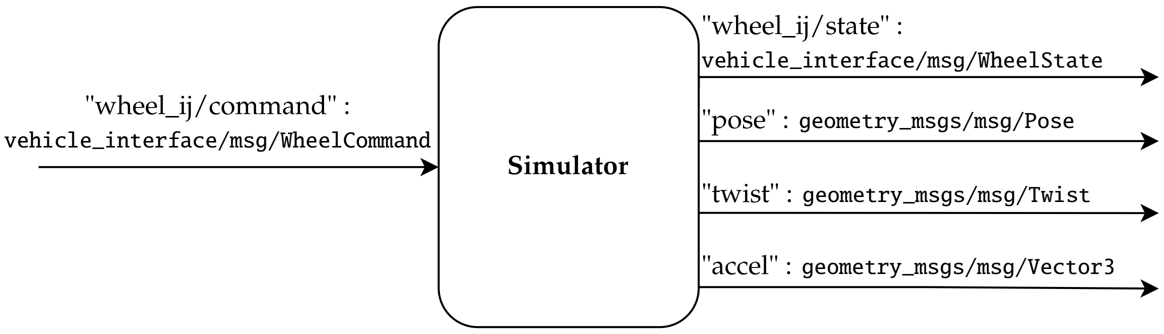
Output Since we neglect state estimation in this project, we use the simulator to publish ground truth data about the vehicle. The simulator publishes angular velocity, steering angle, and the steering angle rate for each in wheel in a `vehicle_interface/msg/WheelState` message, which is defined in Listing 3.4.

```
float64 drive_torque
float64 steer_torque
```

Listing 3.3: vehicle_interface/msg/WheelCommand

```
float64 steering_angle
float64 steering_angle_rate
float64 angular_velocity
```

Listing 3.4: vehicle_interface/msg/WheelState



"wheel_ij/command" : vehicle_interface/msg/WheelCommand
Simulator
"wheel_ij/state" : vehicle_interface/msg/WheelState
"pose" : geometry_msgs/msg/Pose
"twist" : geometry_msgs/msg/Twist
"accel" : geometry_msgs/msg/Vector3

Figure 3.3: Simulator module interfaces.

```
geometry_msgs/Point[] points
```

Listing 3.5: vehicle_interface/msg/Waypoints

The position and orientation of the vehicle is represented by the built in message type `geometry_msgs/msg/Pose`. We also publish the body-frame velocity and angular velocities in a `geometry_msg/msg/Twist` message. The acceleration of the vehicle is published as a `geometry_msgs/msg/Vector3` message. For more details of these messages, we refer to Appendix B.

The simulator module along with its interface topics and interface types is illustrated in Fig. 3.3.

3.4.2 Motion control

The motion control architecture translates high-level motion commands to actuator inputs. There are two modes of operation the architecture needs to support, manual control and waypoint based control.

Input The main inputs of the motion control system are the motion commands. In the waypoint mode, the controllers receives a `vehicle_interface/msg/Waypoints` message as defined in Listing 3.5. The `vehicle_interface/msg/Waypoints` message contains an ordered list of Cartesian coordinates, which is used to define a path the vehicle should follow. In manual control mode, the vehicle controller receieves a `vehicle_interface/msg/Guide` message, which is defined in Listing 3.6.

```
float64 course
float64 speed
```

Listing 3.6: vehicle_interface/msg/Guide

```
# Subscriber can use source to filter for relevant messages
string source
float64 yaw
float64 yawrate
```

Listing 3.7: vehicle_interface/msg/YawReference

The `vehicle_interface/msg/Guide` message says what direction the vehicle should move (course) and with what speed.

In both modes, we can also control the yaw of the vehicle. This is specified independently by a `vehicle_interface/msg/YawReference` message as defined in Listing 3.7. The yaw reference message contains yaw and yawrate. In addition it has a string attribute which says where the yaw-reference comes from. This is useful because the vehicle should support multiple yaw-control modes. In manual mode, it is typically more intuitive to control the derivative of a signal, so instead of controlling yaw directly, we control yawrate. But in waypoint mode, it might be most appropriate for the yaw to align with the path. By adding a source attribute to the yaw-reference and allowing the motion control system to filter out irrelevant yaw-reference sources, the system supports multiple yaw-control modes without modification.

We note that in addition to the special inputs given above, the motion control system also gets feedback signals of vehicle and wheel states.

Output The output of the motion control system are actuator inputs. In our case, the actuators are motor torques that we send to the simulator with `vehicle_interface/msg/WheelCommand` messages.

The motion control interfaces are illustrated in Fig. 3.4.

Architecture

The motion control system is responsible for taking a high level description of how the vehicle should move, and executing it. To do this, we divide it into three main layers. The upper layer is the guidance system, which is responsible for taking in a path and determining which direction and speed the vehicle should move with in order to follow the path. The middle layer is vehicle control, which is responsible for coordinating the wheels. The bottom layer is the wheel control layer, which is responsible for following the reference set by the vehicle control layer on a single

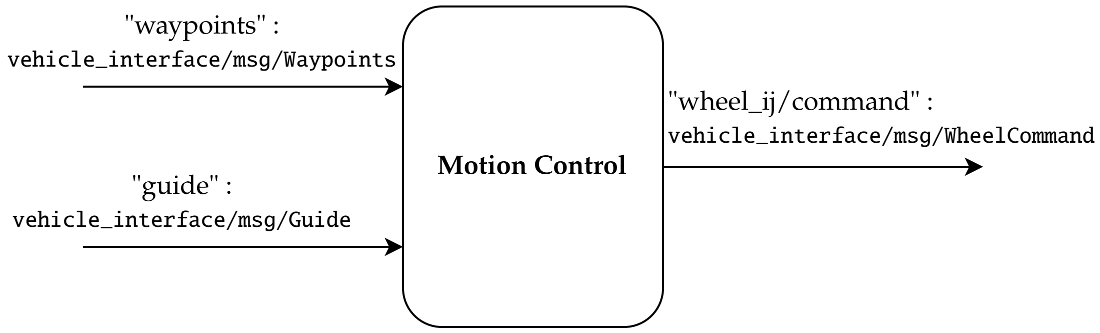

"waypoints" :
vehicle_interface/msg/Waypoints
"guide" :
vehicle_interface/msg/Guide
Motion Control
"wheel_ij/command" :
vehicle_interface/msg/WheelCommand

Figure 3.4: Motion control module interfaces.

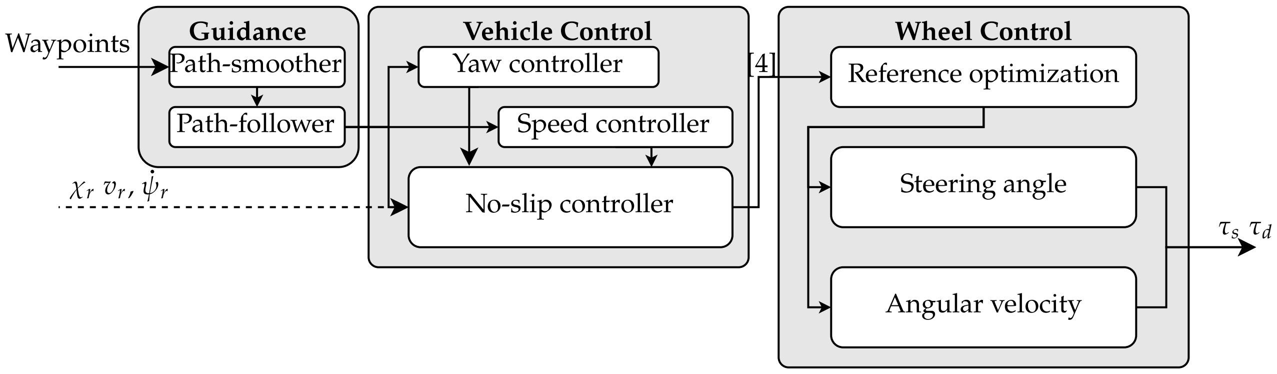


Figure 3.5: Motion control architecture.

wheel. This means that four copies of the wheel control layer run in parallel to control all the wheels of the 4WIS vehicle. The motion control architecture is shown in Fig. 3.5 with arrows showing the signal flow. More details on the motion control modules are given in Chapters 5 and 6.

3.4.3 State estimation

The state estimation module is responsible for determining the states of the vehicle. This involves sensor fusion to determine position, velocity, orientation, and wheel states. And it involves determining other derived quantities like the load on each wheel, or error detection. For this project, the state estimation module is added to show where it fits in a complete system, but it has not been a focus. Since we have access to ground truth data from the simulator, the main task of the state estimation module is to relay the ground truth data.

3.4.4 Operation

The operation module is the highest level in the system. It provides user-facing interfaces to the vehicle, and it is where we place business logic relating to specific operations that the vehicle should be capable of. For this project we have created two operations; waypoint following and manual mode, to demonstrate the capability of the system. More details on these are given in Chapter 7.

3.5 Bandwidth and time-scale separation

The GC architecture is a cascaded control system, where higher levels are used to set references for the lower levels. This structure is chosen mainly due to its simplicity and decoupled design, but it may affect the stability of the system. The vehicle control system assumes that when it sends a reference to the wheel control system, then that reference will be followed immediately. In general a control layer will neglect the dynamics of lower control layers because this assumption greatly simplifies control design. To ensure that this assumption is valid, we need to have bandwidth separation between the control layers.

Bandwidth separation means that the wheel controllers have higher bandwidth requirements than the vehicle controllers. Doing bandwidth separation properly requires system identification, which is beyond the scope of this project. In the wheel control layer we use a reference model for the wheel which has a bandwidth of $\omega_b \approx 3\text{rad}\,\text{s}^{-1}$. The vehicle control layer should then have a significantly lower bandwidth than $3\,\text{rad}\,\text{s}^{-1}$, otherwise the system may become unstable. This is achieved by using a yaw reference model with a quarter of the bandwidth and tuning the guidance system so it does not attempt too aggressive maneouvers.

In addition to bandwidth separation, the control systems should run on different time-scales. This is because the upper control layers have lower bandwidth requirements than the lower layers, so running them at the same frequency is computationally wasteful. This is particularly important in realtime, resource constrained systems. The frequency of each control layer should be related to the required bandwidth of the layer, but without system identification, there is no precise way of determining the frequencies. By trial-and-error, we found that running the vehicle control layer at 50 Hz and the wheel control layer at 100 Hz worked well in the simulated case studies, but the values are arbitrary. More work is required in order to establish the bandwidth requirements at each control level.

3.6 Detailed architecture

A detailed architecture of the system is given in Fig. 3.6. It shows all the essential modules that have been implemented and discussed above.

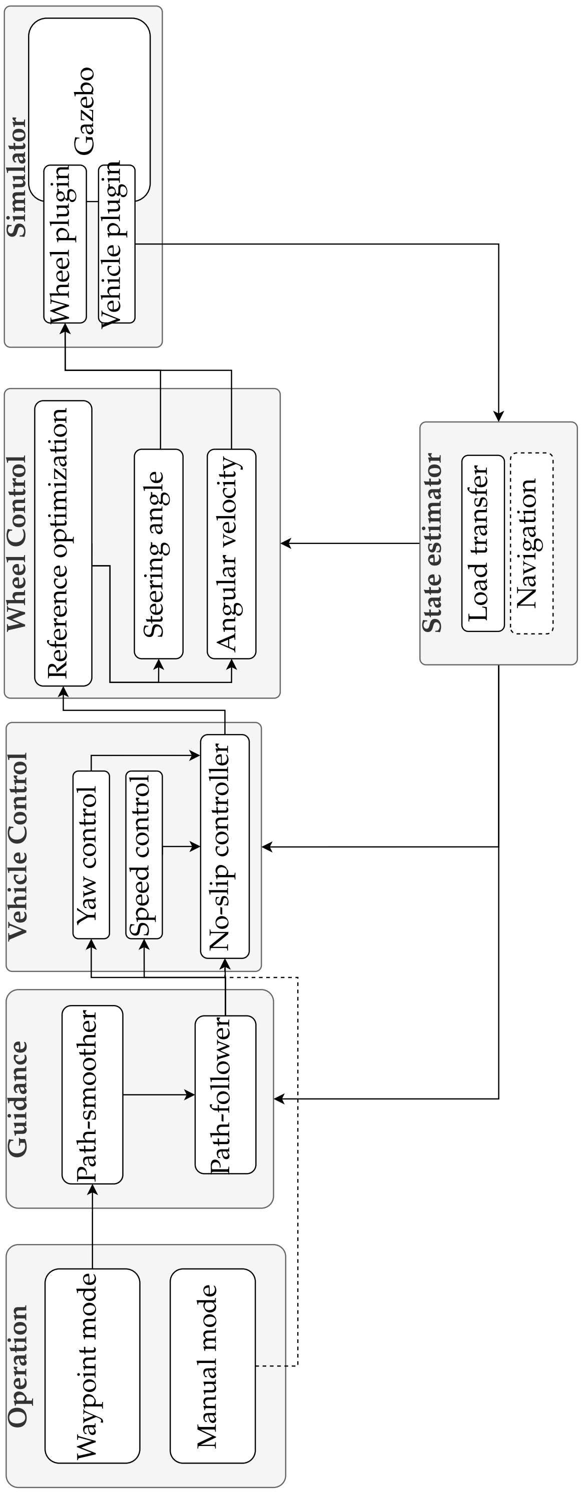


Figure 3.6: Detailed system architecture.

Simulator

4

The simulator is where we test the autonomous behaviour of the vehicle. In the specialization project [6] we developed a vehicle simulator from scratch using Python and ODE. Since we do not want developing and maintaining a simulator to be a focus of this project, we decided that it would be beneficial to migrate to the popular open source robotics simulator Gazebo. Doing this requires some setup as Gazebo is not immediatly capable of doing vehicle simulations. Instead it relies on a strong plugin system that allows users to modify and interact with the simulator by writing C++ plugins. In this chapter, we present design decisions behind the simulator, and we detail how to set up Gazebo to simulate the Autoagri vehicle. At the end we discuss some of the limitations our approach has, and potential solutions.

We note that there is little research using Gazebo for similar use cases. Biber et al. [3] used Gazebo to simulate a 4WIS agricultural robot called "BoniRob". Unfortunately, that project dates back to 2012, and we were unable to aquire source code for it.

4.1 Simulator assumptions

The main design principle behind the simulator is that it should be physically reasonable. The primary goal of this project is to develop and experiment with control systems. To test the control systems, it is not necessary to have accurate simulation. But we should identify the differences between the simulator and real-life and keep them in mind when we evaluate the results.

Since we only require physical reasonability, we implement the simulator as a rigid-body simulation. This means that the objects have mass and geometry, can collide and assert friction on one another, but they do not deform. Actually, the rigid-body design decision is more of a constraint than a decision. Rigid-body dynamics are easier to simulate and there exists openly available software to simulate it, whereas solid-body dynamics is much more difficult.

Most of the equations presented in Chapter 2 assumes flat terrain. To investigate the effect of unmodelled disturabances, we require that the simulator supports sloped terrain.

4.2 Gazebo terminology

A Gazebo simulation consists of several elements, and they are structured neatly into a hierachy, which we illustrate in Fig. 4.1. The top element is world, which contains 5 distinct child elements. The scene element describes how the models and lights are placed at the start of the simulation. The physics element describes the settings for the physics engine, like which engine to use, stepsize and real

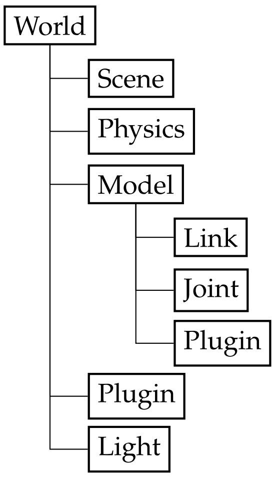


Figure 4.1: Gazebo terminology and hierarchy.

time factors. The model element is the primary element we are concerned with. With the exception of lights, every object in the simulator is described by a model. There can be many model elements in a simulation. The world plugin tag describes plugins that are attached to the world. These can modify world properties like physics parameters, reset the simulation simulation, spawn or delete models, and more. The light element describes the placement of lights which illuminate the scene.

4.2.1 Models

As stated above, the model element are the objects we simulate, so it is where we implement the vehicle and specify its physical properties. A model consists of one or several links, and the links are held together by joints. Note that these terms (link and joint) have the same meaning as in the broader robotics litterature (e.g. [34]). In addition to this, a model may have plugins which allows the model to be controlled by external commands. A detailed model structure is shown in Fig. 4.2.

4.2.2 Links

A link is a rigid body. It has inertial properties like mass and an inertia matrix. To simulate interactions with other objects, it also has a collision geometry, which defines an impenetrable region. To visualize the model, it has a separate visual geometry. The main reason to separate collision and visual geometry, is that collision detection and resolution is computationally expensive, so it is advantageous to use simple collision geometry primitives, like spheres, boxes, and cylinders.

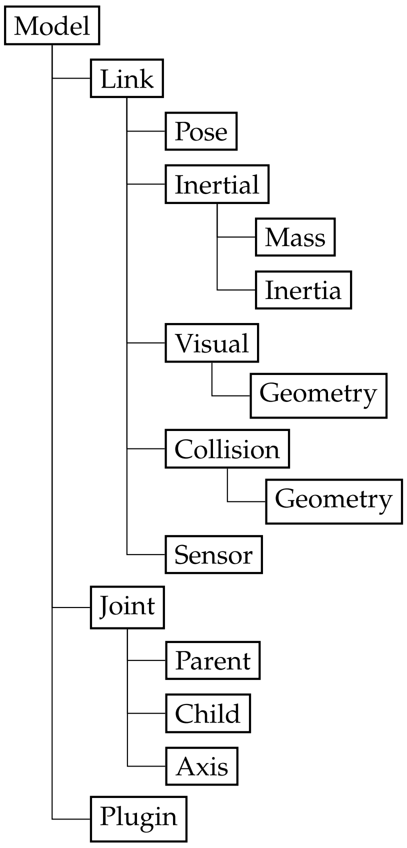


Figure 4.2: Model element hierarchy.

4.2.3 Joints

Joints are used to connect links together. Behind the scenes, they are used to create constraints that the simulation must maintain. They also provide a natural way to add actuators to the simulation. There is a wide variety of joint types used in robotics simulation, and [24] gives a comprehensive list in the context of ODE, which is Gazebo's default physics engine. To implement a basic vehicle, we only need two joint types; rigid and continuous.

Rigid joints

Rigid joints are rigid connections between links. When two links are rigidly connectected, they behave as a single link, meaning that they cannot rotate or move relative to each other.

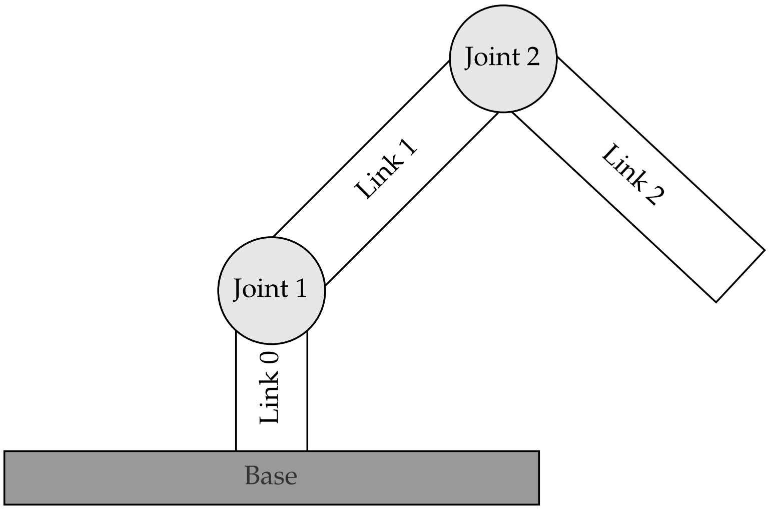


Figure 4.3: Model illustration.

Continuous joints

Continuous joints are joints with a rotional axis. The position and orientation of the rotational axis is defined in relation to the parent link. This means continuous joints are a natural way to implement wheel and steering mechanisms. If we want to constrain the steering angle, then revolute joints are equivalent to continuous joints except with limited range of motion.

4.3 Gazebo setup

To create the Autoagri vehicle in Gazebo, we need to define its properties in the model format shown in Fig. 4.2. This is done using unified robot description format (URDF), which is an XML based file format. Since each wheel is identical, it is natural to create a submodel for a single wheel that we can duplicate and place on a chassis model.

4.3.1 Wheel model

The wheel is created using three links. The first link is the wheel base, which is where the wheel model is rigidly attached to the chassis. The steering link is the part of the wheel that rotate when the wheel steers. It is connected to the base link via a continuous joint that allows rotation about the vehicle z-axis. To drive the wheel, a tire link is created and attached to the steering link with a continuous joint that allows rotation about the y-axis. This is illustrated and related to autoagri vehicle in Fig. 4.4. The tire link is created using a cylinder, which is a primitive

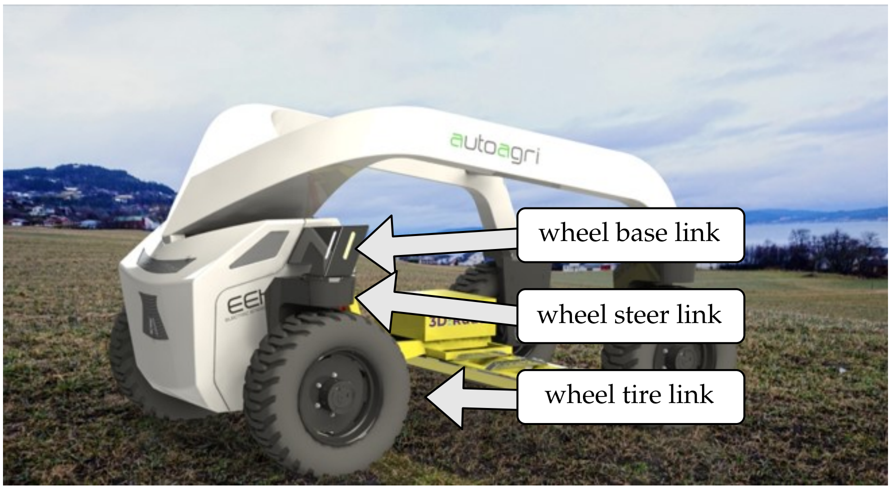


Figure 4.4: Wheel links and their physical interpretation.

shape in Gazebo. As a small graphical improvement we also add a 3D mesh to the visual tag, but this does not affect the physics. Note that the cylinder, like all other objects in Gazebo, is a rigid body. This means that properties like rubber and terrain deformation is not simulated.

4.3.2 Chassis model

The chassis model is simpler as it does not have any moving parts. To model the chassis, we use primarily a single link. In addition to this we add an optional tool link which is rigidly attached to the chassis.

4.3.3 Complete model

The complete model is created by duplicating the wheel models and creating rigid joints to connect them to the chassis where the wheels should be positioned. The final link and joint structure is shown in Fig. 4.5. A screenshot of the complete model is shown in Fig. 4.6. To verify that the joints were placed correctly, we made the model transparent and enabled joint visualization. This is shown in Fig. 4.7. There it can be seen that each wheel has its own coordinate frame with a small yellow circle indicating the the wheel rotates about the green axis, which is the wheel y-axis.

4.3.4 Model plugins

By default the model presented above does not do anything. It is simulated in the physics engine, but we have no way of interacting with it through code. This

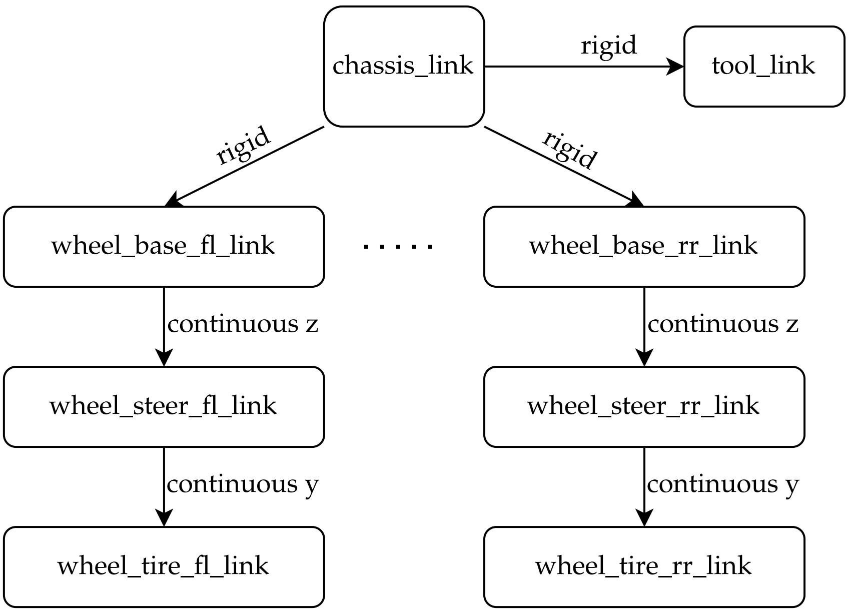


Figure 4.5: Link and joint hierarchy in vehicle model.

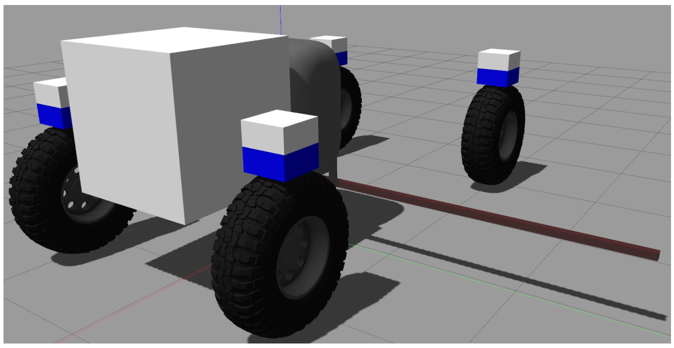


Figure 4.6: Complete model.

is where plugins come into play. Plugins, or more specifically, model plugins, are arbitrary pieces of C++ code that are attached to a model and called upon by Gazebo on runtime. The Gazebo model application programming interface (API) allows us to query properties of the model like position, velocity, link states, joint states, and more, and apply forces and torques dynamically. Since the plugins are written in C++, it is straight forward to integrate them with ROS2.

Keeping with the structure presented above, we create two plugins; one to interact with a single wheel, and one to read out body states.

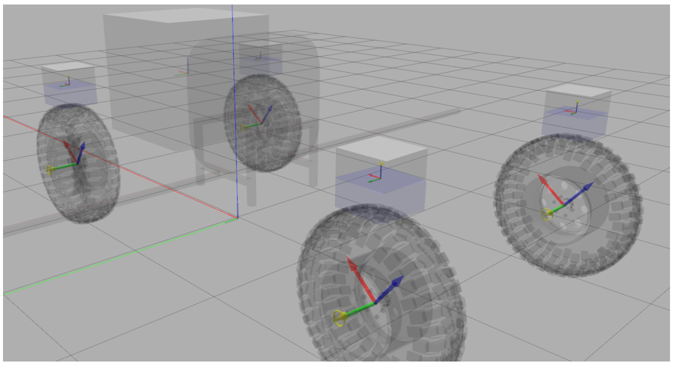


Figure 4.7: Model with joint visualization.

Wheel plugin The wheel plugin is responsible for setting torques on the steer and drive joints on a single wheel. It does this by creating a ROS2 node and subscribing to `vehicle_interface/msg/WheelCommand` messages from the wheel command topic, as illustrated in Fig. 3.3. It forwards the torque messages to their respective joints in the wheel model using the Gazebo API, which has a method for applying torque to a joint. In addition to setting torques, the plugins reads out the wheel states using the Gazebo API. These wheel states are published using `vehicle_interface/msg/WheelState` messages.

Vehicle plugin The vehicle plugin is loaded to the root of the vehicle model. It does not affect the simulator directly, but it is used to publish the vehicle states out of Gazebo and into the ROS2 ecosystem. It contains publishers for pose, twist, and acceleration of the vehicle. These are published using the standard library types `geometry_msgs/PoseStamped`, `geometry_msgs/TwistStamped`, and `geometry_msgs/Vector3Stamped` respectively, which are given in Appendix B.

4.4 Limitations

The simulator developed in this project has several limitations. The main limitations are what we can call interface realism. We interact with the simulator by sending torque signals to the motors. This is reasonable as a signal with physical meaning, but presumably the motors on the vehicle do not allows us to control torque directly. An improvement to the simulator would be to investigate what the inputs of the motors are, and characterize their response curves. Additionally, the simulator outputs ground-truth data, but this is problematic because the quality of the data used in the control system is then unreasonably accurate and noise-free. To make the simulator more realistic, we could emulate sensors such as wheel encoders and GPS.

Another limitation is the lack of terrain. It is only capable of simulating flat or inclined surfaces, but this is not representative of a real field. The problem is that in order to implement terrain, we also need to implement suspensions or other vertical dynamics. With rigid-body simulation, the vehicle is incapable of driving over small objects, because doing so requires several of the wheels to come off of the ground.

There are several small improvements we can make to the simulator to make it more realistic. One is to add real inertial parameters. Currently, the vehicle properties are approximated using geometric primitives from Gazebo such as boxes and cylinders. Using the CAD software the vehicle is designed in, we can obtain the inertia matrix and center of mass which would make the simulation more physically accurate. Another small improvement is to use a better friction model. By default, Gazebo uses a version of static and kinetic friction which is optimized for computational efficiency. Overriding this and implementing a vehicle specific model such as the Burckhardt model would improve the simulator accuracy.

We suspect implementing more advanced effects such as soft-body dynamics and terrain deformation is not possible with Gazebo. This means that there are many aspects that cannot be simulated accurately.

Despite the limitations, we believe the simulator achieves its design goal of physical reasonability. If a control system has no chance of working, we should be able to discover that using the simulator, but after simulator testing, the control system needs rigorous testing on the real vehicle. The lack of sensor and interface emulation is unfortunate, and if we were going to develop this simulator further, those are the most important things we would tackle.

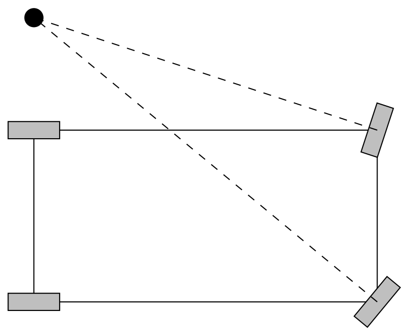
5 Control Systems

The control system is the system which translates reference signals into actuator inputs. Depending on the mode of operation the control system will either recieve reference from the guidance system, or directly from the operation layer. From an engineering perspective it needs to be robust and fault-tolerant. It should also be verifiable. This means that it should either be provably correct, or it should use standard control techniques that have been applied to similar problems in the past. An overview of the control system and its submodules were given in Section 3.4.2. In this chapter, we perform a litterature review of how similar vehicles have been controlled in the past, and then we present each module of the control system in detail, starting at the bottom with the wheel control layer and working upwards. We present and discuss results after each control layer has been presented.

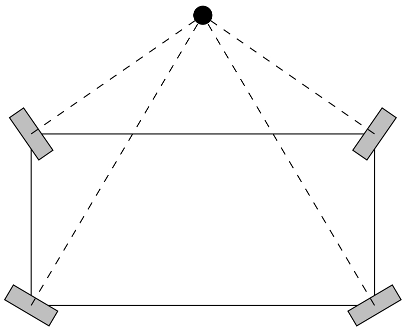
5.1 Litterature review

There exists relevant research, specifically on path-following and trajectory-tracking for 4WIS vehicles. Yim [39] compares different steering modes available to 4WIS vehicles and concludes that by using all the wheels actively, we can improve control performance over just using front-wheel steering. Setiawan et al. [32] presents a guidance system for 4WIS vehicles based on bicycle vehicle model, but they restrict their attention to zero-sideslip manouvers, meaning the vehicle is always facing in the driving direction. In [31], they improve this design to allow parallel steering mode, meaning all the wheels are steered to the same angle. Chen et al. [7] designs a path-following controller based on linear optimal control, but their setup requires small steering angles, and they make the rear-wheel steering angle a linear function of the front wheels. This was done in order to improve steering stability, but it makes the vehicle functionally equivalent to a typical front-wheel steered vehicle.

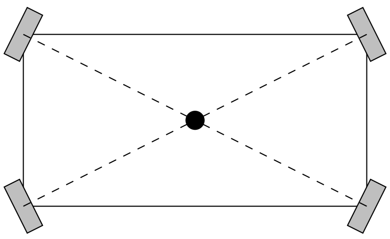
Maybe the most relevant prior research is given by Ye et al. in [38]. They design a four-mode control system for a 4WIS bin managing robot. The robot they worked with was required to operate in tight confined indoor spaces. They argue that by separating the control task into the four distinct modes illustrated in Fig. 5.1, and designing a control system which selects the best mode for each operation, they could achieve the best controllability and performance.



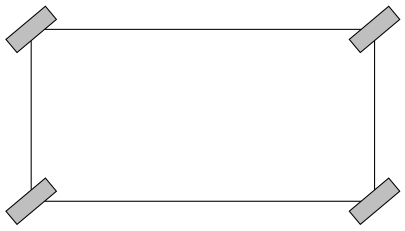
(a) Ackermann steering. Emulates typical front wheel driven vehicles.



(b) Active-front-active-rear. Front and rear wheels are activated symmetrically so ICR lies on the centerline.



(c) Spinning. Vehicle rotates about its center point.



(d) Crab. All wheels are aligned and vehicle can move in an arbitrary direction without changing its orientation.

Figure 5.1: Steering modes for 4WIS vehicles used by [38]. Black dot is ICR.

5.2 Wheel control

5.2.1 Steering angle sliding-mode control

The steering angle is modeled by (2.20), which we restate as

$$J\ddot{\delta} + M_f = \tau_s \tag{5.1}$$

where $J = J_{w,z}$ is the wheel inertia about the z-axis. We want to control the steering angle δ to follow a reference input δ_r. The challenge is that the friction torque is highly nonlinear and dependent on surface and tire parameters, $\dot{\delta}$, and τ_s. It has been demonstrated that we cannot compansate for friction using standard proportional-integral-derivative (PID) control, as it will often lead to limit cycles and track errors [37]. Knowing this, we tried in the specialization project [6] to compansate by estimating M_f offline and using feedforward, but this did not work probably because the offline estimate was significantly different from the real (online) value. We suspect that using an adaptive approach to estimate M_f online

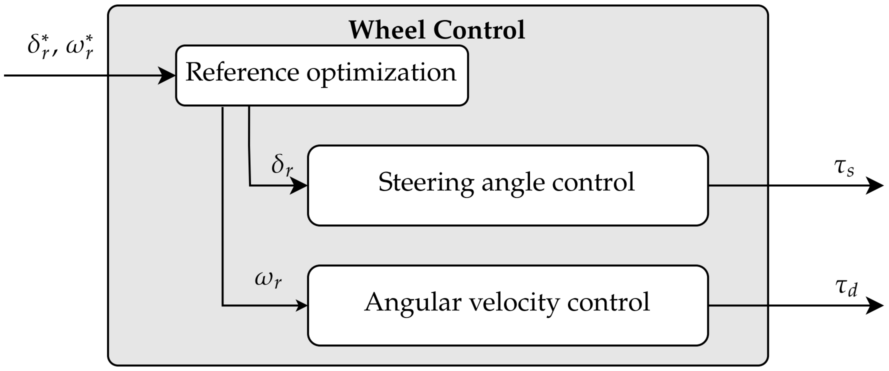


Figure 5.2: Wheel control signals.

could work, but this remains an open question. A problem with adaptive schemes is that they are hard to make robust, and even simple adaptive laws may become unstable in the presence of unmodelled disturbances [16]. Because it is important that the steering angle controller is robust, we did not consider adaptive control to be a viable option.

While estimating M_f accurately is hard, it is relatively straight forward to bound it. Assume we have determined a bound so we can say

$$\left|M_f\right| \leq M \tag{5.2}$$

We can use this bound to design a sliding-mode control (SMC) scheme for the steering angle. Say we want to converge on a constant reference signal δ_r. The error states are $e_1 = \delta - \delta_r$[1] and $e_2 = \dot{e}_1$. The error dynamics are given by

$$\dot{e}_1 = e_2 \tag{5.3}$$

$$\dot{e}_2 = -J^{-1}M_f + J^{-1}\tau_s \tag{5.4}$$

With SMC, we control the system to a sliding surface which we can prove to be asymptotically stable. The surface we want to control to is $s = 0$ in

$$s = e_2 + \lambda e_1 \tag{5.5}$$

If we can reach $s = 0$, then we have $0 = e_2 + \lambda e_1 = \dot{e}_1 + \lambda e_1$ which means e_1 (and $e_2 = \dot{e}_1$) converges exponentially on the origin. Consider the Lyapunov function

[1]Since this is a difference of angles, one may want to use the smallest signed angle as an error state, and then this expression would be wrong. However using smallest signed angle is only appropriate if the steering angle is unconstrained. If the steering angle is constrained, then we argue using the smallest signed error is innapropriate because we need to be able to distinguish between $0, 2\pi, -2\pi$ and other similar angles.

and its derivative

$$V = \frac{1}{2}s^2 \tag{5.6}$$

$$\dot{V} = s\left(-J^{-1}M_f + J^{-1}\tau_s + \lambda e_2\right) \tag{5.7}$$

Using $\left|M_f\right| \leq M$ we can write

$$\begin{aligned}
\dot{V} &= J^{-1}s\left(J\lambda e_2 - M_f\right) + J^{-1}s\tau_s \\
&\leq J^{-1}|s|\left|J\lambda e_2 - M_f\right| + J^{-1}s\tau_s \\
&\leq J^{-1}|s|\left(J\lambda|e_2| + \left|M_f\right|\right) + J^{-1}s\tau_s \\
&\leq J^{-1}|s|\left(J\lambda|e_2| + M\right) + J^{-1}s\tau_s \\
&= J^{-1}\left(|s|\rho(e_2) + s\tau_s\right) \\
&= J^{-1}s\left(\operatorname{sgn}(s)\,\rho(e_2) + \tau_s\right)
\end{aligned} \tag{5.8}$$

where we have defined

$$\rho(e_2) = J\lambda|e_2| + M \tag{5.9}$$

From (5.8), we can see that by using

$$\tau_s = -\left(\rho(e_2) + \beta_0\right)\operatorname{sgn}(s) \tag{5.10}$$

where $\beta_0 > 0$, we obtain

$$\dot{V} \leq -J^{-1}|s|\beta_0 \tag{5.11}$$

Since $\dot{V} < 0$ for all $s \neq 0$ and V is radially unbounded, the system is globally asymptotically stable. Note that it is globally asymptotically stable even though it is an angle controller. One could argue that the steering angle controller should operate on error states defined with the smallest signed angle, which maps angles to the range $(-\pi, \pi]$, but we believe this is not appropriate in this case. This is because the real AutoAgri vehicle has a constrained steering angle, so we need to be able to distinguish between 180 deg and −180 deg. A consequence of this is that if the steering angle is currently −180 deg, and we command it to go to 180 deg, an angle which is functionally equivalent, then the wheel will make a full rotation using this controller. We repeat that this is by design, and in Section 5.2.4 we show how we avoid the problematic behaviour described above.

In addition to being globally asymptotically stable, the origin is finite time stable, meaning that we will reach the sliding surface in finite time. This can be seen by realising that $|s| = \sqrt{s^2} = \sqrt{2}V^{0.5}$, and rewriting (5.11) as

$$\dot{V} + \sqrt{2}J^{-1}\beta_0 V^{0.5} \leq 0 \tag{5.12}$$

Appendix A.1 outlines a proof that this implies that the system will converge to $V = 0$ within a finite time t_r (dependent on the initial conditions).

Using (5.10) is in theory fine, but in practice it will lead to high frequent oscillations about the sliding surface, called chattering. This is caused by the $\operatorname{sgn}(s)$ discontinuity around $s = 0$, and is a known problem of SMC (e.g. [13, 18]). To mitigate chattering, a common approach is to replace $\operatorname{sgn}(s)$ with a soft sign that is continuous about the origin. We use

$$\operatorname{softsign}(s) = \begin{cases} s/\epsilon & \text{for} \quad |s| \leq \epsilon \\ \operatorname{sgn}(s) & \text{otherwise} \end{cases} \tag{5.13}$$

so the final sliding mode control law becomes

$$\tau_s = -\left(\rho(e_2) + \beta_0\right) \operatorname{softsign}(s) \tag{5.14}$$

Khalil [18] discusses convergence and stability for SMC with this modification. They show that s will stabilize to $|s| \leq \epsilon$ in finite time, but there is no guarantee that s converges to zero. This means that the error may not converge to zero either as the sliding mode becomes $\dot{e}_1 = -\lambda e_2 + s$ where $|s| \leq \epsilon$. In theory we can make the error arbitrarily close to 0 by making λ large or ϵ small, but this will bring back the chattering problems. With SMC we have to settle for a compromise between some steady state error and chattering.

5.2.2 Robust rate-limited steering angle control

A problem that can occour with the SMC is that it is very aggressive, meaning it will command unreasonably large actuator torques. To combat this, we develop another robust control strategy for the steering angle rate $\dot{\delta}$. This can be integrated with a proportional controller to control the stering angle δ.

Robust steering angle rate control Consider again the steering angle system

$$J\ddot{\delta} + M_f = \tau_s \tag{5.15}$$

and assume that we can bound the steering resistance such that $\left|M_f\right| \leq M$. We want to control $\dot{\delta}$ to track the reference $\dot{\delta}_r$. Define the error signal $e = \dot{\delta}_r - \dot{\delta}$ and write the Lyapunov function as

$$V = \frac{1}{2} e^2 \tag{5.16}$$

We can then put an upper bound on the derivative using a similar reasoning as with the sliding mode controller.

$$\begin{aligned}
\dot{V} &= e\dot{e} \\
&= e\left(\ddot{\delta}_r - (J^{-1}\tau_s - J^{-1}M_f)\right) \\
&= J^{-1}e\left(J\ddot{\delta}_r + M_f\right) - J^{-1}e\tau_s \\
&\leq J^{-1}|e|\left(J\left|\ddot{\delta}_r\right| + \left|M_f\right|\right) - J^{-1}e\tau_s \\
&\leq J^{-1}|e|\left(J\left|\ddot{\delta}_r\right| + M\right) - J^{-1}e\tau_s \\
&= J^{-1}e\left(J\left|\ddot{\delta}_r\right|\mathrm{sgn}\,(e) + M\mathrm{sgn}\,(e) - \tau_s\right) \qquad (5.17)
\end{aligned}$$

Selecting τ_s so that $\dot{V} < 0$ for all $e \neq 0$, we can use

$$\tau_s = \left(J\left|\ddot{\delta}_r\right| + M + \beta_0\right)\mathrm{sgn}\,(e) \qquad (5.18)$$

where β_0 is a margin used to ensure $\dot{V} < 0$. Because $\dot{V} \leq -J^{-1}\beta_0|e| = -k|V|$ for some $k > 0$ the system is finite time stable, so $\dot{\delta}$ will converge on $\dot{\delta}_r$ in a finite reaching time. Refer to Appendix A.1 for details.

The structure of this controller is essentially identical to a sliding mode controller. Thus we expect the same chattering problems that we discussed in Section 5.2.1 to reapear here, and we expect that replacing $\mathrm{sgn}\,(e)$ with softsign(e) is beneficial here as well. The final robust rate control law is then

$$\tau_s = \left(J\left|\ddot{\delta}_r\right| + M + \beta_0\right)\mathrm{softsign}(e) \qquad (5.19)$$

Rate-limited steering angle control Using the controller proposed above, we can control the steering angle rate in the presence of complex friction forces. To control the steering angle, we use a rate limited proportional control law.

$$\dot{\delta}_r = \mathrm{sat}\left(K_p(\delta_r - \delta), \dot{\delta}_{w,\mathrm{limit}}\right) \qquad (5.20)$$

The reason to use rate limited control law, is that we can explicitly set the maximum steering rate. In theory this will give worse tracking performance compared to the sliding mode controller of Section 5.2.1, but we expect that it will give more reasonable results when applied in practice.

5.2.3 Angular velocity control

The angular velocity of a wheel is modeled as

$$J\dot{\omega} + rF_f = \tau_w \qquad (5.21)$$

where $J = J_{w,y}$ is the inertia about the driving axis. Our goal is to control ω to a constant reference ω_r. To do this we will expand the state space to include the wheel velocity and a dynamical friction model using the quarter vehicle model as presented in Section 2.3.4, which we restate here.

$$\dot{v} = \frac{1}{m}F_f \tag{5.22}$$

$$\dot{\omega} = -\frac{r}{J}F_f + \frac{1}{J}\tau_d \tag{5.23}$$

$$\dot{F}_f = k\frac{r\omega - v}{l} - \frac{r|\omega|}{l}F_f \tag{5.24}$$

As noted in Section 2.3.4, the system is passive, which is good news since then using a passive control law in a feedback connection will result in a passive closed loop system [18]. Note that in steady state conditions, we have $\dot{v} = 0 \Rightarrow F_f = 0$. This means that a proportional controller is sufficient if we want to converge on a constant reference ω_r. Thus the angular velocity control law is

$$\tau_d = K_p e_\omega \tag{5.25}$$

$$e_\omega = \omega_r - \omega \tag{5.26}$$

If there is an unmodelled disturbance which causes some driving resistance, then the controller above will lead to steady-state error, so it stands to reason that we should implement an integral effect to compensate for this. We decided against this, because we implement integral effects in the control layer above instead.

5.2.4 Reference model and optimization

Using the control systems given above, we can control the steering angle δ and the angular velocity ω. This is in principle all that is needed from a wheel controller, but if we want to apply this in practice there are some additional aspects we have to deal with.

The steering angle may be limited so that it cannot turn more than (for instance) a full rotation each way, that is $|\delta| \leq 2\pi$. Another related issue is that the steering angle and angular velocity are coupled, since driving one way is equivalent to reversing the opposite way. Mathematically we can say that the state pair (δ, ω) is equivalent to $(\delta + k\pi, -\omega)$ for any integer k. Of course making an additional full turn with the wheel will also give an equivalent state $(\delta + k2\pi, \omega)$ for any integer k. This equivalency class between the states, means that we can improve the wheel control system selecting the equivalent input which is "closest" to the current wheel state. To do this we assign a cost to moving from the current wheel state to a given reference state. We think it is desirable to minimize wheel turning, so we write the cost as the smallest absolute steering angle difference.

$$C(\delta, \delta_r) = |\delta - \delta_r| \tag{5.27}$$

When the wheel controller is given a reference (δ_r^*, ω_r^*) to follow, it first solve a small discrete optimization problem to find the equivalent reference with the lowest cost.

Given a reference (δ_r^*, ω_r*) and a state (δ, ω). The closest equivalent reference with the same angular velocity is

$$\delta_r^1 = \delta + \mathrm{ssa}(\delta_r^* - \delta) \tag{5.28}$$

$$\omega_r^1 = \omega_r^* \tag{5.29}$$

Where the smallest signed angle function $\mathrm{ssa}(\theta)$ computes the smallest signed equivalent angle of θ. A trigonometrically inspired implementation is

$$\mathrm{ssa}(\theta) = \mathrm{atan2}(\sin\theta, \cos\theta) \tag{5.30}$$

If we allow reversing the wheels, then two more references should be considered, namely

$$(\delta_r^2, \omega_r^2) = (\delta_r^1 + \pi, -\omega_r) \tag{5.31}$$

$$(\delta_r^3, \omega_r^3) = (\delta_r^1 - \pi, -\omega_r) \tag{5.32}$$

To find the optimal reference, we compute the costs $C_i = C(\delta, \delta_r^i)$, and select the reference with the lowest cost. In a real application, the steering angles may be constrained, which can be implemented by extending the cost function to include a high cost of exceeding the constraints. Once the optimal references (δ_r^*, ω_r^*) are determined, we use reference models to ensure that the signals sent to the controllers (δ_r, ω_r) are feasible. The steering angle references are filtered through a second-order reference model, as recommended by [13], given by

$$\frac{\delta_r}{\delta_r^*}(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2} \tag{5.33}$$

where ζ is the damping ratio and ω_0 is the natural frequency. The angular velocity references are filtered through the first-order reference model

$$\frac{\omega_r}{\omega_r^*} = \frac{1}{1 + Ts} \tag{5.34}$$

where T is the time constant.

5.2.5 Chattering statistic

Sections 5.2.1 and 5.2.2 proposes two controllers which both experience chattering. To evaluate them against one another, we need a way to measure chatter, but there are no standard ways to do this. With SMC, one can look at the standard-devation of the sliding surface s since it should oscillate about 0, but this is not applicable to the robust rate controller. Because of this, we propose a controller-independent way to measure chatter. It operates on the control signal u directly and it has the same unit as u, which gives it a physical interpretation.

Given a discrete control signal u_i for $i = 1, \ldots, N$, compute the smoothened control signal u_i^s by smoothing the signal with a normalized Hanning window. The Hanning window is defined in [8] by

$$h_i = 0.5 - 0.5\cos\left(\frac{2\pi i}{M-1}\right) \quad \text{for} \quad 0 \leq i \leq M-1. \tag{5.35}$$

The normalized Hanning window is then given by

$$w_i = \frac{h_i}{\sum_{j=0}^{M-1} h_j} \tag{5.36}$$

To compute the smoothened control signal, we convolve u_i with w_i, using

$$u_i^s = \sum_{j=0}^{M-1} u_i w_{i-j} \tag{5.37}$$

where $u_i = 0$ for $i < 0$ and $i > N$. The choice of Hanning window versus any other triangular window function is is an arbitrary choice. We look at the difference between the smoothened and original signal

$$\Delta u_i = u_i - u_i^s \tag{5.38}$$

When the difference is small, the original signal is approximately constant. To estimate the chatter in the original signal we compute the standard deviation of Δu_i.

$$\text{chatter} = \text{std}(\Delta u_1, \Delta u_2, \ldots, \Delta u_N) \tag{5.39}$$

We have not been able to find this or any similar metrics in the litterature. The idea is that chattering is related to the variance/standard deviation of the signal, but the signal has a time-varying mean, so we cannot use the typical formulas for standard devation. By smoothening the signal, we create local estimates of the mean, so we assume $\Delta u_i = u_i - u_i^s$ is a zero-mean signal, which allows us to compute its standard devation using statistical formulas. More analysis and work is required in order to verify that this is a good measure of chatter, but from experience it tends to give results around the same order of magnitude as the visible chatter in the signal.

5.2.6 Parameters

The wheel controller parameters are given in Table 5.1. The parameters were mostly arrived at through trial and error. We believe the parameters are set within the correct order of magnitude, but the exact value are for the most part arbitrary.

Table 5.1: Wheel controller parameters.

(a) Angular velocity controller parameters.

K_p	250

(b) Sliding mode steering angle controller.

λ	5
ϵ	5
M	$2F_z$
β_0	0.1

(c) Robust-rate steering angle controller.

K_p	5
$\dot{\psi}_{\max}$	60 deg/s
ϵ	5
M	$2F_z$
β_0	0.1

(d) Reference models.

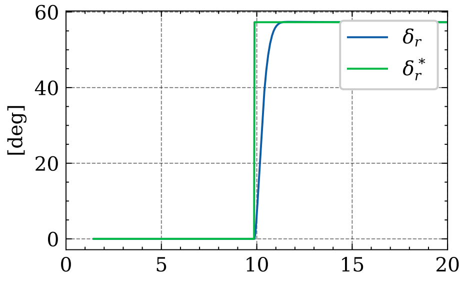
T	1
ω_0	4
ζ	0.9

The exceptions are the maximum steering resistance M and the reference model parameters. The steering resistance M was determined by using the Gazebo GUI to incrementally add torques to the steering angle until it started to move. We then expressed this in terms of the load on the wheel and found that it was almost twice the load, so $M = 2F_z$. The reference model parameters, particularly ω_0 and ζ for the steering angle, have a big effect on the performance of the system once the vehicle control layer is involved. From experimenting, we found that setting ω_0 lower than 4 could lead to oscillations, probably due to lack of bandwidth separation. This is discussed more in Section 6.5.3.

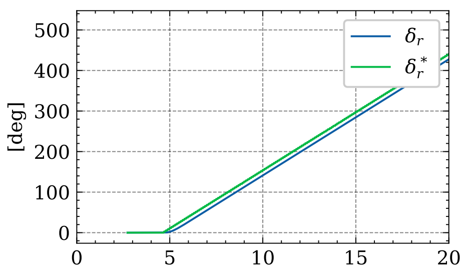
5.2.7 Results

In the preceding sections, we have developed two steering angle controllers and an angular velocity controller. In this section we compare the steering angle controllers against one another, and we evaluate the angular velocity controller.

To have a fair comparison, we created two cases for the steering angle controllers to track. The first case is a step response from 0 to 1 radians, and the second case is a linear ramp function with slope $0.5\,\mathrm{rad\,s^{-1}}$, meaning the wheels will do approximately one rotation every 13 seconds. The reference steering angles are shown in Fig. 5.3. Note that the ramp doesn't start immediately, which is done to give the system some time to initialize. The goal of these tests are to evaluate specific aspects of the controllers. The step-response is primarily used to test convergence on a constant reference. Good performance will look like fast convergence, and when it has converged, the controller will use little control input to stay on the reference. With the ramp-response, we want to see how the system tracks a time-varying reference, and we are particularly interested in the mean



(a) Case 1: step function reference signal.



(b) Case 2: ramp function reference signal.

Figure 5.3: Reference signal for steering angle controller cases. δ_r^* is the reference signal and δ_r is the reference model signal which is sent to the controller.

error and chattering statistics.

Sliding-mode controller

The SMC is tested on the references shown in Fig. 5.3. The step response results are given in Fig. 5.5 and Table 5.2. In Figs. 5.4a and 5.4b, we can see that the error quickly converges to zero when the reference input is constant. During the transient phase, there was a peak error of 5.967 deg, and the final error was 0.021 deg.

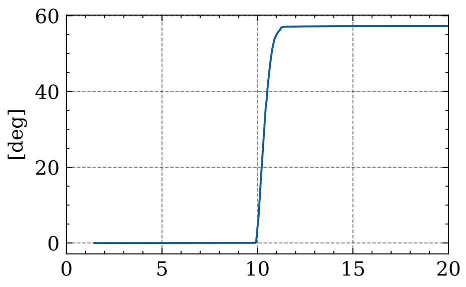
Based on the step response case, the sliding-mode controller is able to converge on a constant reference signal quickly. The results for the time-varying case are given in Fig. 5.7 and Table 5.3. From Fig. 5.6c, we can see that there is significantly more chatter compared to the step response, and this is evident by the chatter metric which grew from 184.449 N m to 428.484 N m.

Robust rate-limited controller

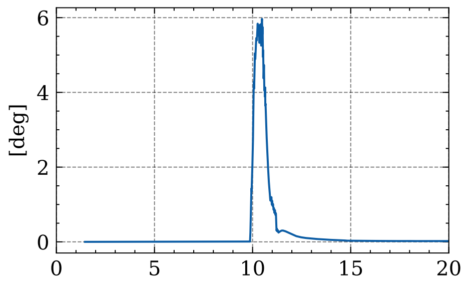
Testing the robust rate-limited controller on a step response, we obtain the results given in Fig. 5.9 and Table 5.4. The steering angle converges in about 2 s, and there is little chatter in the commanded torque, especially after converging. Testing the controller on the time-varying reference case, we get the results presented in Fig. 5.11 and Table 5.5. With time-varying reference, the chatter increases substantially from 23.320 N m to 476.903 N m. The robust rate-limited controller is able to maintain track of the time-varying reference with mean error 2.331 deg.

Steering angle controller comparison

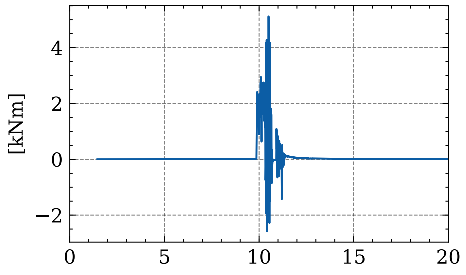
Section 5.2.1 and Section 5.2.2 presents two robust steering angle controllers. Since we can only use one of them, we evaluate them against one another in this section. The peak error of the robust rate-limited controller grows to 17.417 deg, which is



(a) Steering angle.



(b) Steering angle error.



(c) Steering torque.

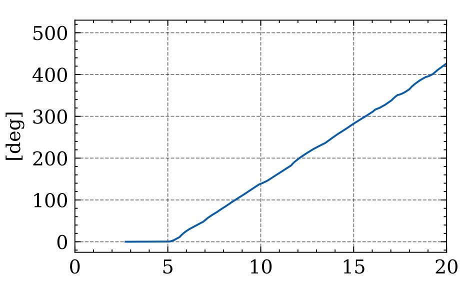
Figure 5.5: Front-left wheel step plots with sliding mode controller.

Table 5.2: Front-left wheel step performance metrics with sliding mode controller.

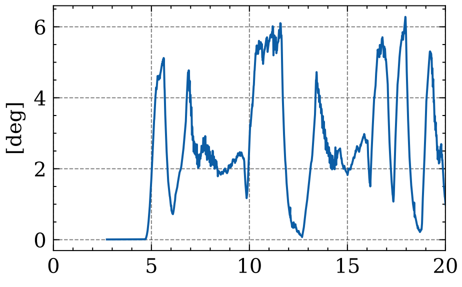
Metrics	
Mean absolute error [deg]	0.263
Stationary deviation [deg]	0.021
Chatter [Nm]	184.449
Peak torque [kNm]	5.120
Mean torque [kNm]	0.078

significantly higher than the 5.967 deg that the sliding mode-controller gave. The reason for this is that the rate-limited controller saturates when given such a large step input, because the reference model does not have a rate limitation. This can be clearly seen in Fig. 5.12, where the robust rate-limited controller saturates at $\dot{\psi}_{\max} = 60 \deg \mathrm{s}^{-1}$.

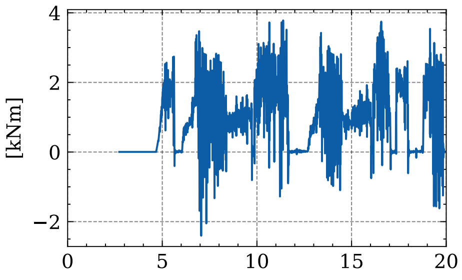
Based on the results above, we see no significant difference between the robust rate-limited controller and the SMC. Theoretically, they are equally robust in the sense that their robustness proofs rely on the same assumption. We have found that the robust rate-limited controller experiences less chatter for constant references and more for time-varying ones. It also tends to use less torque, but in the time-varying case, it had a higher peak torque. We acknowledge that the differences



(a) Steering angle.



(b) Steering angle error.



(c) Steering torque.

Figure 5.7: Front-left wheel ramp plots with sliding mode controller.

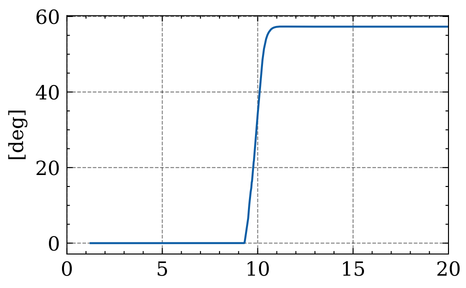
Table 5.3: Front-left wheel ramp performance metrics with sliding mode controller.

Metrics	
Mean absolute error [deg]	2.520
Chatter [Nm]	428.484
Peak torque [kNm]	3.782
Mean torque [kNm]	0.904

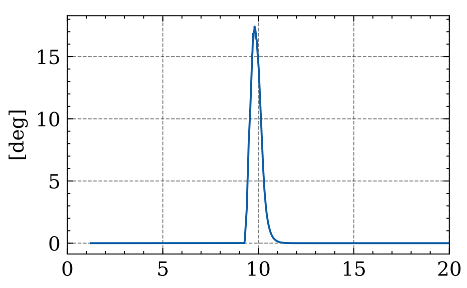
in the controllers may be a matter of parameter tuning. But since we see no significant difference between the controllers, we decided to go forward with SMC. The primary reason for this choice is that SMC is backed by the robust control litterature, which gives it credence.

Angular velocity controller

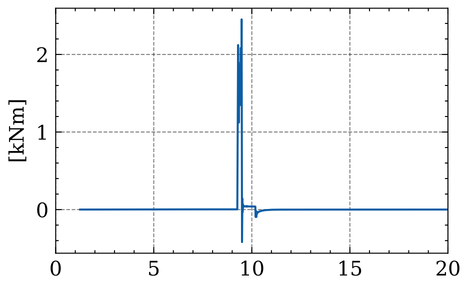
To evaluate the angular velocity controller, we apply a step input from 0 to 8.25 rad s^{-1}, which corresponds to the vehicle moving at 15 km h^{-1}. The reference signal and reference model are shown in Fig. 5.13. To avoid wheel misalignment forces, the SMC was used to keep all wheels pointing forward. Applying the step response, we get the results in Fig. 5.15. Note that we only show results for the front left wheel because we found no difference in the wheels' responses. In



(a) Steering angle.



(b) Steering angle error.



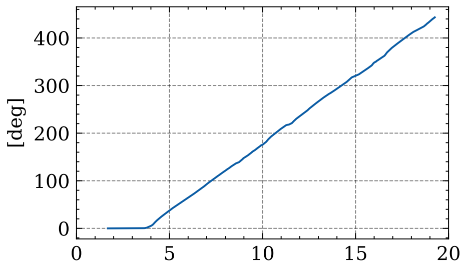
(c) Steering torque.

Figure 5.9: Front-left wheel step plots with robust rate-limited controller.

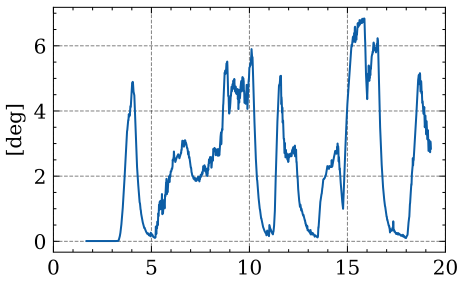
Table 5.4: Front-left wheel step metrics with robust rate-limited controller.

Metrics	
Mean absolute error [deg]	0.636
Stationary deviation [deg]	0.000
Chatter [Nm]	23.320
Peak torque [kNm]	2.452
Mean torque [kNm]	0.021

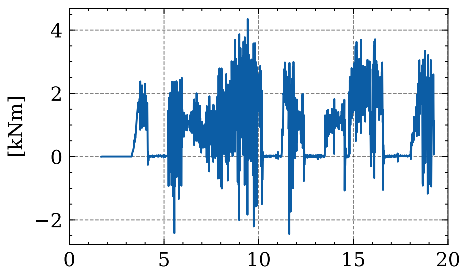
Fig. 5.14a, we can see that the angular velocity tracks the reference with some lag, and the error peaks at 158.614 deg/s. After about 7 s, the error converges to zero. It is probably possible to increase the gains so that the error converges faster, but since the simulator does not implement the physical limitations of the vehicle, we decided against it, as it may lead to irreproducible results in the field. Without doing system identification, it is hard to say whether the results given above are reasonable, but intuitively it seems that even a midsized agricultural vehicle should be able to go from 0 km h^{-1} to 15 km h^{-1} in 7 s.



(a) Steering angle.



(b) Steering angle error.

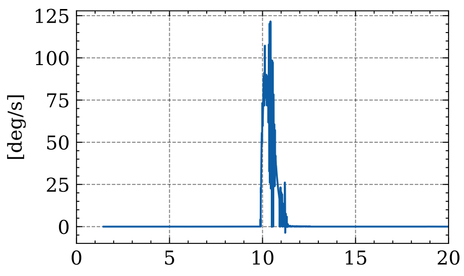


(c) Steering torque.

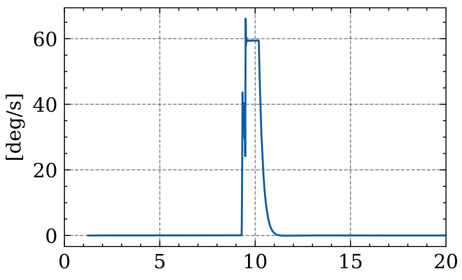
Figure 5.11: Front-left wheel ramp plots with robust rate-limited controller.

Table 5.5: Front-left wheel ramp metrics with robust rate-limited controller.

Metrics	
Mean absolute error [deg]	2.331
Chatter [Nm]	476.903
Peak torque [kNm]	4.349
Mean torque [kNm]	0.807



(a) Sliding-mode control.



(b) Robust rate-limited control.

Figure 5.12: Steering angle rates.

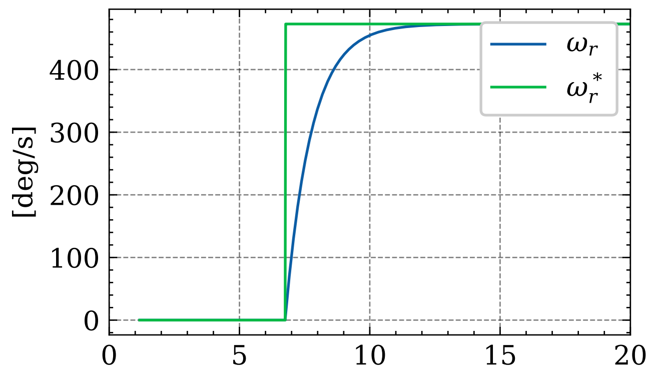
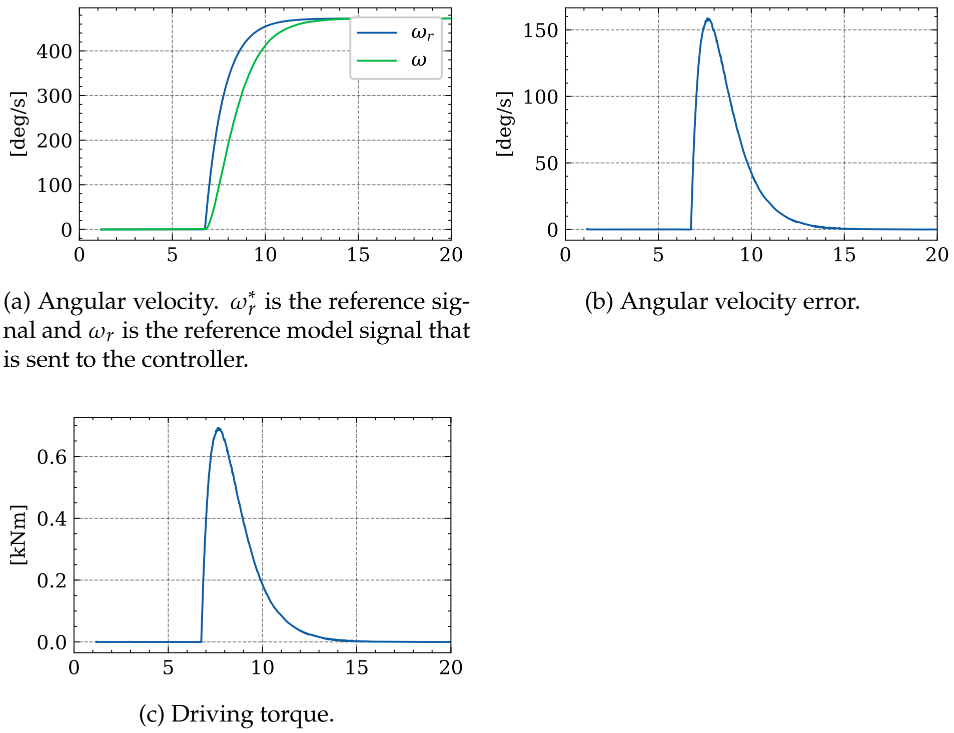


Figure 5.13: Angular velocity step function reference signal. ω_r^* is the reference signal and ω_r is the reference model signal that is sent to the controller.



(a) Angular velocity. ω_r^* is the reference signal and ω_r is the reference model signal that is sent to the controller.

(b) Angular velocity error.

(c) Driving torque.

Figure 5.15: Angular velocity tracking plots of step response from 0 to 8.25 rad s^{-1}.

Table 5.6: Angular velocity tracking metrics of step response from 0 to 8.25 rad s^{-1}.

Metrics	
Stationary deviation [deg/s]	0.028
Peak torque [kNm]	0.692

5.3 Vehicle control

The vehicle controller is the top layer in the control system. It gets a high-level description of how the vehicle should move and orient, and translates it into steering angles and angular velocities, as illustrated in Fig. 5.16. Movement is described with the desired course χ_d and desired speed v_r. Orientation is described with the desired yaw ψ_r or desired yawrate $\dot{\psi}_r$.

5.3.1 No-slip yawrate-course control

A big advantage of 4WIS vehicles is their ability to control course and yaw independently. This is what allows them to drive sideways and rotate on the spot. To take advantage of this, we develop a control law which can control both yawrate and course simultaneously. The controller presented here is fundamental to the system, as it bridges the gap between the vehicle states and the wheel states. The primary inputs to the controller are shown in Fig. 5.16, and they are desired course χ_d, commanded yawrate $\dot{\psi}_c$ and commanded speed v_c. In addition to this the controller also has access to the vehicle states like position, orientation, and their derivatives.

The relationship between sideslip β, course χ and yaw ψ is defined by [13] as

$$\chi = \beta + \psi \tag{5.40}$$

Using this, we express the commanded sideslip as a function of the desired course and actual yaw.

$$\beta_c = \chi_d - \psi \tag{5.41}$$

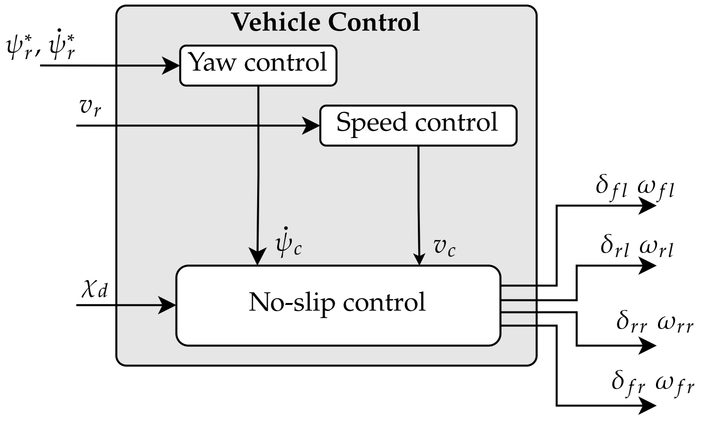

Vehicle Control
$\psi_r^*, \dot{\psi}_r^*$
Yaw control
v_r
Speed control
$\dot{\psi}_c$
v_c
χ_d
No-slip control
δ_{fl} ω_{fl}
δ_{rl} ω_{rl}
δ_{rr} ω_{rr}
δ_{fr} ω_{fr}

Figure 5.16: Vehicle control signals.

The vehicle now has a commanded sideslip and speed, meaning the commanded velocity relative to the body is given by

$$\mathbf{v}_c = v_c \begin{bmatrix} \cos\beta_c \\ \sin\beta_c \\ 0 \end{bmatrix} \tag{5.42}$$

Denote the position of wheel ij relative to the body as $\mathbf{p}_{ij}$. The commanded velocity of wheel ij relative to the ground is a sum of the vehicle velocity and rotational velocity, given as

$$\mathbf{v}_{ij,c} = \mathbf{v}_c + \dot{\psi}_c \hat{\mathbf{z}} \times \mathbf{p}_{ij} \tag{5.43}$$

Under no-slip conditions as given in Section 2.6, the wheel states $(\delta_{ij}, \omega_{ij})$ are fully[2] determined by the wheel velocity $\mathbf{v}_{ij,c}$. This is reasonable because the body velocity $\mathbf{v}$ and $\dot{\psi}$ uniquely determine the ICR of the vehicle, as shown in Section 2.5. Each wheel's y-axis must point toward the ICR, meaning that the wheel direction is orthogonal to the line that points toward the ICR.

No-slip conditions imply that the wheel sideslip must be equal to the steering angle. Eq. (2.24) then gives that the steering angle references as

$$\delta^*_{ij,r} = \text{atan2}(v_{ij,y}, v_{ij,x}). \tag{5.44}$$

No-slip conditions also require that the angular velocity of wheel is equal to the ground speed of the wheel divided by the radius, that is

$$\omega^*_{ij,r} = \frac{\|\mathbf{v}_{ij,c}\|}{r}. \tag{5.45}$$

Note that the wheels angular velocity reference $\omega^*_{ij,r}$ will always be positive with this setup. The reference optimization described in Section 5.2.4 that is implemented in the wheel control layer, will find an equivalent reference and potentially reverse the wheels.

In Fig. 5.17, we illustrate how the no-slip controller aligns the steering angle with the ICR.

Feasability of the wheel references

Using a high level description of the vehicle as presented above to translate between vehicle state references to wheel state references can be problematic. This is because many of the details of the vehicle are abstracted away. For instance the no-slip model has no notion of time delays and wheel actuators. It is reasonable to question whether the wheel references $\omega^*_{ij,r}$ and $\delta^*_{ij,d}$ that are found by using this model can even be executed on real hardware. To check whether it is possible

[2]Except if the speed of the wheel is zero, i.e. $\|\mathbf{v}_{ij,c}\| = 0$. Then the vehicle rotates about wheel ij and δ_{ij} can take on any value.

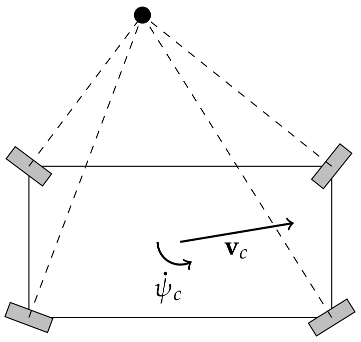


Figure 5.17: Wheel angles computed with $\dot{\psi} = 1$, $v_x = 3$, and $v_y = 0.5$. The ICR is illustrated as a black dot. Note that all wheels point orthogonally to the ICR.

we compute the derivatives of the wheel references. We argue that if there is a natural way to constrain $\dot{\omega}^*_{ij,r}$ and $\dot{\delta}^*_{ij,r}$ by constraining $\dot{\chi}_c$,$\ddot{\psi}_c$ or $\dot{\mathbf{v}}_c$, then the wheel references computed using this controller can always be made feasible by using slowly varying inputs.

For clarity we will drop the subscript ij in the derivations below, but the derivations apply to all wheels. Denote the computed velocity of the wheel by $\mathbf{v}_{ij,c} = \mathbf{v}_w$ with components $v_{w,x}$ and $v_{w,y}$, and norm $\|\mathbf{v}_w\| = v_w$. The derivative of δ_r is

$$\begin{aligned}\dot{\delta}^*_r &= \frac{\mathrm{d}\,\mathrm{atan2}(v_{w,y}, v_{w,x})}{\mathrm{d}t} \\ &= \frac{1}{v_w^2}\left(-v_{w,y}\frac{\mathrm{d}v_{w,x}}{\mathrm{d}t} + v_{w,x}\frac{\mathrm{d}v_{w,y}}{\mathrm{d}t}\right) \\ &= \frac{(\mathbf{v}_w \times \dot{\mathbf{v}}_w)_z}{v_w^2} \end{aligned} \tag{5.46}$$

From (5.46) we see that it is unfortunately not possible to put a finite upper bound on $\dot{\delta}_r$ in terms of $|\dot{\mathbf{v}}_w|$ or other vehicle states. This makes physical sense, because as the norm of $\mathbf{v}_w$ goes to zero, the wheel has no speed relative to the ground. The no-slip model defines the steering angle by the speed relative to the ground, so in this situation the steering angle is actually undefined.

This may be important to account for, because when the wheel is stationary, the no-slip model will compute $\omega^*_r = 0$, but the computed value of δ^*_r could be anything between $-\pi$ and π. If the wheel controller does not account for this, then it will try to control the steering angle to the undefined arbitrary value computed by (5.44), when the best option would probably be to stand still. While it is important, we do not account for it in the system as it stands now. We believe the reference optimization module in the wheel controller can be extended to deal with this by adding the current wheel state to the optimization problem if the commanded velocity is low enough, but this is left for future work.

Fortunately, the angular velocity reference ω_r^* is better behaved. The position of wheel ij relative to the body is $\mathbf{p}_w$ because we remove the subscripts for clarity. The velocity of the body is denoted $\mathbf{v}$. The derivative of ω_r^* is given as

$$\begin{aligned}\dot{\omega}_r^* &= \frac{1}{r}\frac{\mathrm{d}\left\|\mathbf{v}_c + \dot{\psi}_c\hat{\mathbf{z}} \times \mathbf{p}_w\right\|}{\mathrm{d}t} \\ &= \frac{1}{r}\left(\frac{\partial\left\|\mathbf{v}_c + \dot{\psi}_c\hat{\mathbf{z}} \times \mathbf{p}_w\right\|}{\partial \mathbf{v}_c}\dot{\mathbf{v}}_c + \frac{\partial\left\|\mathbf{v}_c + \dot{\psi}_c\hat{\mathbf{z}} \times \mathbf{p}_w\right\|}{\partial \dot{\psi}_c}\ddot{\psi}_c\right) \\ &= \frac{1}{r}\left(\frac{\mathbf{v}_c^T}{\|\mathbf{v}_c\|}\dot{\mathbf{v}}_c + \|\hat{\mathbf{z}} \times \mathbf{p}_w\|\ddot{\psi}_c\right) \end{aligned} \tag{5.47}$$

Note that $\dot{\omega}_r^*$ is linear in $\dot{\mathbf{v}}_c$ and $\ddot{\psi}_c$, which means it is straightforward to constrain. As a rough upper bound, we can write

$$|\dot{\omega}_r^*| \leq \frac{1}{r}|\dot{\mathbf{v}}_c| + \|\mathbf{p}_w\|\left|\ddot{\psi}_c\right| \tag{5.48}$$

Based on the above analysis, we argue the control systems should implement some logic to prevent problems where δ_r^* is undefined. This can easily be checked because δ_r^* is undefined when ω_r^* is zero. Controlling the angular velocity using the references given by this control should not pose any problems, because if the controller inputs are smoothly varying, then the angular velocity output will also be smoothly varying. By implementing a reference model for the vehicle controller inputs we can ensure this is the case, but this is left for future work.

5.3.2 Speed control

The speed controller gets a reference speed v_r from the guidance system and computes a commanded speed v_c which is sent to the no-slip controller. A simple approach for this is to use pure feedforward so that $v_c = v_r$, but we found that this leads to some steady state error. To mitigate this, we extend the feedforward controller with a proportional-integral (PI) controller, and write

$$e_v = v_r - v \tag{5.49}$$

$$v_c = v_r + K_p e_v + K_i \int e_v \, \mathrm{d}t \tag{5.50}$$

5.3.3 Yaw control

When it comes to controlling yaw, there are several modes of operation the vehicle controller needs to support. In a typical farming application, we may want the yaw to follow the course of the path, since then the tool mounted on the vehicle is aligned with the path. Likewise we may want the yaw to follow the course of the vehicle χ_d, which resembles driving like a typical car. But with 4WIS vehicles we can control the yaw indepedently of the course. In manual operation, we believe it is more intuitive to control the yawrate than the yaw. All this is to say that we need both yaw control and yawrate control.

Section 5.3.1 presents a control system for controlling the yawrate $\dot{\psi}$, but it provides no guarantee that the yawrate converges on the commanded yawrate $\dot{\psi}_c$. This means we need to include an integral effect if we want the controlled state to converge on the reference. We believe that accurate yawrate control is not required, because we are not aware of any farming or driving operations that require tight control of yawrate. It is however important to control the yaw, since it determines the direction the tool is pointed in. Based on these considerations we decided to only employ integral action on the yaw controller.

Yawrate controller The yawrate controller controls the yawrate $\dot{\psi}$ to the desired yawrate $\dot{\psi}_r$, which either comes as an external signal or is decided by the yaw controller below. To control the yawrate, it sets the commanded yawrate $\dot{\psi}_c$ using feedforward and a proportional effect given by

$$e_{\dot{\psi}} = \dot{\psi}_r - \dot{\psi} \tag{5.51}$$

$$\dot{\psi}_c = \dot{\psi}_r + K_p e_{\dot{\psi}} \tag{5.52}$$

Yaw reference model The yaw reference signal ψ_r^* is filtered through a reference model. This is done to ensure smooth control signals, which allows us to increase the gains and improve tracking performance [13]. The reference model we use is the second order linear model given by

$$\frac{\psi_r}{\psi_r^*}(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \tag{5.53}$$

where ω_0 is the natural frequency and ζ is the damping ratio. Since ψ_r represents an angle, we need to make a slight modification to this reference model. Instead of giving it the external signal ψ_r^* directly, we compute the equivalent angle of ψ_r^* which is closest to the reference model. This is done by

$$\psi_r^* := \psi_r + \mathrm{ssa}(\psi_r^* - \psi_r) \tag{5.54}$$

Note that ψ_r^* is redefined by this transformation.

Yaw controller To control the yaw, we can use the yawrate controller given above. Then we need to design a yawrate reference $\dot{\psi}_r$. A natural way to do this is to use

Table 5.7: Vehicle controller parameters.

(a) Speed controller.

K_p	1
K_i	0.1

(b) Yawrate controller.

K_p	1.0

(c) Yaw controller.

K_p	4
K_i	2
K_d	4
$\dot{\psi}_{\max}$	60 deg/s

(d) Yaw reference model.

ω_0	1
ζ_0	0.9

a saturated PID controller.

$$e_\psi = \psi_r - \psi \tag{5.55}$$

$$\dot{\psi}_r = \text{sat}\left(K_p e_\psi + K_i \int e_\psi \, \mathrm{d}t + K_d \dot{e}_\psi, \dot{\psi}_{\max}\right) \tag{5.56}$$

The reason to use a saturated controller is to ensure that the desired yawrates generated by the controller are reasonable. Saturating the controller also allows us to increase the gains, which improves tracking performance.

5.3.4 Parameters

The parameters for the vehicle control system are given in Table 5.7.

5.3.5 Results

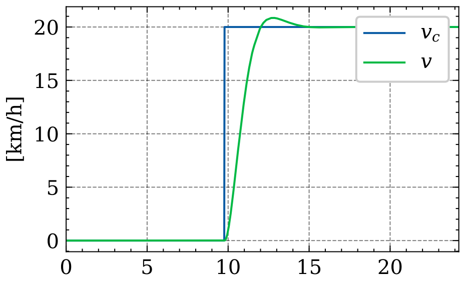
Speed controller

The speed controller developed in Section 5.3.2 is tested on a step response from 0 to 20 km h^{-1}, and the results are given in Fig. 5.19 and Table 5.8. In Fig. 5.18a, we can see there is a slight overshoot before the speed converges on the desired speed. Using this controller, the vehicle accelerates to 20 km h^{-1} in about 6 s, which could be unrealistic for such a large vehicle. As with the wheel controllers, we need to do system identification before we can make that judgement. An approach to make the response more realistic is to employ a reference model on the desired speed. We did not implement a reference model, because the desired speed is constant in our implementation[3], but a more complete system would need to account for it.

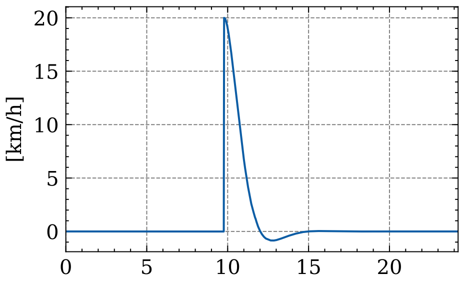
Yawrate controller

We test the yawrate controller on a step response from 0 to 15 deg/s. The results are given in Fig. 5.20. Looking at the yawrate error in Fig. 5.20a, we see that it has a slight overshoot, but it converges within about 3 s. Note however that there is a stationary deviation of 0.575 deg/s, which stems from the fact that the no-slip model is a simplified model and the controller does not include an integral effect.

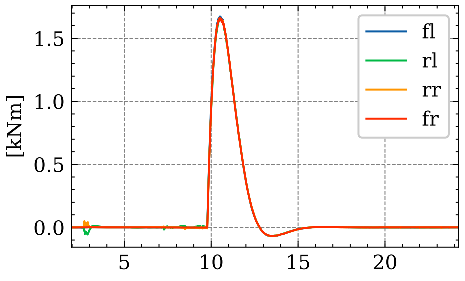
[3]See details in Section 6.3.



(a) Speed.



(b) Speed error.



(c) Drive torques on all the wheels.

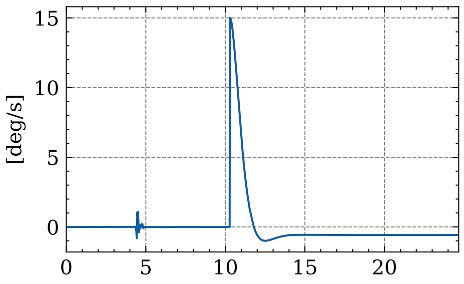
Figure 5.19: Speed controller plots for step response from 0 to 20 km h^{-1}.

Table 5.8: Speed controller performance metrics for step response from 0 to 20 km h^{-1}.

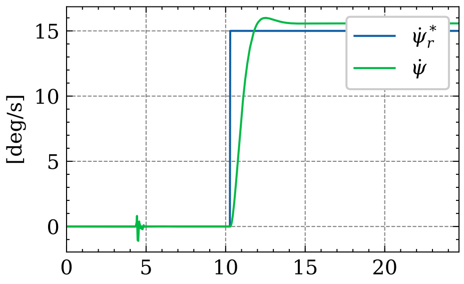
Metrics	
Stationary deviation [km/h]	0.002
Peak torque [kNm]	1.673
Chatter [Nm]	0.841

Yaw controller

The yaw controller is tested with a step response from 0 to 60 deg, and the results are shown in Fig. 5.22 and Table 5.9. The controller is able to follow the reference model with a peak error of 8.959 deg, but it does need 8 s before it converges on the reference value. We believe this could be improved with controller tuning, but from a practical standpoint, we think the performance is sufficient is it stands now.

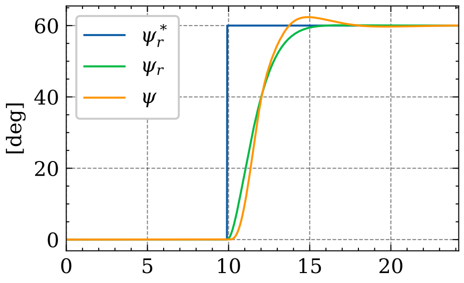


(a) Yawrate error.

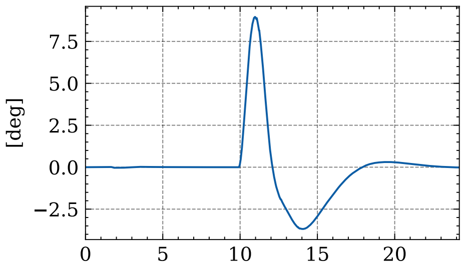


(b) Yawrate.

Figure 5.20: Step response applied to yawrate controller.



(a) Yaw with step input and reference model. ψ_r^* is the reference input that was sent to the controller. ψ_r is the reference model signal. ψ is the yaw of the vehicle.



(b) Yaw error.

Figure 5.22: Step response plots for yaw controller.

Table 5.9: Step response performance metrics for yaw controller.

Metrics	
Mean absolute error [deg]	1.335
Stationary deviation [deg]	0.294
Peak yawrate [deg/s]	32.293

5.4 Limitations

The hierarchical control system presented above provides a good basis for controlling the vehicle. The results above indicate that it works well in theory and in simulated case studies, but there could be problems if we port it to the real vehicle. A big potential problem is the chattering caused by both SMC and robust rate controllers. When the wheels get a constant steering angle reference, the measured chatter was the lowest. This can be seen in Figs. 5.4c and 5.8c, where the measured chatters were 184.449 N m and 23.320 N m, respectively. In the time varying cases, both controllers exhibit significant chatter, and the steering torque jumps significantly from timestep to timestep, as can be seen in Figs. 5.6c and 5.10c. This has not been a problem in simulation because we do not simulate motor dynamics so the commanded torque is applied immediately. We suspect that if we add motor dynamics to the simulation, then the results would be significantly worse and perhaps unstable. To reduce the chatter, we can increase softregion parameter ϵ, which will increase the steady-state error of the controller. The steady-state error of the SMC in the step response case was 0.021 deg, which is very low. So we believe there is room to increase ϵ without a noticable performance decrease.

Another limitation of the system is that it outputs torques. This is convenient because the simulator accepts torques as inputs, but a complete system needs to output motor signals. This could be done via a motor driver or a hardware abstraction layer beneath the wheel control layer, since we suspect that solving this could be hardware specific.

A related problem to the ones described above is that we do not know how well the controllers and results described here translate to the real vehicle. This is because we have not done system identification on the vehicle. By doing system identification and creating a detailed data-driven model for the vehicle, we would be able to create a more accurate simulation, which could give us more confidence in the results.

6 Guidance System

The guidance system is the system that tells the vehicle where to move. It gets a list of ordered waypoints, constructs a continuous path betweeen them, and tells the vehicle control layer how to follow the path. The signal flow is shown in Fig. 6.1. The waypoints are used by a path-smoothing algorithm to create an internal representation of the smoothened path. This is then used by a path-following algorithm to generate course and speed setpoints.

A path is a continuous curve through space which the vehicle should follow to get from a starting position to an end position. A commonly used (e.g. [13, 22]) mathematical definition for a path is that it is

Definition 6.1 (Path)**.** A path is geometric curve $\mathbf{P}_d(t_p) \in \mathbb{R}^q$ where $t_p \in [0, 1]$ [13].

The variable t_p is the path-parametrization variable, which is without a physical interpretation. By geometric curve, we mean that we are only concerned with the shape of the curve over the entire domain $t_p \in [0, 1]$. Infinitely parametrizations can exist for the same geometric curve[1], but they are all considered equivalent.

Designing a continuous function $\mathbf{P}_d(t_p)$ directly is difficult so a common approach is to specify a list of waypoints, and then generate a path from those waypoints. There are many technical details and design decisions surrounding this, and Lekkas [22] provides an extensive resource on the topic.

In this chapter we use the theory covered by Lekkas to build a path-smoothing system suitable for path-following. Additionally we will cover some practical aspects that are relevant for software implementation and the end user. We then use the path-smoothing system to implement path-following.

[1] $\{\mathbf{P}_d(t_p^a) \quad \text{for all} \quad a \in \mathbb{R}\}$ are all the same geometric curve.

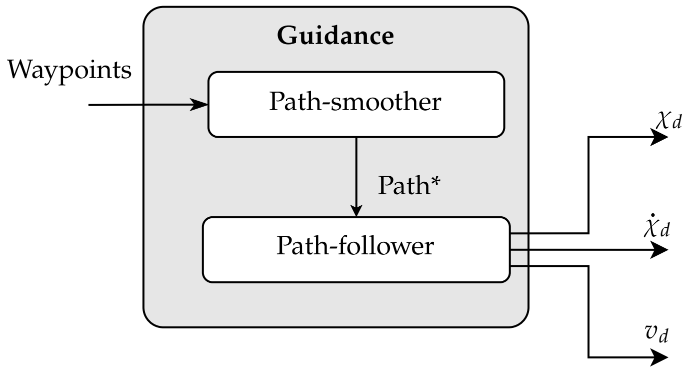


Figure 6.1: Vehicle guidance signals.

6.1 Waypoint specification

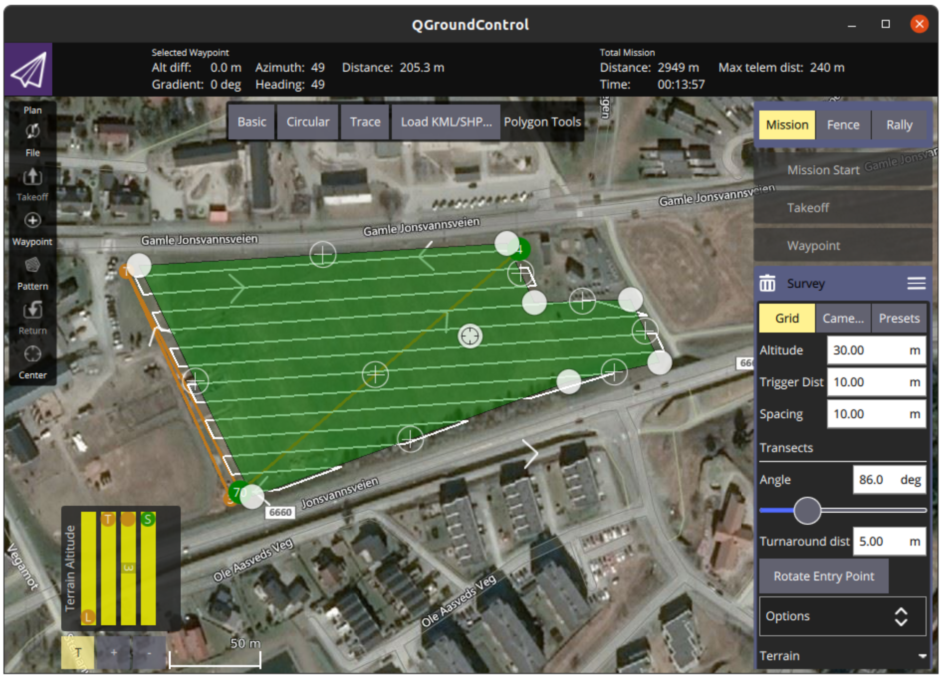
Waypoints are used in many industries as input to the guidance system. From an end-user perspective, waypoints provide an intuitive and fast way to give a path to a vehicle. The exact definition of a waypoint depends on the application, but an example is a longitude-latitude pair (ϕ_i, λ_i). Over short distances, waypoint positions can instead be described with Cartesian coordinates. The pair could be extended with properties like altitude, heading, and speed depending on the application. To keep the description simple, we have for this project used Cartesian coordinates to describe the waypoints.

Definition 6.2 (Waypoint)**.** A waypoint is a pair of Cartesian coordinates in the inertial frame.

$$\text{waypoint}_i = \mathbf{p}_i = \begin{bmatrix} x_i & y_i \end{bmatrix} \tag{6.1}$$

We have not done any work relating to automatic waypoint generation. Many path-planning algorithms naturally fit this description, but from an operational perspective of the AutoAgri vehicle, we believe more value is gained from easily specifying waypoints, rather than automatic path planning. So we decided it was more important to explore intuitive ways of adding waypoints. Figure 6.2 shows an example of QGroundControl (QGC) being used to build a lawnmover pattern across a small field close to Dragvoll, Trondheim. More details are presented in Chapter 7. The guidance system is agnostic as to how the waypoints are generated, so in this chapter we assume we are given a list of waypoints denoted

$$\text{Waypoints} = \{\mathbf{p}_1, \mathbf{p}_1, \dots, \mathbf{p}_N\} \tag{6.2}$$


QGroundControl

Selected Waypoint
Alt diff: 0.0 m Azimuth: 49 Distance: 205.3 m
Gradient: 0 deg Heading: 49

Total Mission
Distance: 2949 m Max telem dist: 240 m
Time: 00:13:57

Survey
Altitude 30.00 m
Trigger Dist 10.00 m
Spacing 10.00 m
Transects
Angle 86.0 deg
Turnaround dist 5.00 m

Figure 6.2: QGC lawnmover pattern setup using surveys.

6.2 Path-smoothing

An operator or a path-planning system creates an ordered list of waypoints that will be used to create a path. From this, we can choose to build either an interpolating or an approximating path. The difference is that interpolating paths are required to pass through all the waypoints, whereas approximating paths are not. In this project we decided to use approximating paths as they are easier to work with with the parametrization that will be discussed in the subsequent sections.

6.2.1 Path segment types

The path-smoothing system needs a palette of path types that it can connect together to build a large path. Depending on the segment type, these segments are used to connect two or three waypoints, and there are several to choose from. In this work we are only concerned with straight, circular, and Fermat spiral segments. We mention that Lekkas also discusses clothoids and spline interpolating path-types like Beziér curves. Clothoids have many of the same properties as Fermat spirals, but they do not have a closed form expression, which makes them computationally demanding. Lekkas argues against the use of spline interpolating paths because it is hard to show that they have continuous curvature. In the following sections, we present the path segment types that are used in this work.

Linear path segments

The simplest way to generate a path from a two waypoints is to connect them with a straight line. Given waypoint positions $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$, the straight path between them is

$$\mathbf{P}(t_p) = \mathbf{p}_i(1 - t_p) + t_p\mathbf{p}_{i+1} = \mathbf{p}_i + t_p(\mathbf{p}_{i+1} - \mathbf{p}_i) \tag{6.3}$$

Circular path segments

A circular arc path is defined by circle center $\mathbf{p}_c$, circle radius R, starting angle α_0, and ending angle α_1. It is then given by

$$\mathbf{P}(t_p) = \mathbf{p}_c + \begin{bmatrix} R\cos\big(\alpha_0 + t_p(\alpha_1 - \alpha_0)\big) \\ R\sin\big(\alpha_0 + t_p(\alpha_1 - \alpha_0)\big) \end{bmatrix} \tag{6.4}$$

Fermat spiral path segments

The Fermat spiral is a curve described by the polar coordinate expression

$$r = k\sqrt{\theta} \tag{6.5}$$

By shifting this expresion to be centered on $\mathbf{p}_s$, rotating it with angle χ_s, and expressing it in Cartesian coordinates we get the parametrization used by Lekkas.

$$\mathbf{P}(\theta) = \mathbf{p}_s + \begin{bmatrix} k\sqrt{\theta}\cos\big(\rho\theta + \chi_s\big) \\ k\sqrt{\theta}\sin\big(\rho\theta + \chi_s\big) \end{bmatrix} \tag{6.6}$$

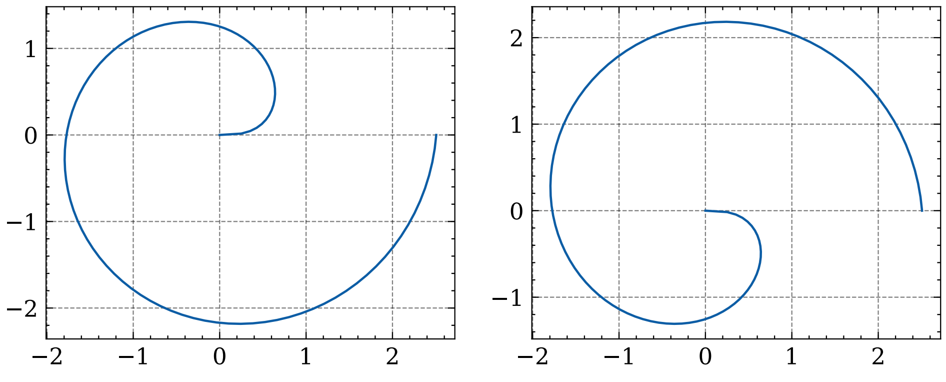
Note that $\rho = \pm 1$ decides the turning direction. The spiral is parametrized in terms of the angle θ. In Fig. 6.3, we illustrate full rotations of Fermat spirals in both directions. To aid in software implementation, we found it helpful to modify this paremetrization slightly. In [22], they assume $\theta \in [0, \theta_{\text{end}}]$ and use ρ to change the direction. We remove ρ as a parameter and allow θ to take on negative values. We then write θ in terms of the path variable t_p as

$$\theta = \theta_{\text{begin}} + t_p(\theta_{\text{end}} - \theta_{\text{begin}}) \tag{6.7}$$

where $t_p \in [0, 1]$. By making $\theta_{\text{begin}} > \theta_{\text{end}}$ we can change the turn direction without using the turn direction parameter ρ. The Fermat spiral is now parametrized as

$$\mathbf{P}(\theta) = \mathbf{p}_s + \begin{bmatrix} k\text{sgn}\,(\theta)\,\sqrt{|\theta|}\cos(\theta + \chi_s) \\ k\text{sgn}\,(\theta)\,\sqrt{|\theta|}\sin(\theta + \chi_s) \end{bmatrix} \tag{6.8}$$

The curvature of the Fermat spiral is given by [35] as



(a) Counter-clockwise turn ($\rho = 1$).

(b) Clockwise turn ($\rho = -1$).

Figure 6.3: Examples of Fermat spirals with $k = 1$ and θ ranging from 0 to 2π.

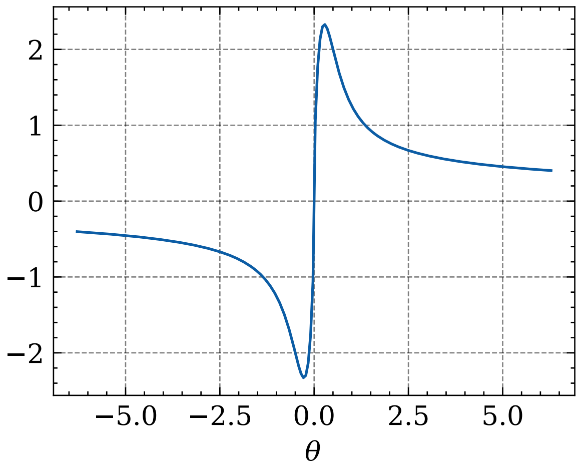


Figure 6.4: Curvature of Fermat spiral with $k = 1$.

$$\kappa(\theta) = \frac{1}{k}\frac{2\sqrt{\theta}(3+4\theta^2)}{(1+4\theta^2)^{3/2}} \quad \text{for} \quad \theta \geq 0 \tag{6.9}$$

Extending the curvature formula to allow $\theta < 0$, we write

$$\kappa(\theta) = \frac{1}{k}\frac{2\operatorname{sgn}(\theta)\sqrt{|\theta|}(3+4\theta^2)}{(1+4\theta^2)^{3/2}} \tag{6.10}$$

The curvature is illustrated in Fig. 6.4.

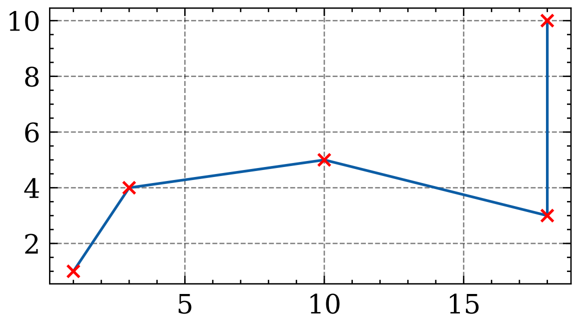


Figure 6.5: Path constructed with straight segments.

Combined path segments

If we are given n paths $\mathbf{P}_i(t_p)$ for $i = 1, 2, \ldots, n$ where $t_p \in [0, 1]$ for all the paths individually. Then the path described by following all of the subpaths is given as

$$\mathbf{P}(t_p) = \begin{cases} \mathbf{P}_1(nt_p) & \text{for} \quad t_p \in [0, 1/n) \\ \cdots & \\ \mathbf{P}_i(nt_p) & \text{for} \quad t_p \in [(i-1)/n, i/n) \\ \cdots & \\ \mathbf{P}_n(nt_p) & \text{for} \quad t_p \in [(n-1)/n, 1] \end{cases} \tag{6.11}$$

6.2.2 Path smoothing

Now that we have a collection a path segment types, we turn our attention to the path-smoothing problem. We want to use the paths described above to create a connected path between the waypoints.

The most straightforward way to do this is to use straight path segments between the waypoints, as illustrated in Fig. 6.5. According to [22], such paths are infeasible for underactuated vehicles. And even though they are feasible for fully actuated vehicles, they require the vehicle to stop and turn at each waypoint. This problem can be seen by looking at the curvature of the path, which is zero everywhere except at the waypoints where it is infinite. Due to this problem, we will explore two more sophisticated methods of path smoothing; circular smoothing and spiral smoothing.

Circular path-smoothing improves upon the straight line approach, by adding circle segments at each turn. By doing this, we create a variant of Dubin's path[2] which is shown to be the fastest path between two poses for particles moving with constant speed and a maximum curvature constraint [11]. Figure 6.6 shows

[2]Strictly speaking, Dubin's path applies to interpolating curves, but we are designing an approximating curve.

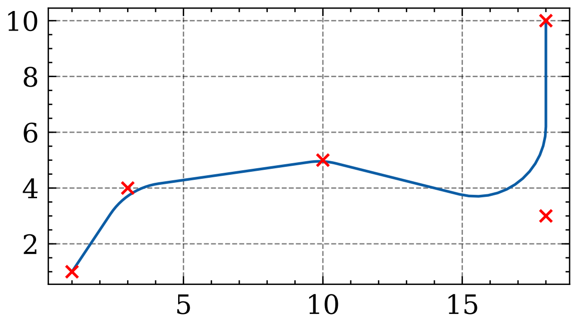


Figure 6.6: Path constructed with straight and circular segments.

a circularly smoothed path. The algorithm for constructing this path is given in Appendix C. To the human eye, a circularly smoothed path may look good, but as [22] argues, it also has a curvature discontinuity. The straight line segments have curvature 0 and the circle segments have curvature $1/R$. This is problematic because to maintain track during the transitions between the line and circle segments, the wheel need to follow a step response, which is infeasible in practice.

Fermat spiral path-smoothing

To solve the curvature discontinuity problem, [22] proposes that we use Fermat spirals in the turns. This is reasonable because as Fig. 6.4 shows, the curvature of a Fermat spiral changes continously and is 0 at $\theta = 0$ so we can use them to transition out of straight line segments into turns. We will now restate the path-smoothing algorithm presented by [22], and give the small modification we used since we allow $\theta < 0$.

Before starting we need to specify the maximum curvature parameter κ_{max}. Assume we are given three consecutive waypoints $\mathbf{p}_{i-1}$, $\mathbf{p}_i$, and $\mathbf{p}_{i+1}$ as illustrated in Fig. 6.7a. We will smooth the path by constructing two mirrored spirals: one exiting the straight path from $\mathbf{p}_{i-1}$ to $\mathbf{p}_i$ denoted $\mathbf{P}(\theta)$, and one entering the straight path from $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$ denoted $\bar{\mathbf{P}}(\theta)$. This is illustrated in Fig. 6.7b.

Compute the normalized velocities in and out of center point as

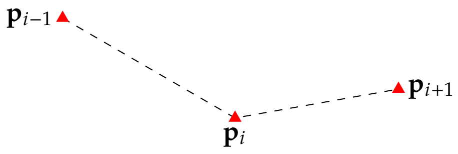
$$\mathbf{v}_{\text{in}} = \frac{\mathbf{p}_i - \mathbf{p}_{i-1}}{||\mathbf{p}_i - \mathbf{p}_{i-1}||} \tag{6.12}$$

$$\mathbf{v}_{\text{out}} = \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{||\mathbf{p}_{i+1} - \mathbf{p}_i||} \tag{6.13}$$

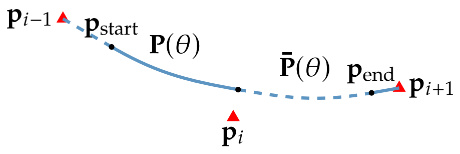
From these we can also compute the courses as

$$\chi_{\text{in}} = \text{atan2}(\mathbf{v}_{\text{in},y}, \mathbf{v}_{\text{in},x}) \tag{6.14}$$

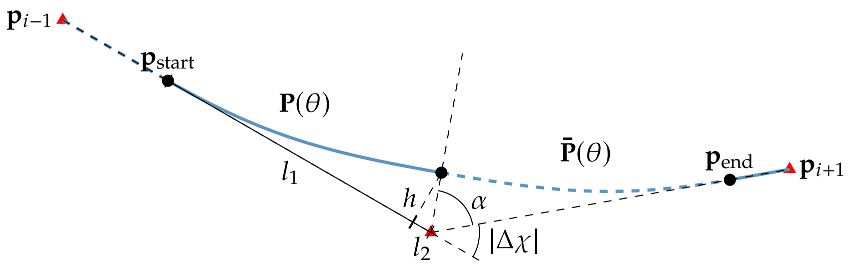
$$\chi_{\text{out}} = \text{atan2}(\mathbf{v}_{\text{out},y}, \mathbf{v}_{\text{out},x}) \tag{6.15}$$



(a) Three consecutive waypoints.


$\mathbf{p}_{i-1}$
$\mathbf{p}_{\text{start}}$
$\mathbf{P}(\theta)$
$\bar{\mathbf{P}}(\theta)$
$\mathbf{p}_{\text{end}}$
$\mathbf{p}_{i+1}$
$\mathbf{p}_i$

(b) Spiral constructions.



(c) Spiral positions.

Figure 6.7: Fermat smoothing illustrations.

The course change from the turn is computed as

$$\Delta\chi = \text{ssa}(\chi_{\text{out}} - \chi_{\text{in}}) \tag{6.16}$$

where ssa is the smallest signed angle function. The turn direction ρ is computed as

$$\rho = \text{sgn}\,(\Delta\chi) \tag{6.17}$$

The first spiral path $\mathbf{P}(\theta)$ goes from $\theta = 0$ to $\theta = \theta_{\text{mid}}$. To determine θ_{mid} we express the course change as a function of θ. Following [22], we use

$$\Delta\chi(\theta) = \theta + \text{atan}(2\theta) \tag{6.18}$$

Ideally we would invert this to find $\theta(\Delta\chi)$, but this is not possible. Instead we'll exploit the fact that $\Delta\chi(\theta)$ is continuous and differentiable to find θ_{mid} numerically. At the midpoint, the course must have changed by exactly half of the total course change. This means that θ_{mid} is defined by

$$\frac{|\Delta\chi|}{2} = \theta_{\text{mid}} + \text{atan}(2\theta_{\text{mid}}) \tag{6.19}$$

Note that we take the absolute value $\Delta\chi$. This is not required, but it ensures that $\theta_{\text{mid}} > 0$ which simplifies some the remaining steps. Lekkas, and reference therein [26], recommends using Halley's method to solve this. Halley's method is a root finding algorithm, meaning it solves $f(x) = 0$. In [36], it is given as

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2f'(x_n)^2 - f(x_n)f''(x_n)} \tag{6.20}$$

To solve (6.19), we use

$$f(\theta) = \theta + \operatorname{atan}(2\theta) - |\Delta\chi|/2 \tag{6.21}$$

$$f'(\theta) = 1 + \frac{2}{1+4\theta^2} \tag{6.22}$$

$$f''(\theta) = -\frac{16\theta}{(1+4\theta^2)^2} \tag{6.23}$$

and we initialize it at $\theta_0 = 0$. [26] found that θ_n converged with tolerance 10^{-3} after only one iteration[3], but in practice we will do multiple iterations of (6.20) until it converges.

Now that the domain is determined, we need to determine the scale parameter k. Looking at (6.9), we see that the k is used to scale the curvature, so it has to be set so that the spiral does not exceed the curvature constraint κ_{max}. Fermat spirals always have a maximum curvature at $\theta = \sqrt{\frac{\sqrt{7}}{2} - \frac{5}{4}}$, and the curvature is strictly increasing on $0 \leq \theta < \sqrt{\frac{\sqrt{7}}{2} - \frac{5}{4}}$. Thus the point of maximum curvature on the domain $0 \leq \theta \leq \theta_{\text{mid}}$ is

$$\theta_{\kappa,\text{max}} = \min\left(\theta_{\text{mid}}, \sqrt{\frac{\sqrt{7}}{2} - \frac{5}{4}}\right) \tag{6.24}$$

To maintain the curvature constraint, we use $\kappa(\theta_{\kappa,\text{max}}) = \kappa_{\text{max}}$ in (6.9) and solve for the scaling parameter. This gives

$$k = \frac{1}{\kappa_{\text{max}}} \frac{2\sqrt{\theta_{\kappa,\text{max}}}(3 + 4\theta_{\kappa,\text{max}}^2)}{(1 + 4\theta_{\kappa,\text{max}}^2)^{3/2}} \tag{6.25}$$

Now that we have determined the shape and orientation of the spiral, all that remains is determining the start position $\mathbf{p}_{\text{start}}$ and end position $\mathbf{p}_{\text{end}}$, which are where the positions where the spirals intersect with the straight line paths as shown in Fig. 6.7b. To determine the spiral positions, we use Fig. 6.7c. The length l_1 is the x-component of the Fermat spiral relative to the starting position and orientation. To determine l_2, we define an angle α which satisfies $2\alpha + |\Delta\chi| = \pi$. Along with the height h, this allows us to compute l_2. Putting it all togheter, we have

$$\alpha = \frac{\pi - |\Delta\chi|}{2} \tag{6.26}$$

$$h = k\sqrt{\theta_{\text{mid}}}\sin(\theta_{\text{mid}}) \tag{6.27}$$

$$l_1 = k\sqrt{\theta_{\text{mid}}}\cos(\theta_{\text{mid}}) \tag{6.28}$$

$$l_2 = h/\tan(\alpha) \tag{6.29}$$

$$l = l_1 + l_2 \tag{6.30}$$

[3]Meaning that $|f(\theta_1)| < 10^{-3}$

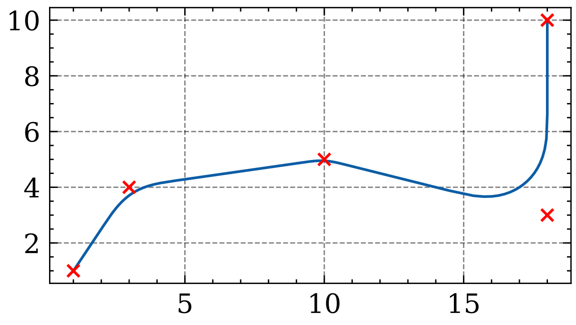


Figure 6.8: Fermat smoothing around waypoint.

By combining the above equations with the normalized velocities in and out of the center point, we compute the spiral positions as

$$\mathbf{p}_{\text{start}} = \mathbf{p}_i - l\mathbf{v}_{\text{in}} \tag{6.31}$$

$$\mathbf{p}_{\text{end}} = \mathbf{p}_i + l\mathbf{v}_{\text{out}} \tag{6.32}$$

Now that all the variables are determined, we can create the smoothed path between $\mathbf{p}_{i-1}$ and $\mathbf{p}_{i+1}$ via $\mathbf{p}_i$. First construct a straight path from $\mathbf{p}_{i-1}$ to $\mathbf{p}_{\text{start}}$. If the turn direction is positive ($\Delta\chi > 0$), then add the following spirals to the path

$$\mathbf{P}(\theta) = \mathbf{p}_{\text{start}} + \begin{bmatrix} k\,\text{sgn}(\theta)\sqrt{|\theta|}\cos(\theta + \chi_{\text{in}}) \\ k\,\text{sgn}(\theta)\sqrt{|\theta|}\sin(\theta + \chi_{\text{in}}) \end{bmatrix} \quad \text{with} \quad \theta = t_p\theta_{\text{mid}} \tag{6.33}$$

$$\bar{\mathbf{P}}(\theta) = \mathbf{p}_{\text{end}} + \begin{bmatrix} k\,\text{sgn}(\theta)\sqrt{|\theta|}\cos(\theta + \chi_{\text{out}}) \\ k\,\text{sgn}(\theta)\sqrt{|\theta|}\sin(\theta + \chi_{\text{out}}) \end{bmatrix} \quad \text{with} \quad \theta = (t_p - 1)\theta_{\text{mid}} \tag{6.34}$$

If the turn direction is negative, then add the following spirals instead

$$\mathbf{P}(\theta) = \mathbf{p}_{\text{start}} + \begin{bmatrix} k\,\text{sgn}(\theta)\sqrt{|\theta|}\cos(\theta + \chi_{\text{in}} + \pi) \\ k\,\text{sgn}(\theta)\sqrt{|\theta|}\sin(\theta + \chi_{\text{in}} + \pi) \end{bmatrix} \quad \text{with} \quad \theta = -t_p\theta_{\text{mid}} \tag{6.35}$$

$$\bar{\mathbf{P}}(\theta) = \mathbf{p}_{\text{end}} + \begin{bmatrix} k\,\text{sgn}(\theta)\sqrt{|\theta|}\cos(\theta + \chi_{\text{out}} + \pi) \\ k\,\text{sgn}(\theta)\sqrt{|\theta|}\sin(\theta + \chi_{\text{out}} + \pi) \end{bmatrix} \quad \text{with} \quad \theta = (1 - t_p)\theta_{\text{mid}} \tag{6.36}$$

Following the above paths, we get a continuous curvature path from $\mathbf{p}_{i-1}$ to $\mathbf{p}_{\text{end}}$. If $\mathbf{p}_{i+1}$ is the final waypoint, then we finish with a straight path from $\mathbf{p}_{\text{end}}$ to $\mathbf{p}_{i+1}$. Otherwise we repeat the process untill we reach the final waypoint. Doing this process for the path illustrated previously, we get the path shown in Fig. 6.8. The same path is illustrated in Fig. 6.9 with different curvature parameters. We can see that when the maximum allowed curvature is bigger, then the path will have sharper turns. The curvature will always be continuous, but the vehicle may need

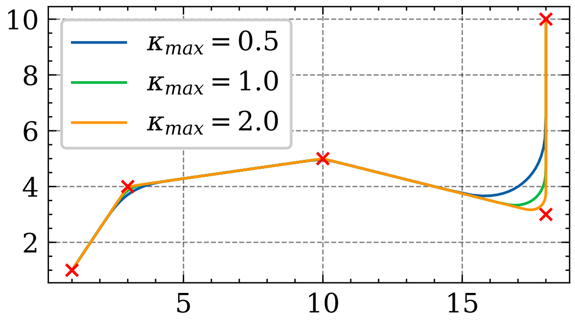


Figure 6.9: Fermat smoothing with different curvature parameters.

to slow down in order to make the turn if the curvature parameter is too high. Another thing to notice in Fig. 6.9, is that paths with lower curvature will deviate more from the straight line path, especially in sharp turns. This property is called allowance and is discussed more in [22], where they gave a closed form expression for the allowance of Fermat spirals.

$$\text{allowance} = h = k\sqrt{\theta_{\text{mid}}}\sin(\theta_{\text{mid}}) \tag{6.37}$$

6.2.3 Closest point determination

Since a smoothed path consists of several path segments, we need to determine the segment that is closest to our current position. We note that this is only applicable to path-following, and not trajectory-tracking. With trajectory-tracking, we specify how the path parameter t_p changes as a function of time, and then $\mathbf{P}(t_p)$ will be the point on the path that we should track. With path-following we are agnostic to the value of t_p, so we must consider the entire path. To find out which path segment the vehicle is closest to, we compute the minimum distance to each segment individually. The segment with the lowest minimum distance is the segment the vehicle is closest to.

Computing the distance from a point to a line segment or a circle segment is trivial, so we will not cover those cases. However, computing the distance from a point to a Fermat spiral is not straight forward, so we will cover it in depth.

Closest point on Fermat spiral A general Fermat spiral is given as

$$\mathbf{P}(\theta) = \mathbf{p}_s + k\,\text{sgn}\,(\theta)\sqrt{|\theta|}\begin{bmatrix}\cos(\theta+\chi_s)\\ \sin(\theta+\chi_s)\end{bmatrix} \tag{6.38}$$

where

$$\theta = \theta_{\text{begin}} + t_p(\theta_{\text{end}} - \theta_{\text{begin}}) \tag{6.39}$$

We want to determine the value of θ that is closest to a given position $\mathbf{p}$, denoted θ_{closest}. That is, we want to solve a constrained optimization problem

$$\theta_{\text{closest}} = \operatorname{argmin}(||\mathbf{P}(\theta) - \mathbf{p}||) \tag{6.40}$$

To initialize the optimization method, we sample the spiral at a small number[4] of θ values in the domain, and select the best value. After this, we solve it as an unconstrained optimization problem.

A problem with this parametrization is that its derivative has a singularity at the origin. To solve this, [22] and reference therein [27], propose that we use a coordinate transformation $u = \sqrt{\theta}$. This works in the setup used by [22], since they formulate the spiral so that $\theta \geq 0$. To simplify implementation we have allowed negative values of θ, which won't work in the suggested transformation. To remedy this we make a slight modification and instead use the transformation

$$u = \operatorname{sgn}(\theta)\sqrt{|\theta|} \tag{6.41}$$

With this transformation, we have $\theta = \operatorname{sgn}(u)\, u^2$, and the Fermat spiral becomes

$$\mathbf{P}(u) = \mathbf{p}_s + ku \begin{bmatrix} \cos\big(\operatorname{sgn}(u)\, u^2 + \chi_s\big) \\ \sin\big(\operatorname{sgn}(u)\, u^2 + \chi_s\big) \end{bmatrix} \tag{6.42}$$

Reformulating the optimization problem we have

$$u_{\text{closest}} = \operatorname{argmin}(||\mathbf{P}(u) - \mathbf{p}||) \tag{6.43}$$

This is equivalent to minimizing the distance squared, which we can write as a cost function

$$J(u) = (\mathbf{P}(u) - \mathbf{p})^T(\mathbf{P}(u) - \mathbf{p}) \tag{6.44}$$

The optimal value of u is given by the minimum of the cost function, which we can find by solving $\frac{\mathrm{d}J}{\mathrm{d}u} = 0$.

$$\frac{\mathrm{d}J}{\mathrm{d}u} = 2(\mathbf{P}(u) - \mathbf{p})^T \frac{\mathrm{d}\mathbf{P}}{\mathrm{d}u} \tag{6.45}$$

Using the new parametrization, we can compute the derivative as

$$\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}u} = k \begin{bmatrix} \cos\big(\operatorname{sgn}(u)\, u^2 + \chi_0\big) \\ \sin\big(\operatorname{sgn}(u)\, u^2 + \chi_0\big) \end{bmatrix} + 2k\operatorname{sgn}(u)\, u^2 \begin{bmatrix} -\sin\big(\operatorname{sgn}(u)\, u^2 + \chi_0\big) \\ \cos\big(\operatorname{sgn}(u)\, u^2 + \chi_0\big) \end{bmatrix} \tag{6.46}$$

Using gradient descent, we could find the minimum with the above equation. But by computing the second derivative aswell and employing Newton's method we can in theory improve the convergence rate on the optimal value. Another

[4]We used 10 samples, but the scheme doesn't seem sensitive to this.

advantage of Newton's method is that it does not require us to tune a stepsize parameter. The second derivative of the cost function is

$$\frac{\mathrm{d}^2 J}{\mathrm{d}u^2} = 2\left(\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}u}\right)^T \frac{\mathrm{d}\mathbf{P}}{\mathrm{d}u} + 2(\mathbf{P}(u) - \mathbf{p})^T \frac{\mathrm{d}^2\mathbf{P}}{\mathrm{d}u^2} \tag{6.47}$$

The second derivative of the spiral parametrization is

$$\frac{\mathrm{d}^2\mathbf{P}}{\mathrm{d}u^2} = 6k\operatorname{sgn}(u)\, u \begin{bmatrix} -\sin\left(\operatorname{sgn}(u)\, u^2 + \chi_0\right) \\ \cos\left(\operatorname{sgn}(u)\, u^2 + \chi_0\right) \end{bmatrix} + 4ku^3 \begin{bmatrix} -\cos\left(\operatorname{sgn}(u)\, u^2 + \chi_0\right) \\ -\sin\left(\operatorname{sgn}(u)\, u^2 + \chi_0\right) \end{bmatrix} \tag{6.48}$$

The above expressions allow us to find $\frac{\mathrm{d}J}{\mathrm{d}u} = 0$ with Newtons method by computing

$$u_{i+1} = u_i - \frac{\mathrm{d}J}{\mathrm{d}u} \Big/ \frac{\mathrm{d}^2 J}{\mathrm{d}u^2} \tag{6.49}$$

until $\left|\frac{\mathrm{d}J}{\mathrm{d}u}\right| < \epsilon$ where ϵ is a threshold we set to 0.001. After u_i converges to u^*, we compute $\theta^* = \operatorname{sgn}(u^*)\,(u^*)^2$. If θ^* is between θ_{begin} and θ_{end}, then $\theta_{\text{closest}} = \theta^*$. Otherwise we fallback to using the best sampled value from the initialization.

6.2.4 General path properties

Path velocity

The path velocity at a point t_p is given by

$$\mathbf{v}_p = \frac{\mathrm{d}\mathbf{P}}{\mathrm{d}t} = \frac{\partial \mathbf{P}}{\partial t_p}\dot{t}_p \tag{6.50}$$

Assume the desired velocity is v_d. Then $\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}t}$ gives the direction for the path velocity and v_d gives the magnitude. Based on this we write the path velocity as

$$\mathbf{v}_p = v_d \hat{\mathbf{v}}_p \tag{6.51}$$

$$\hat{\mathbf{v}}_p = \frac{\partial \mathbf{P}/\partial t_p}{\left\|\partial \mathbf{P}/\partial t_p\right\|} \tag{6.52}$$

We note that [22] gives a closed form expression for $\dot{t}_p$ for Fermat-spirals which can be used in (6.50), and the expression they obtain is equivalent to the one above.

Path acceleration

The path acceleration at a point t_p is given by

$$\begin{aligned} \mathbf{a}_p &= \frac{\mathrm{d}^2\mathbf{P}}{\mathrm{d}t^2} \\ &= \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \mathbf{P}}{\partial t_p}\dot{t}_p \\ &= \frac{\partial^2 \mathbf{P}}{\partial t_p^2}\dot{t}_p^2 + \frac{\partial \mathbf{P}}{\partial t_p}\ddot{t}_p \end{aligned} \tag{6.53}$$

Path curvature

Assume a point on the path $\mathbf{P}(t_p)$ has velocity $\mathbf{v}_p = [v_x, v_y]$ and acceleration $\mathbf{a}_p = \dot{\mathbf{v}}_p = [a_x, a_y]$. According to [9], the signed curvature is then given by

$$\kappa = \frac{v_x a_y - v_y a_x}{(v_x^2 + v_y^2)^{3/2}} \tag{6.54}$$

which we can rewrite as

$$\kappa = \frac{1}{\left\|\mathbf{v}_p\right\|^3} \left(\mathbf{v}_p \times \dot{\mathbf{v}}_p\right)_z \tag{6.55}$$

Path courserate

Since the course of a path is given as $\chi_p = \operatorname{atan2}(v_y, v_x)$, the courserate is given by

$$\begin{aligned} \dot{\chi}_p &= \frac{\mathrm{d}\operatorname{atan2}(v_y, v_x)}{\mathrm{d}t} \\ &= \frac{1}{\left\|\mathbf{v}_p\right\|^2} \left(-v_y \frac{\mathrm{d}v_x}{\mathrm{d}t} + v_x \frac{\mathrm{d}v_y}{\mathrm{d}t}\right) \\ &= \frac{1}{\left\|\mathbf{v}_p\right\|^2} \left(\mathbf{v}_p \times \dot{\mathbf{v}}_p\right)_z \end{aligned} \tag{6.56}$$

6.3 Path-following

Path-following is the problem of moving the vehicle so it converges on a path and follows it. Contrary to trajectory-tracking, path-following specifies the path independent of time, so the vehicle is allowed to converge anywhere on the path. In this section we present a guidance system that solves the path-following problem. A full guidance system needs to determine the desired course χ_d and a desired speed v_d, but in this project we have not done any work to determine the desired speed dynamically, so it is given as a constant parameter to the system. The consequences of this are discussed in in Section 6.6.

6.3.1 Course determination

Course determination in guidance systems is a common problem in autonomous systems with many popular solutions. In [13], Fossen provides a survey of different guidance laws for marine crafts. From an engineering perspective, we argue it is desirable to use a simple guidance system, because it is easier to understand their failure modes and limitations. Based on this we decided to implement lookahead-based guidance as presented by [2]. Lookahead-based guidance is a vector-field guidance method, which in its simplest form means that the guidance law has a closed form expression. This is contrary to optimization based methods.

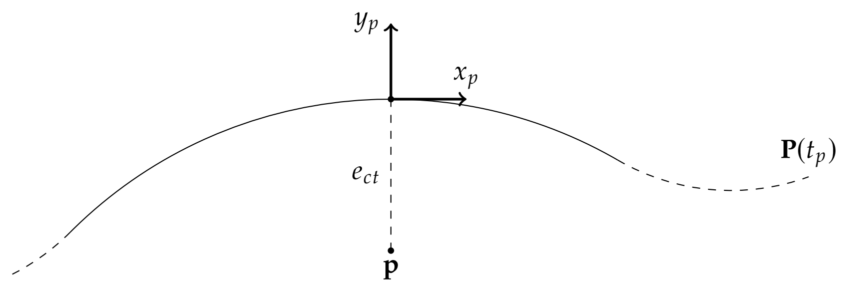


Figure 6.10: Cross-track error and guidance law.

To determine a desired course, we compute the cross-track error e_{ct}, which is defined as the error perpendicular to the path, as illustrated in Fig. 6.10. To do this in software, we first need to determine the path-parameter t_p that is closest to the position of the vehicle. This is done by following the procedure outlined in Section 6.2.3. The position of the path is then $\mathbf{P}(t_p)$, so the position error of the vehicle relative to the path in the inertial frame is

$$\mathbf{e}_p^i = \mathbf{p} - \mathbf{P}(t_p) \tag{6.57}$$

By using the course of the path $\chi_p(t_p)$, we can express the error in the Serret-Frenet frame [12] as

$$\mathbf{e}_p^f = \mathbf{R}_z\left(-\chi_p(t_p)\right)\mathbf{e}_p^i \tag{6.58}$$

The cross-track error is by definition the y-component of $\mathbf{e}_p^f$.

In [2], they propose the guidance law

$$\chi_d = \chi_p + \chi_r(e_{ct}) \tag{6.59}$$

where

$$\chi_r(e_{ct}) = \chi_a \frac{2}{\pi} \operatorname{atan}(-K_p e_{ct}) \tag{6.60}$$

The approach angle χ_a is a tunable parameter which controls the angle the vehicle approaches the path at when the cross-track error is large. Figure 6.11 illustrates the vector field for following a straight line path and the desired trajectory that the vehicle should follow.

A problem with (6.59) and (6.60) is that is designed for straight paths. During turns, the vehicle will thus only attempt to converge on a line tangential to the closest point on the curve, which could lead to significant cross-track error. To counteract this, we propose an extension of (6.60), which is to compute the path-curvature κ and feeding it forward to the guidance law with

$$\chi_d = \chi_p + \chi_r(e_{ct}) + K_\kappa v_d \kappa \tag{6.61}$$

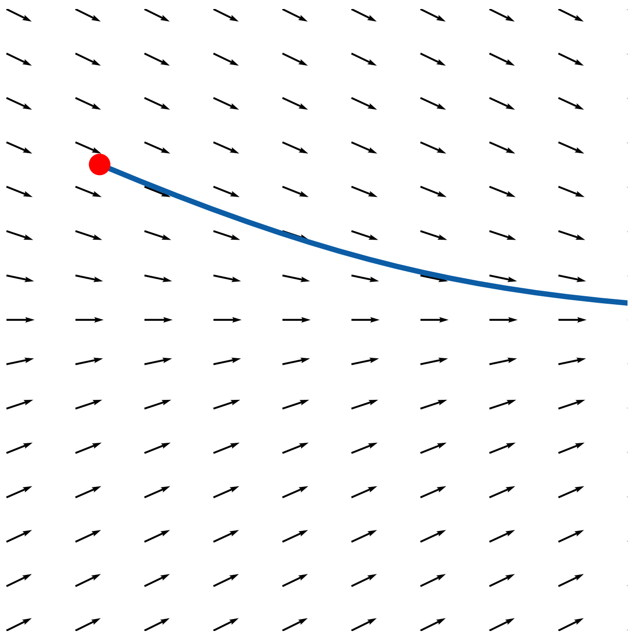


Figure 6.11: Vector field converging on straight line path defined by $y = 0$.

where K_κ is a tunable parameter controlling how strong the curvature term should be. We note that we have not seen this type of curvature modification in the path-following litterature. If we compare (6.55) with (6.56), we see that we can write $v_d\kappa = \dot{\chi}_p$. This means that we can interpret $\chi_p + K_\kappa v_d \kappa = \chi_p + K_\kappa \dot{\chi}_p$ as predicting the path-curvature into the future, which we suspect may improve tracking on curved paths.

6.4 Parameters

The parameters for guidance system are given in Table 6.1.

Table 6.1: Guidance system parameters.

$\kappa_{\max}$	$0.5\,\mathrm{m}^{-1}$
K_p	0.5
K_κ	0
χ_a	60 deg
v_d	$10\,\mathrm{km\,h}^{-1}$

6.5 Results

We test the guidance system on two distinct cases. In the first case, the vehicle starts far away from a straight line path and should converge on it. In the second case, the vehicle is presented with the lap shown in Fig. 6.12, and it should follow it counter-clockwise. When following the laps, we start the vehicle in the origin so that the initial error does not dominate the results. Since the second case contains both right and left, small and big turns, and long and short straight segments, we argue it is a representative example of what the vehicle should be capable of, so we can use it to analyse the effect of different system parameters.

6.5.1 Straight line convergence

The straight line path starts in the origin and follows the x-axis positively. The vehicle is initialized at [20, 20]. Also the initial yaw angle is 180 deg and it should be controlled to 0 deg. The results for this case are given in Fig. 6.13. Looking at the cross-track error in Fig. 6.13a, we see that it starts at 20 m, and after about 20 s it converges on zero. The yaw, which is controlled to be the same as the path course, converges in about 10 s, which can be seen in Fig. 6.13b. Based on this simple case, we are confident that the system is able to converge on a path when it starts far away. But we suspect that this only applies when the path is straight. We believe that the guidance system provides no guarantees that it is able to converge on curved paths, because it is designed for straight paths.

6.5.2 Baseline case

To evaluate the performance of the GC system and compare the effect of different parameters, we setup a baseline case. The baseline we use is the lap shown in Fig. 6.12 with curvature 0.5 m^{-1}. Additionally, we set the desired speed of the

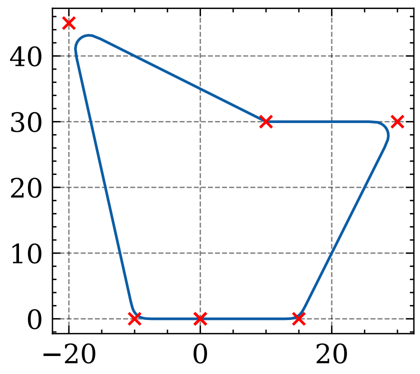
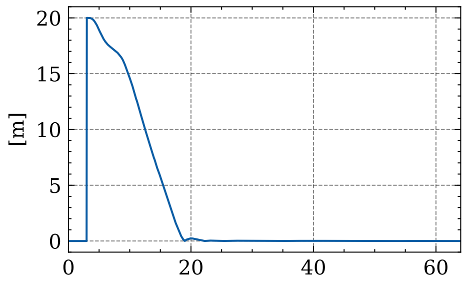
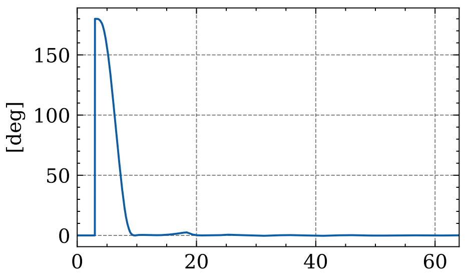


Figure 6.12: Lap with curvature at 0.5.



(a) Cross-track error.



(b) Yaw error.

Figure 6.13: Straight line converge results.

vehicle to 10 km h^{-1}. The results for this case are given in Fig. 6.15 and Table 6.2, and for visual reference, a screenrecording of the case is uploaded to https://youtu.be/m5kxQ7tw4iU.

Looking at the cross-track error in Fig. 6.14a, we see that it has many spikes corresponding to the turns in the path, and the peak cross-track error was 1.030 m. The vehicle is able to maintain track of the path during straight line driving. This indicates that the guidance system is suboptimal during the turns. Even the small third turn at $(10, 30)$ leads to significant cross-track errors.

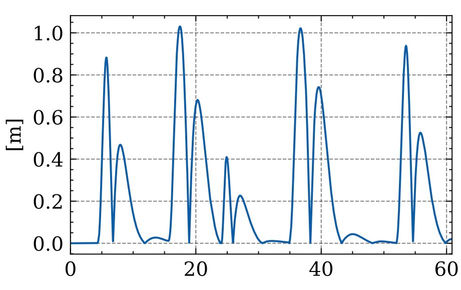
6.5.3 Effect of reference model

A possible explanation for poor performance during turns is that the wheels are not allowed to turn fast enough. To investigate this, we disable the reference model and rerun the experiment. The results are given in Fig. 6.17 and Table 6.3.

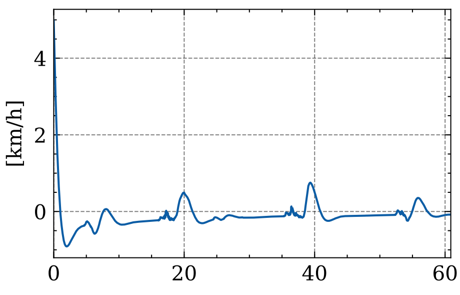
Without the wheel reference model, the mean cross-track error decreased from 0.226 m to 0.098 m, and the effect of this is clear in the trajectory plot in Fig. 6.16c where it is hard to see any significant overshoot even during turns. The cost is a large increase in chatter, from 70.666 N m to 518.598 N m, and similarly a large increase in peak steering torque from 2.582 kN m to 14.108 kN m. In a real system, these costs are presumably not acceptable, so removing the reference model is not a viable option.

The fact that the GC system has better performance when the wheel reference model is removed is an indication that we do not have sufficient bandwidth separation between the control layers. The upper control layers expect the wheel control layer to be able to follow more aggressive commands than it is capable of. To test this hypothesis, we increase the natural frequency of the wheel steering angle reference model from 4 rad s^{-1} to 6 rad s^{-1}. The results for this case are given in Fig. 6.19 and Table 6.4.

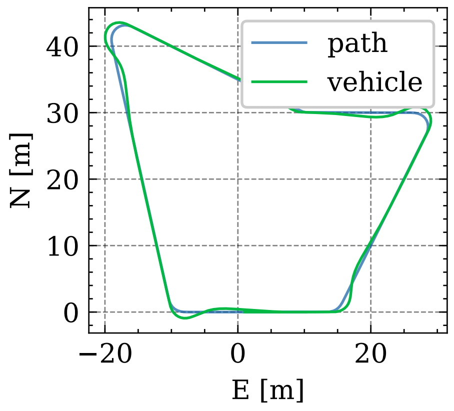
Making the reference model faster by increasing the natural frequency improves the cross-track error from 0.226 m to 0.143 m. As with the case above, we



(a) Cross-track error.



(b) Speed error.



(c) Trajectory.

Figure 6.15: Guidance plots on a simple lap.

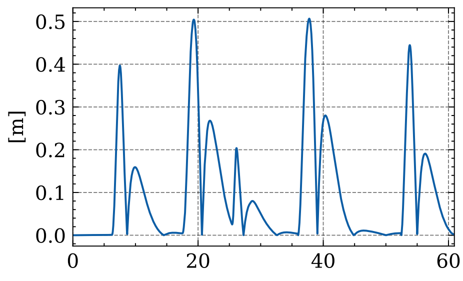
Table 6.2: Guidance performance metrics on a simple lap.

Metrics	
Mean cross-track error [m]	0.226
Peak cross-track error [m]	1.030
Mean speed error [km/h]	0.243
Steering chatter (front left) [Nm]	70.666
Peak steering torque (front left) [kNm]	2.582
Peak steering rate (front left) [deg/s]	108.929

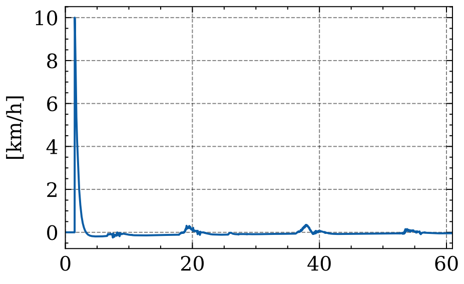
see an increase in chatter and peak steering torques compared with the baseline case, but it is significantly less than when the reference models are removed.

6.5.4 Effect of speed and curvature

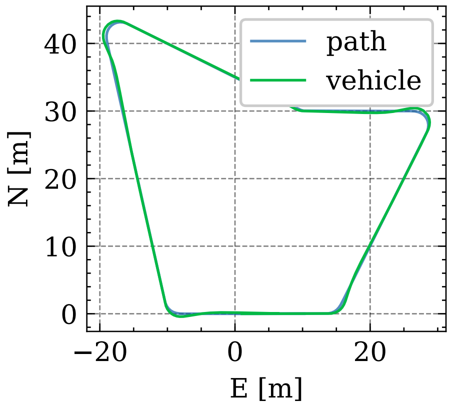
Another strategy for improving performance during turns is to decrease the speed of the vehicle. There are several reasons this may improve performance. When



(a) Cross-track error.



(b) Speed error.



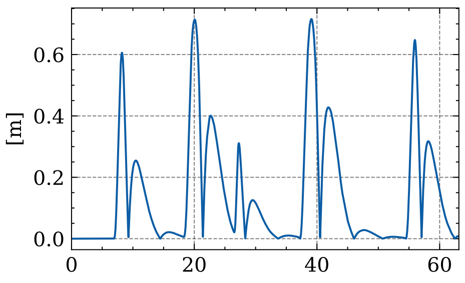
(c) Trajectory.

Figure 6.17: Guidance plots on a simple lap without wheel reference model.

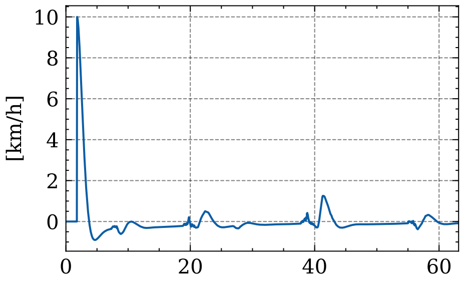
Table 6.3: Guidance performance metrics on a simple lap without wheel reference model.

Metrics	
Mean cross-track error [m]	0.098
Peak cross-track error [m]	0.507
Steering chatter (front left) [Nm]	518.598
Peak steering torque (front left) [kNm]	14.108
Peak steering rate (front left) [deg/s]	108.428

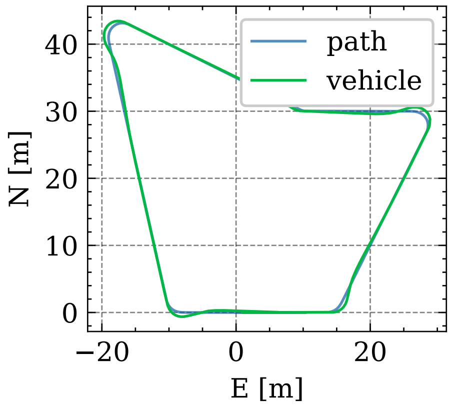
going slower, the dynamics of the vehicle have more time to converge on the controller setpoints, so transient effects matter less. Another reason is that the friction forces that the vehicle needs to generate in a turn are related to the circular acceleration of the vehicle, which is given by v^2/r where v is the circular speed and r is the radius of rotation. By decreasing the speed, the wheels are not required to generate as much friction.



(a) Cross-track error.



(b) Speed error.



(c) Trajectory.

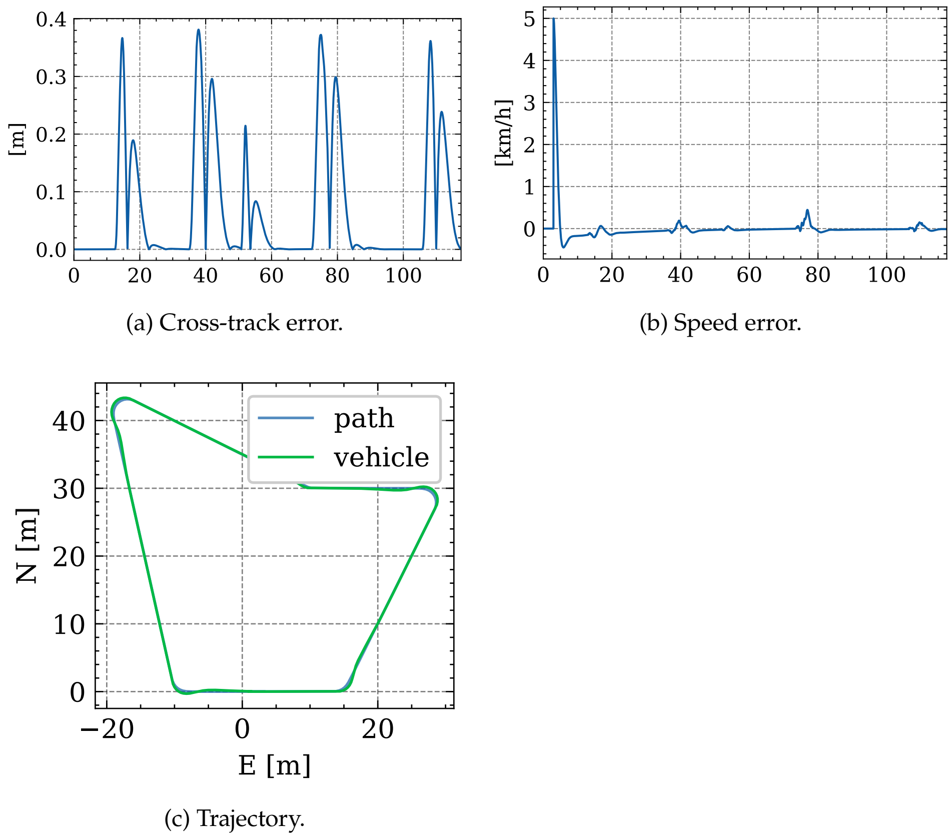
Figure 6.19: Guidance plots on a simple lap with faster wheel reference model.

Table 6.4: Guidance performance metrics on a simple lap with faster wheel reference model.

Metrics	
Mean cross-track error [m]	0.143
Peak cross-track error [m]	0.716
Steering chatter (front left) [Nm]	105.184
Peak steering torque (front left) [kNm]	5.904
Peak steering rate (front left) [deg/s]	107.025

To test the effect of speed, we reduce the desired speed of the vehicle from 10 km h^{-1} to 5 km h^{-1} and rerun the same lap presented above. The results are given in Fig. 6.21 and Table 6.5. By decreasing the desired speed of the vehicle, the peak cross-track error drops from 1.030 m to 0.381 m.

Note that since $v^2/r = \kappa v^2$, there is a relation between speed and curvature. Based on this formula, we hypothesize that by doubling the speed to 20 km h^{-1} and



(a) Cross-track error.

(b) Speed error.

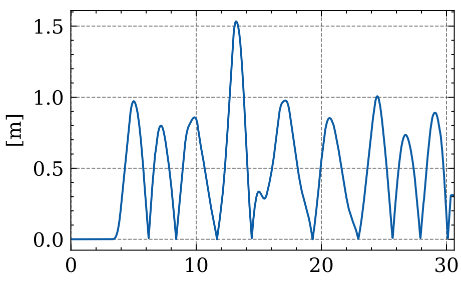
(c) Trajectory.

Figure 6.21: Guidance plots on a simple lap with lower speed. Desired speed is $5\,\mathrm{km\,h^{-1}}$.

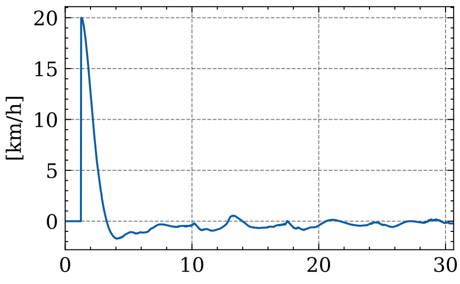
Table 6.5: Guidance performance metrics on a simple lap with lower speed. Desired speed is $5\,\mathrm{km\,h^{-1}}$.

Metrics	
Mean cross-track error [m]	0.069
Peak cross-track error [m]	0.381
Steering chatter (front left) [Nm]	48.762
Peak steering torque (front left) [kNm]	3.790
Peak steering rate (front left) [deg/s]	88.277

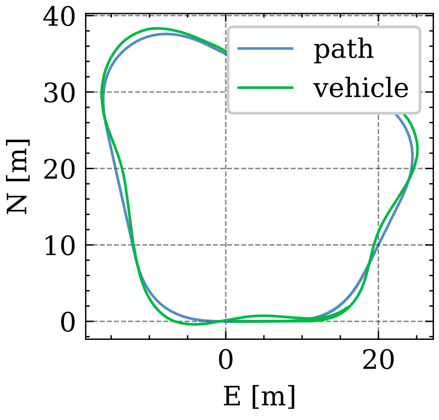
quartering the curvature to $0.125\,\mathrm{m^{-1}}$ as compared with the baseline, we should obtain a comparable tracking performance. The results for this case are presented in Fig. 6.23 and Table 6.6. Looking at the performance metrics in Table 6.6, and comparing them with Table 6.2, we see that the performance is significantly worse in this case. The mean cross-track error increased from 0.226 m to 0.472 m. If we



(a) Cross-track error.



(b) Speed error.



(c) Trajectory.

Figure 6.23: Guidance performance on a lap with high speed (20 km h^{-1}) and low curvature (0.125 m^{-1}).

Table 6.6: Guidance performance metrics on a lap with high speed (20 km h^{-1}) and low curvature (0.125 m^{-1}).

Metrics	
Mean cross-track error [m]	0.472
Peak cross-track error [m]	1.533
Steering chatter (front left) [Nm]	285.706
Peak steering torque (front left) [kNm]	3.674
Peak steering rate (front left) [deg/s]	73.013

look at the cross-track error plot in Fig. 6.22a, we see the error rarely converges to zero. This is probably because there are not enough long straight line segments in the path for the vehicle to stabilize on. In either case, this result seems to disprove the hypothesis that the curvature and speed have the relationship given above. We argue that the curvature-speed relationship is only relevant when the friction

Table 6.8: Guidance performance with curvature feed-forward.

	$K_\kappa = 0$	$K_\kappa = 0.1$	$K_\kappa = 0.3$
Mean cross-track error [m]	0.226	0.182	0.327
Peak cross-track error [m]	1.030	0.869	1.860
Steering chatter (front left) [Nm]	70.666	221.300	272.665
Peak steering torque (front left) [kNm]	2.582	7.092	12.068
Peak steering rate (front left) [deg/s]	108.929	114.874	136.908

Table 6.9: Guidance performance with curvature feed-forward and desired speed $5\,\mathrm{km\,h^{-1}}$.

	$K_\kappa = 0$	$K_\kappa = 0.1$
Mean cross-track error [m]	0.069	0.106
Peak cross-track error [m]	0.381	0.690
Steering chatter (front left) [Nm]	48.762	151.560
Peak steering torque (front left) [kNm]	3.790	4.798
Peak steering rate (front left) [deg/s]	88.277	103.689

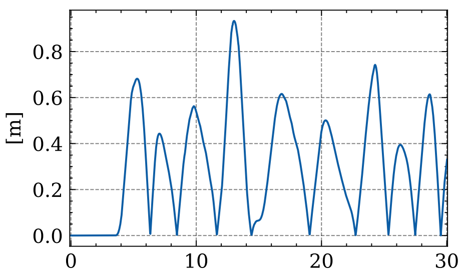
forces are the limiting factor. If the wheel is able to operate in a region of static friction, then it may be able to turn faster than dictated by the relationship above. Thus we suspect that the reason the performance is significantly worse in this scenario, is that the guidance system requires faster wheel and vehicle dynamics in order to track the path at $20\,\mathrm{km\,h^{-1}}$. Using the faster reference model, we obtain the results in Fig. 6.25 and Table 6.7. It performs significantly better with mean cross-track error 0.302 m, as opposed to 0.472 m, but it is still worse than the baseline which had 0.226 m.

6.5.5 Effect of curvature feedforward

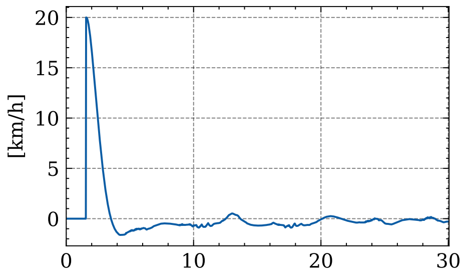
In Section 6.3, we proposed using the curvature of the path to compensate for turns, but we set the curvature feedforward gain to $K_\kappa = 0$, so it has not been active in any of the results presented above. In Table 6.8, we present performance metrics with $K_\kappa = 0.1$ and $K_\kappa = 0.3$, respectively. For quick reference, the baseline performance metrics are restated also restated.

Using $K_\kappa = 0.1$ there is a improvement in the peak cross-track error from 1.030 m to 0.869 m. When we increase the gain further to 0.3, the peak error increases to 1.860 m. Interestingly, we found the using $K_\kappa = 0.1$ and decreasing the desired speed to $5\,\mathrm{km\,h^{-1}}$ led to worse performance. Results for that case are given in Table 6.9. When the desired speed is $5\,\mathrm{km\,h^{-1}}$ the mean cross-track error increased from 0.069 m to 0.106 m, and likewise for the peak cross-track error.

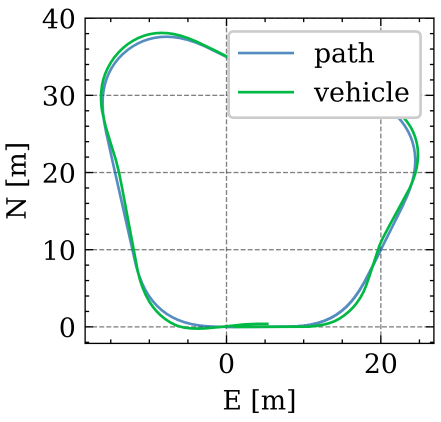
The results shown above indicate that there is a relationship between the cur-



(a) Cross-track error.



(b) Speed error.



(c) Trajectory.

Figure 6.25: Guidance plots on a lap with high speed (20), low curvature (0.125) and faster wheel dynamics.

Table 6.7: Guidance performance metrics on a lap with high speed (20), low curvature (0.125) and faster wheel dynamics.

Metrics	
Mean cross-track error [m]	0.302
Peak cross-track error [m]	0.934
Steering chatter (front left) [Nm]	357.142
Peak steering torque (front left) [kNm]	4.746
Peak steering rate (front left) [deg/s]	75.715

vature feedforward gain and the desired speed. By finding that relationship and tuning the feedforward gain to the speed, we can increase tracking performance during turns. But if we get the tuning incorrect, then the performance of the guidance system worsens. Because of this, we decided to keep $K_\kappa = 0$, so unless otherwise stated, the curvature feedforward term is disabled.

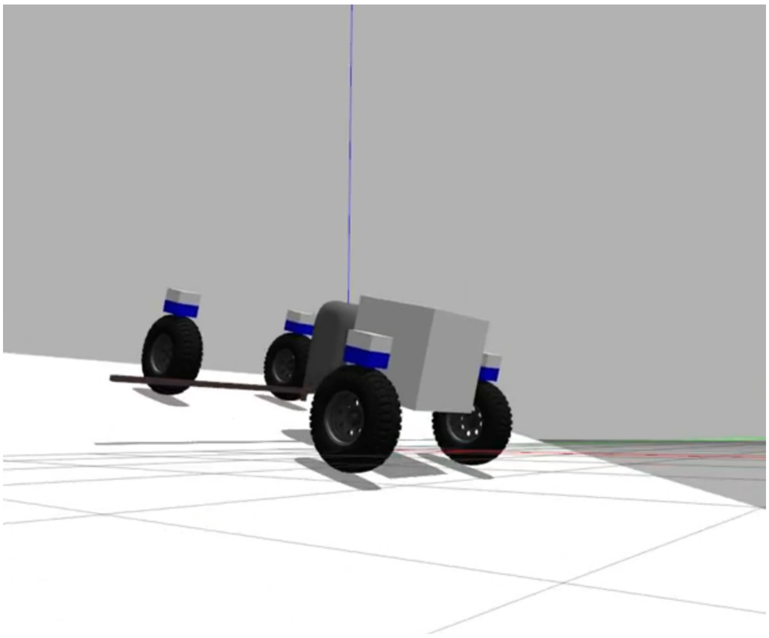


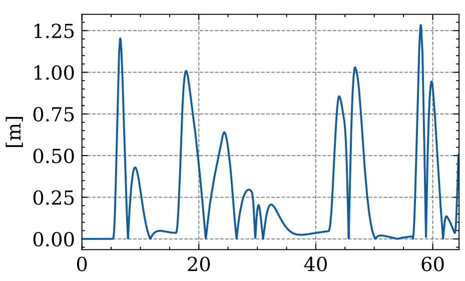
Figure 6.26: Vehicle on sloped terrain. Picture from beginning of `https://youtu.be/9PLcTj5GGQw`.

6.5.6 Effect of sloped terrain

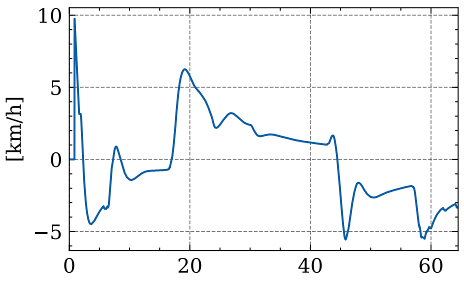
The vehicle model the controller is based on has several simplifying assumptions. One of the assumptions is flat terrain. A consequence of this is that the controllers do not try to compensate for the terrain directly, and instead rely on integral action in the upper control levels. All the results presented above are with flat terrain, but to investigate the effect of unmodelled disturbances, we rerun the simulations with the lap recreated on a surface with slope 15 deg. For reference, a video of this is uploaded to `https://youtu.be/9PLcTj5GGQw`, and a screenshot is shown in Fig. 6.26. The results are presented in Fig. 6.28 and Table 6.10.

The results show an increase in mean cross-track error from 0.226 m to 0.263 m, and the peak cross-track error increased from 1.030 m to 1.284 m. The biggest difference is seen in the speed error. In the baseline case in Fig. 6.14b, the speed error converges quickly to a value close to zero, and the mean error was 0.243 $\mathrm{km\,h^{-1}}$. When the vehicle works in sloped terrain, the speed error in Fig. 6.27b takes a long time to converge, and the mean speed error is 2.468 $\mathrm{km\,h^{-1}}$. This can be clearly seen at 0:32 in the video when the vehicle changes from going slightly downhill to directly uphill, and the vehicle almost comes to a standstill.

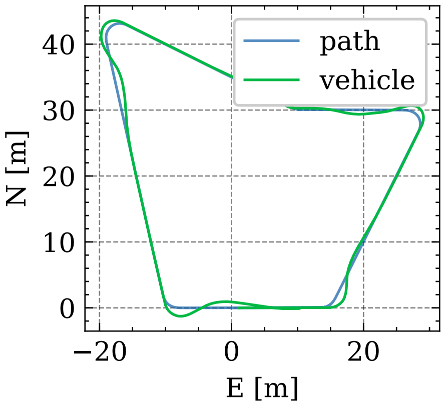
We think that the speed tracking performance can be improved by making the integral effect of the speed controller greater. Looking at the period from 20 s to 40 s in Fig. 6.14b, which corresponds to 0:35 to 1:00 in the video, we can see that



(a) Cross-track error.



(b) Speed error.



(c) Trajectory.

Figure 6.28: Guidance plots on a lap sloped terrain.

Table 6.10: Guidance performance metrics on a lap sloped terrain.

Metrics	
Mean cross-track error [m]	0.263
Peak cross-track error [m]	1.284
Mean speed error [km/h]	2.468
Steering chatter (front left) [Nm]	222.915
Peak steering torque (front left) [kNm]	7.117
Peak steering rate (front left) [deg/s]	102.345

the vehicle slowly gains speed as it travels uphill. It stands to reason that a more aggressive integral action would correct for the slope faster.

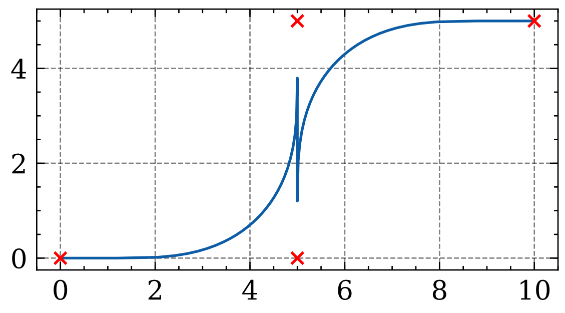


Figure 6.29: Fermat smoothing invalid waypoint/curvature configuration.

6.6 Limitations

There exists several problems with the path-smoothing system we have presented. One problem is that there is no error checking on the waypoints, and if the waypoints are too close together then the smoothened path will be invalid. This is illustrated in Fig. 6.29. It may be possible to modify the algorithm to prevent this, but this is something we leave for future work. There are two ways to mitigate this. We can

1. increase the maximum curvature to allow sharper turns, or
2. increase the distance between the waypoints.

The results above demonstrate that the speed of the vehicle should have an effect on the maximum curvature. 4WIS vehicles are even able to handle infinite curvature when stationary. A problem is that the speed is specified as a constant parameter and not computed dynamically from the path. A possibility is to add curvature and speed data to each waypoint. Then the path-smoothing would be able to use the maximum curvature at each waypoint, and we could interpolate the speed between the waypoints.

Another limitation is that the we do not support self-intersecting paths. This is because when we compute the distance to the path, we always consider the entire path. This was done to keep the implementation simple, but a real implementation has to solve this problem. In Ch. 12 of [2], they propose a software architecture for dealing with this and many other path-following related problems for small Unmanned Aerial Vehicles (UAVs), and the architecture seems like it would extend to ground based vehicles as well.

The curvature of Fermat spiral is continuous, but it changes rapidly close the origin as illustrated in Fig. 6.4. In practice, this means that at the start of a turn, the vehicle has to rapidly change direction, which requires rapid wheel responses. To reduce these problems, we can reduce the maximum curvature parameter. But since the curvature of Fermat spirals starts to decrease after turning 15 deg, decreasing the curvature parameter will increase the size of the turns significantly. Based on this we suggest that instead of using Fermat spirals throughout the turn, we should instead use Fermat spirals to transition into and out of circular segments. Fraichard and Scheuer [14] develops an algorithm for this using Clothoids instead of Fermat spirals, but we believe a similar algorithm could be developed for Fermat spirals.

When we tested the vehicle on sloped terrain, we found that it was able to maintain track, but it struggled maintaining speed. This is because the controllers do not account for the extra force needed to drive uphill. We believe this problem could be tuned away with more integral action on the speed controller, but we should also model this extra resistance so that we can use it actively in the controller.

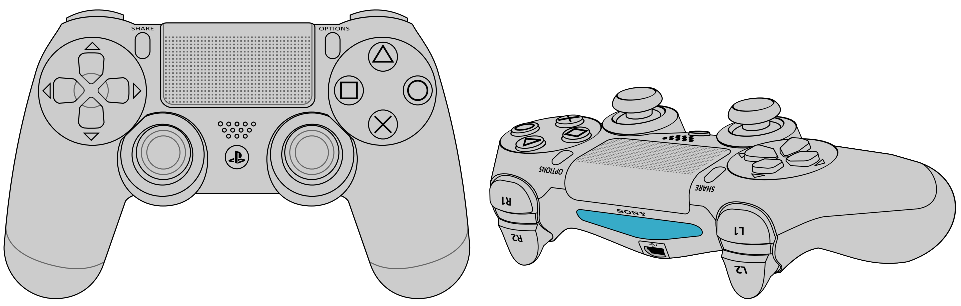
7 Example Applications

In this chapter, we want to take a step back and consider how the control systems developed in this project can be used in a complete system. To do this we present two working protoype applications. In the first application, we connect the system to a Playstation controller and use it to control the vehicle manually. In the second application we use an open source flight control software to control the vehicle via waypoints. The goal of creating these example applications is primarily to sketch out what features a complete system may contain. Because this has not been a big focus of this project, we decided to take several shortcuts in the implementation. So the implementations we present are not robust or finalized, but they allow us to experiment with the vehicle interface.

7.1 Manual control with Playstation controller

The Playstation 4 controller, called Dualshock 4, is typically used for video games. It is illustrate in Fig. 7.1 with a button layout. Using Bluetooth we connect it to the computer. Linux recognizes it as a joystick input device, so we could use the Linux input driver or a more high-level API like SDL2 to communicate with it. Fortunately, there is already a ROS2 package called joy which implements joystick drivers for many controllers, including the Dualshock 4. It publishes the controller state as a `sensor_msgs/msg/Joy` message. To control the vehicle we setup the left joystick, denoted L3, to control the courserate and yawrate, and the right joystick, denoted R3, to control the yawrate. We also programmed R2 and L2 triggers on the back to accelerate and brake, respectively.

To be specific, the manual control module is implemented in the operation layer, as illustrated in Fig. 3.6. It computes a desired course χ_d, desired speed v_d, and desired yawrate $\dot{\psi}_d$ for the vehicle. Moving the left joystick left and right creates a signal from -1 to +1, denoted L3$_{LR}$, and likewise for the right joystick.



(a) Top. CC BY 3.0 (b) Rear. CC BY 3.0

Figure 7.1: Dualshock 4 controller layout.

The desired courserate is given by the left joystick as

$$\dot{\chi}_d = \mathrm{L3}_{LR}. \tag{7.1}$$

The desired yawrate is given as

$$\dot{\psi}_d = \dot{\chi}_d + 0.2\mathrm{R3}_{LR} = \mathrm{L3}_{LR} + 0.2\mathrm{R3}_{LR} \tag{7.2}$$

The reason both joystick feed into the yawrate, is that then yaw will follow the movement direction of the vehicle. This may not be desired in all situations, but for this prototype we found that it made the most intuitive sense. When the L2 or R2 triggers are pressed, we update the desired speed using

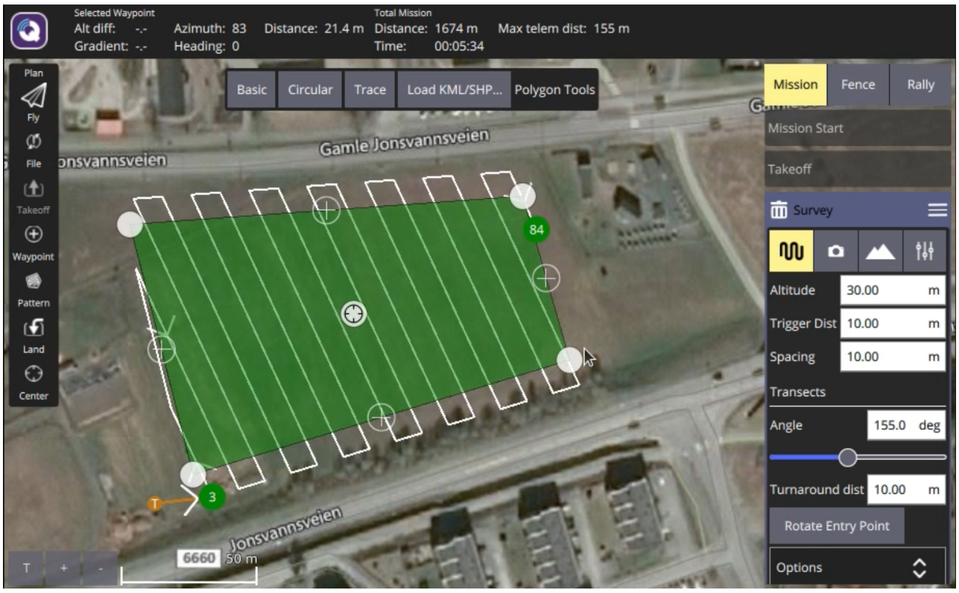
$$\dot{v}_d = \begin{cases} 1 & \text{if R2 pressed} \\ -0.8v_d & \text{if L2 pressed} \\ 0 & \text{otherwise} \end{cases} \tag{7.3}$$

A video of the vehicle being controlled with this button layout is uploaded to `https://youtu.be/00WzG6nqd84`. It is hard to tell from the video, but the joystick control presented above gives an intuitive way to control the vehicle, but there are a couple of issues with it. One problem is that when the vehicle is stationary, there is no way to tell which direction it is going to accelerate. This is evident at 0:52 in the video were we press the R2 to accelerate, and the vehicle reverses. Another problem is that the we don't scale down the joystick input for courserate in (7.1), which could lead to aggressive driving.

7.2 Waypoint control with QGroundControl

Manual control is useful for maneuvering the vehicle over short distances or through tight spaces, but to execute long-running farming operations, we need a way to design a mission that the vehicle should execute. Based on prior experience with UAVs, we knew that QGC provides a powerful waypoint based mission planning system. QGC is designed for aerial vehicles, so most of its features are not applicable for farming vehicles. The only aspect of QGC that we have used is the mission planner, which provides a map of the world and allows us to draw waypoints on the map. A video of QGC being used to create a mission is uploaded to `https://youtu.be/D7ouowuOn14`. In the video, we use the survey feature of QGC, which creates a lawnmover pattern across a large area. Figure 7.2 shows a screenshot from the video close to the end when the mission is finalized.

QGC is a fully capable ground station, and it is typically used to communicate with and control the vehicle during operation. But setting up this with the current system requires a substantial amount of work, so we decided not go this route. To get the waypoints out of QGC and into the guidance module, we save the



Selected Waypoint
Alt diff: -.- Azimuth: 83 Distance: 21.4 m
Gradient: -.- Heading: 0

Total Mission
Distance: 1674 m Max telem dist: 155 m
Time: 00:05:34

Altitude 30.00 m
Trigger Dist 10.00 m
Spacing 10.00 m
Angle 155.0 deg
Turnaround dist 10.00 m

Figure 7.2: Waypoint setup in QGC.

mission as a plan-file, which is a custom JSON based file format used by QGC. We then created a small library to parse the plan-file and extract the waypoints as latitude-longitude pairs. Since the guidance system expects Cartesian waypoints, we need to convert the latidude-longitude pairs to Cartesian coordinates. There are many ways to do this properly, taking into account the curvature of Earth, terrain features, and more, but since this is just a prototype, we decided to do something simpler. We define the first waypoint as the origin. To get ENU coordinates, we define the x-axis as east, which is positively increasing latitude λ, and the y-axis as north, which is positively increasing longitude ϕ, as illustrated in Fig. 7.3. Assume Earth is a sphere with radius $R_{\text{Earth}} = 6371\text{km}$. This allows us to compute the great circle distance between the waypoints.

Given two latitude-longitude pairs, (λ_0, ϕ_0) and (λ, ϕ), the great circle distance is the shortest on-surface distance between them. Several formulas exist to compute it. In [15], and reference therein [33], they compute the great circle distance d as

$$d = R_{\text{Earth}} c \tag{7.4}$$

$$c = 2\,\text{atan2}(\sqrt{a}, \sqrt{1-a}) \tag{7.5}$$

$$a = \sin^2\left(\frac{\phi - \phi_0}{2}\right) + \cos\phi_0 \cos\phi \sin^2\left(\frac{\lambda - \lambda_0}{2}\right) \tag{7.6}$$

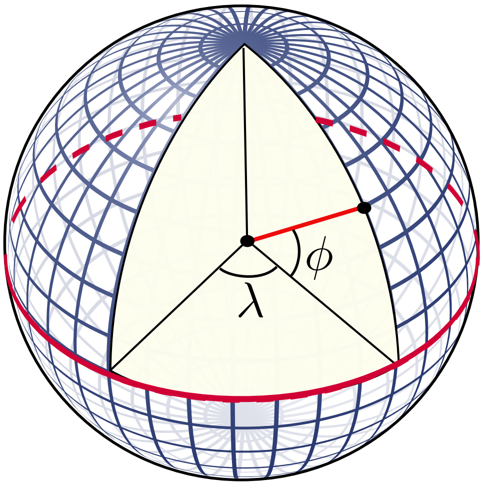


Figure 7.3: Latitude (λ) and longitude (ϕ). CC BY-SA 3.0

To compute the Cartesian equivalent (x, y) of waypoint (λ, ϕ), we use

$$x = d(\lambda_0, \phi_0, \lambda, \phi_0)\operatorname{sgn}(\operatorname{ssa}(\lambda - \lambda_0)) \tag{7.7}$$

$$y = d(\lambda_0, \phi_0, \lambda_0, \phi)\operatorname{sgn}\big(\operatorname{ssa}(\phi - \phi_0)\big) \tag{7.8}$$

where (λ_0, ϕ_0) is the first waypoint, and $d(\cdot)$ is computed using (7.4) to (7.6).

After saving the mission-plan file, the waypoint module in the operation package sends the waypoints to the guidance system for the vehicle to follow. This process is illustrated in a video that can be found at `https://youtu.be/Sdw67MjuN_U`. A screenshot from the video is given in Fig. 7.4. In the screenshot we can see QGC being used to make the path, Gazebo being used to visualize the vehicle, and RViz being used to give a live preview. The video covers the whole path, but it is sped up since at normal speed it takes about 14 minutes to cover.

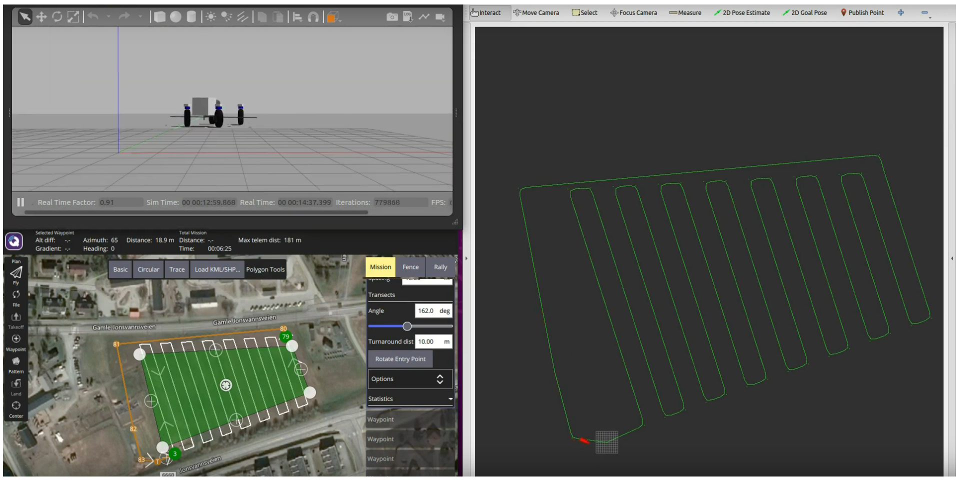


Figure 7.4: Waypoint control with QGC. Top left is Gazebo. Bottom left QGC. Right half is RViz.

8 Discussion & Conclusion

In this project, we have developed and evaluated a hierarchical control system for 4WIS vehicles, and we have created two example applications to see how they could work in practice. In this final chapter we retrace the report and discuss what works well and what needs improvement. We also suggest features that our system lacks that we believe are needed in a real implementation.

8.1 System design

An important focus of this work has been to create a modular and extensible software architecture for the system. This was done by separating the motion control task into independent layers. The wheel control layer is responsible for control of a single wheel, and the vehicle control layer is responsible for coordinating the wheels. This provides a decoupled software architecture, but there could be issues moving this from simulation to real hardware.

One problem with the system design is error handling. A more complete system would be equipped with sensors and code to detect errors during operation. Take for instance the case of excessive wheel slip on one of the wheels. Excessive wheel slip can be detected by comparing measurements from a wheel encoder to the inertial measurements of the vehicle. If the wheel control system detects this error condition, it can either handle it internally, or it can relegate the task to the vehicle control system above. If it handles it internally, then the wheel controller might need to ignore references from the vehicle controller for a period of time, which could lead to errors in the vehicle controller. If it relagates error handling to the vehicle controller, then we lose some decoupling between the layers, because the vehicle control layer then needs to be able to control the wheels aswell. It is possible to make the case that we can improve fault tolerance and error handling, and thus robustness, by creating a more tightly coupled, monolithic motion control system, but a cost of this would be a system that is harder to understand and modify. The hierarchical motion control system is optimized for readability and normal operation. Since it mostly uses standard control techniques, it is easier to understand and tune its parameters.

There are many practical aspects that we have not accounted for in this work. From a control perspective, the two main things we have neglected are available onboard sensors and actuator limitations. In the detailed system arhitecture in Fig. 3.6, we show where a state estimation module would fit in, but since we have only worked with simulator cases we have not needed to implement state estimation. In the simulator we have access to noise-free data on every imaginable state in the system, but this is unrealistic because many states cannot be measured directly so they have to be estimated. An example of where this could be a problem is the angular velocity controller. To control the angular velocity we proposed a proportional controller, and we argued that based on the passivity of the system it should be able to follow constant references. To improve tracking of time-varying references, one could add a derivative effect, and this would probably work in theory and simulator cases. But adding derivative effect to the angular velocity controller, assumes that we are able to estimate the angular acceleration of the wheel. If the wheel is equipped with a typical wheel encoder, which can only measure angle and angular velocity, then the acceleration estimate will be noisy. Over the course of this work, we have attempted to limit ourselves to only use states that we believe are possible to estimate or measure, but this would have to be revised once a more detailed specification of the sensors are in place.

Another aspect of the system we have not considered is braking. The wheel controller calculates a drive torque τ_d which we input to the motor. But often wheels have a separate actuator for braking, and this could be relevant particularly for emergency braking. Emergency braking, for instance in the case that a person or an animal walks in front of the vehicle, does not fit neatly in the system decribed here. The way we would implement it now is that the guidance system would send a speed reference $v_r = 0$ to the vehicle control system. This would then need to propagate through the control hierarchy and reference models, and the vehicle would smoothly come to a standstill. Based on the step response test of the speed controller, this could take as much as 5 s, which means that if the average speed of the vehicle was $10\,\mathrm{km\,h^{-1}}$, then it would have travelled approximately 13 m since the braking was initiated. For normal operation this may be acceptable, but emergency braking might need to use more of the capabilities of the system to brake as fast as possible. This could involve using friction estimation to maximize the braking force from each wheel, and purposefully misaligning the wheels to create an even larger braking force on the vehicle.

8.2 Performance

To evaluate the performance of the control system there are many metrics we can look at. The primary goal of this project has been to find a control system arhictecture suitable for the AutoAgri vehicle. Instead of fine-tuning the controllers to obtain optimal simulator performance, we decided it was more beneficial to investigate how the different tuning parameters affect the system, and to look for problems that need to be mitigated in a full system.

The results presented in the Chapter 6 show that there are several issues with the current system. A key issue is that the speed of the vehicle has a strong influence on how sharp turns it is able to make. By reducing the speed from $10\,\mathrm{km\,h^{-1}}$ to $5\,\mathrm{km\,h^{-1}}$, we could reduce the mean cross-track error from 0.226 m 0.069 m on the baseline case. During the straight line segments, the vehicle is able to maintain track with centimeter precision, even at high speeds. The problem that needs to be addressed is that the speed of the vehicle should not be a constant parameter to the system. The results show that if we change the speed dynamically, increasing it for long straight segments and reducing it for the turns, the vehicle should be able to maintain high tracking performance on the entire path. From an operational perspective, it makes sense that the operator should be able to specify the speed, so we think a good solution is embed the desired speeds into the waypoints, and interpolate the speed between the waypints. A related issue is that the curvature parameters are the same for all the turns. The path-smoothing algorithm supports independent curvature parameters without any modifications, so the main problem is deciding how the curvatures should be specified. Again, we believe a good option is to embed the curvature parameter into the waypoints, since then the operator gets more control of how the path will look. One could argue that we should also redesign the path-follower to be capable of following curved paths, because the system we proposed is only designed to converge on straight paths. While this would probably improve performance during turns, we suspect the main problem lies in speed determination. Typical farming operations will involve driving in straight lines majority of the time, so based on this we do not think it is necesarry to use a more sophisticated course-law to account for turns.

Another issue that we need to address is chattering caused by the steering angle controller. Chatter is mainly a problem when the steering angle controller tries to track a time-varying reference. Looking at step response in Fig. 5.4c, we see that there is almost no chatter on the control signal once the system has stabilized, but in the time-varying case of Fig. 5.6c the control signal never stabilizes. In Section 5.4, we argued that there is room to increase the softregion parameter ϵ to reduce chatter without sacrificing performance significantly. The main reason we get significant chatter is that the steering resistance bound $\left|M_f\right| \leq M$ is estimated offline, and in order to guarantee robustness the estimate cannot be a tight bound. The steering angle SMC gets its robustness by adding margins on top of margins, and the end-result is a controller which is designed to overcompansate for the steering resistance. To reduce chatter, we could estimate M online, which would hopefully give a tighter bound, leading to less chatter. We found that the wheel reference model also had a big impact on chatter. When we removed it from the baseline case, the chatter increased from 70.666 N m to 518.598 N m. This is probably because removing the reference model causes large jumps in the reference to be sent to the controller. Before this controller can be deployed, we need to determine what level of chatter is acceptable. Since chatter causes wear and tear on electromechanical system, and it is a known problem of SMC, we recommend that alternative robust control strategies should be investigated as well.

We observed that the performance of the motion control system was strongly dependent on the parameters to the wheel reference model. This means we need to ensure bandwidth separation between the control layers, either by making the wheel control layer faster, or the vehicle control layer slower, but the bandwidth of each layer should be identified by doing system identification on the vehicle. The

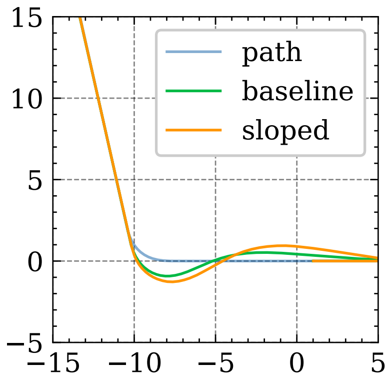


Figure 8.1: Baseline and sloped terrain trajectories.

reference model parameters that we used in this project were selected based on physical intuition, but we acknowledge that they may be pessimistic or optimistic. A problem related to this is that the vehicle controller does not account for the dynamics of the wheel controller. Instead it relies solely on bandwidth separation, but we have observed that if the bandwidth separation between the controllers is insufficent, then the vehicle will tend to oscillate around the path. To mitigate this, it might be possible to use the wheel reference models actively in the vehicle controller. This is especially applicable if we want to implement an optimization scheme in the vehicle controller, where we would replace the complex wheel dynamics with the linear wheel reference models.

When we tested the vehicle on sloped terrain, we found that the tracking performance was slightly worse, but we suspect that the root cause was not related to the slope directly. Instead, the slope of the terrain magnified problems that were already present in the system, particularly track loss during turns. We suspect this because when the vehicle was driving in a straight line, even perpendicularly to the slope, it did not struggle to keep on the path. But the slope had a big effect the speed error, which indicates that the speed controller was not aggressive enough. We believe that this is the primary reason why the tracking performance was worse with sloped terrain. On the downhill sections, the vehicle built up a more momentum which lead to greater overshoots during turns. We can demonstrate this by overlaying the baseline trajectory with the slope trajectory, which is done in Fig. 8.1. If we compare this with the speed error plot in Fig. 6.27b or the video of the case[1], we can see that the vehicle is traveling downhill and going faster compared to the baseline. As stated before, this could be counteracted by having a more aggressive speed controller and integral effect. In addition to this, we should consider expanding the model to include non-flat terrain. By modeling

[1] 1:15 in https://youtu.be/9PLcTj5GGQw

the additional resistance and using feedforward to account for it, we believe it is possible to significantly improve speed tracking when driving on terrain.

8.3 Future work

Throughout the report and in this chapter, we have mentioned many places were the system can be improved. To wrap up the thesis, we present a short-list of the improvements we believe are most important, particularly for bringing the autonomous features developed here to the AutoAgri vehicle.

System identification: We should perform system identification on the vehicle in order to build a data-driven model. This will allow us to build a more accurate simulation and tune the controllers and reference models.

Chatter reduction: If we are going to use SMC or similar control techniques to control the steering angle, we should look for ways to reduce chattering.

Motor drivers: In order to apply the control system to the AutoAgri vehicle, we need to translate the control system outputs to motor signals with a motor driver. If the motor driver does not allow torque control, then this might require a redesign of the wheel controllers.

State estimation: All the signals that are used in the control system need to be either measured or estimated in the real vehicle. This could be a big task, but assuming GPS is available, we can use a standard inertial navigation system to estimate position, velocity, acceleration, orientation, and angular rates. This can extended with measurements from the wheels to estimate wheel states and friction parameters for each wheel.

Mechanical limits: On the AutoAgri vehicle, the wheel steering angles are constrained, and this is a mechanical limitation that the wheel controller must maintain. We believe the reference optimization scheme in the wheel controllers can be trivially extended to deal with this by adding a large (functionally infinite) cost of violating the constraints.

Lemmas A

A.1 Finite time stability

Assume a system has a Lyapunov function V which satisfies

$$\dot{V} + \alpha V^{\lambda} \leq 0 \tag{A.1}$$

where $\alpha > 0$ and $0 < \lambda < 1$. This is a separable differential equation. Solving it we get

$$\frac{\mathrm{d}V}{\mathrm{d}t} \leq -\alpha V^{\lambda} \tag{A.2}$$

$$V^{-\lambda}\,\mathrm{d}V \leq -\alpha\,\mathrm{d}t \tag{A.3}$$

We integrate both sides from $t = 0$ to t, and get

$$\int_{V(0)}^{V(t)} V^{-\lambda}\,\mathrm{d}V \leq \int_{0}^{t} -\alpha\,\mathrm{d}\tau \tag{A.4}$$

$$\frac{1}{1-\lambda}\left(V(t)^{-\lambda+1} - V(0)^{-\lambda+1}\right) \leq -\alpha t \tag{A.5}$$

$$\tag{A.6}$$

Denote the reaching time by t_r which is where we guarantee that $t \geq t_r \Rightarrow V(t) = 0$. Inserting $t = t_r$, $V(t_r) = 0$ and $V(0) = V_0$ into the above expression gives

$$\frac{1}{1-\lambda}\left(V(t_r)^{-\lambda+1} - V_0^{-\lambda+1}\right) \leq -\alpha t_r \tag{A.7}$$

$$\frac{1}{1-\lambda}V_0^{-\lambda+1} \geq \alpha t_r \tag{A.8}$$

$$t_r \leq \frac{V_0^{-\lambda+1}}{\alpha(1-\lambda)} \tag{A.9}$$

Since $t_r < \infty$ for all V_0, $\alpha > 0$ and $0 < \lambda < 1$, this proves that the system will converge to $V(t) = 0$ within the finite reaching time t_r.

B Interface definitions

B.1 Custom definitions

B.1.1 vehicle_interface/msg/Guide

```
float64 drive_torque
float64 steer_torque
```

B.1.2 vehicle_interface/msg/Waypoints

```
geometry_msgs/Point[] points
```

B.1.3 vehicle_interface/msg/WheelCommand

```
float64 drive_torque
float64 steer_torque
```

B.1.4 vehicle_interface/msg/WheelLoad

```
float64 load
```

B.1.5 vehicle_interface/msg/WheelState

```
float64 steering_angle
float64 steering_angle_rate
float64 angular_velocity
```

B.1.6 vehicle_interface/msg/YawReference

```
# Subscriber can use source to filter for relevant messages
string source
float64 yaw
float64 yawrate
```

B.2 Standard library definitions

B.2.1 std_msgs/msg/Header

http://docs.ros.org/en/noetic/api/std_msgs/html/msg/Header.html

```
uint32 seq
time stamp
string frame_id
```

B.2.2 geometry_msgs/msg/Point

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Point.html

```
float64 x
float64 y
float64 z
```

B.2.3 geometry_msgs/msg/Pose

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Pose.html

```
Point position
Quaternion orientation
```

B.2.4 geometry_msgs/msg/PoseStamped

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseStamped.html

```
std_msgs/Header header
Pose pose
```

B.2.5 geometry_msgs/msg/Twist

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html

```
Vector3 linear
Vector3 angular
```

B.2.6 geometry_msgs/msg/TwistStamped

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/TwistStamped.html

```
std_msgs/Header header
Twist twist
```

B.2.7 geometry_msgs/msg/Quaternion

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Quaternion.html

```
float64 x
float64 y
float64 z
float64 w
```

B.2.8 geometry_msgs/msg/Vector3

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Vector3.html

```
float64 x
float64 y
float64 z
```

B.2.9 geometry_msgs/msg/Vector3Stamped

http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Vector3Stamped.html

```
std_msgs/Header header
Vector3 vector
```

B.2.10 sensor_msgs/msg/Joy

http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Joy.html

```
std_msgs/Header header
float32[] axes
int32[] buttons
```

C Circular path smoothing algorithm

This appendix presents an algorithm for constructing a circularly smoothed path between three waypoints, which can be extended to construct a path between any number of waypoints. It is an approximating path, meaning it does not pass through the intermediary waypoint, but it does end on the final waypoint. This is related to Dubin's path [11], but Dubin's path also handles orientations and requires the the path passes through the intermediary waypoints.

Consider three succesive waypoints $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$. We want to create a circularly smoothed path from $\mathbf{p}_0$ to $\mathbf{p}_2$ via $\mathbf{p}_1$ as illustrated in Fig. C.1. We want to smooth out the path using a circle segment with radius r, which is given as a parameter.

First compute the normalized velocities and turn direction ρ

$$\mathbf{v}_0 = \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|} \tag{C.1}$$

$$\mathbf{v}_1 = \frac{\mathbf{p}_2 - \mathbf{p}_1}{\|\mathbf{p}_2 - \mathbf{p}_1\|} \tag{C.2}$$

$$\rho = \operatorname{sgn}\left((\mathbf{v}_0 \times \mathbf{v}_1)_z\right) \tag{C.3}$$

The path-normals that point into the turns at $\mathbf{p}_0$ and $\mathbf{p}_2$ are given by

$$\mathbf{n}_0 = \rho \begin{bmatrix} -v_{0,y} \\ v_{0,x} \end{bmatrix} \tag{C.4}$$

$$\mathbf{n}_1 = \rho \begin{bmatrix} -v_{1,y} \\ v_{1,x} \end{bmatrix} \tag{C.5}$$

The center of the circle segment is at the intersection of the straight line segments given by

$$l_0(t_0) = \mathbf{p}_0 + r\mathbf{n}_0 + t_0(\mathbf{p}_1 - \mathbf{p}_0) \tag{C.6}$$

$$l_1(t_1) = \mathbf{p}_2 + r\mathbf{n}_1 + t_1(\mathbf{p}_2 - \mathbf{p}_1) \tag{C.7}$$

The intersection is then given by $l_0(t_0) = l_1(t_1)$, which we can write as

$$\mathbf{p}_0 + r\mathbf{n}_0 + t_0(\mathbf{p}_1 - \mathbf{p}_0) = \mathbf{p}_2 + r\mathbf{n}_1 + t_1(\mathbf{p}_2 - \mathbf{p}_1) \tag{C.8}$$

$$\Leftrightarrow \begin{bmatrix} \mathbf{p}_1 - \mathbf{p}_0 & \mathbf{p}_1 - \mathbf{p}_2 \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} = \mathbf{p}_2 + r\mathbf{n}_1 - \mathbf{p}_0 - r\mathbf{n}_0 \tag{C.9}$$

This system is invertible as long as the points are not colinear, so we can solve for t_0 and t_1. The circle center is the given by

$$\mathbf{p}_c = \mathbf{p}_0 + r\mathbf{n}_0 + t_0(\mathbf{p}_1 - \mathbf{p}_0) \tag{C.10}$$

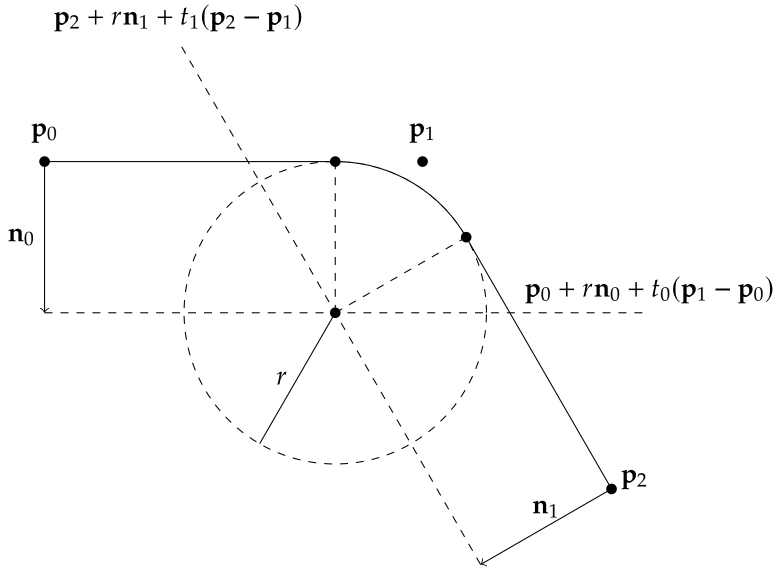

$\mathbf{p}_2 + r\mathbf{n}_1 + t_1(\mathbf{p}_2 - \mathbf{p}_1)$
$\mathbf{p}_0$
$\mathbf{p}_1$
$\mathbf{n}_0$
$\mathbf{p}_0 + r\mathbf{n}_0 + t_0(\mathbf{p}_1 - \mathbf{p}_0)$
r
$\mathbf{p}_2$
$\mathbf{n}_1$

Figure C.1: Circular path smoothing illustration.

Recall that a circle segment is parametrized as

$$\mathbf{P}(t_p) = \mathbf{p}_c + \begin{bmatrix} r\cos\big(\alpha_0 + t_p(\alpha_1 - \alpha_0)\big) \\ r\sin\big(\alpha_0 + t_p(\alpha_1 - \alpha_0)\big) \end{bmatrix} \tag{C.11}$$

We have now established $\mathbf{p}_c$, so we need to determine α_0 and α_1. The tangent points where the path will turn into and out of the circle segment are given by

$$\mathbf{p}_{t0} = \mathbf{p}_0 + t_0(\mathbf{p}_1 - \mathbf{p}_0) \tag{C.12}$$
$$\mathbf{p}_{t1} = \mathbf{p}_2 + t_1(\mathbf{p}_2 - \mathbf{p}_1) \tag{C.13}$$

The vector from the circle center to the tangent points are $-\mathbf{n}_0$ and $-\mathbf{n}_1$. The angles are then given by

$$\alpha_0 = -\operatorname{atan2}(n_{0,y}, n_{0,x}) \tag{C.14}$$
$$\alpha_1 = -\operatorname{atan2}(n_{1,y}, n_{1,x}) \tag{C.15}$$

To avoid unwanted behaviour, we recompute α_1 to ensure that it is the closest it can be to α_0.

$$\alpha_1 := \alpha_0 + \operatorname{ssa}(\alpha_1 - \alpha_0) \tag{C.16}$$

References

[1] Autoagri AS. *Autoagri website*. 2020. URL: https://autoagri.no/ (visited on Dec. 11, 2020).

[2] Randal W. Beard and Timothy W. McLain. *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012. ISBN: 9780691149219. URL: http://www.jstor.org/stable/j.ctt7sbc4.

[3] Peter Biber et al. "Navigation System of the Autonomous Agricultural Robot "BoniRob"". In: ().

[4] Meghan Brown. *Smart Farming - Automated and Connected Agriculture*. 2018. URL: https://www.engineering.com/story/smart-farming-automated-and-connected-agriculture (visited on May 28, 2021).

[5] Manfred Burckhardt and Jörnsen Reimpell. *Fahrwerktechnik: Radschlupf-Regelsysteme*. Vogel, 1993. ISBN: 9783802304774.

[6] Rendell Cale. "Modeling of a Four-wheel Agricultural Robot". Project-thesis for Cybernetics and Robotics. 2020.

[7] X. Chen et al. "Path Tracking Control of Four-wheel Independent Steering Electric Vehicles Based on Optimal Control". In: (2020), pp. 5436–5442. DOI: 10.23919/CCC50068.2020.9189047.

[8] SciPy community. *numpy.hanning*. URL: https://numpy.org/doc/stable/reference/generated/numpy.hanning.html (visited on June 14, 2020).

[9] *Curvature*. 2021. URL: https://en.wikipedia.org/wiki/Curvature (visited on June 21, 2020).

[10] P.R. Dahl. "A Solid Friction Model". In: (1968).

[11] L. E. Dubins. "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents". In: *American Journal of Mathematics* 79.3 (1957), pp. 497–516. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2372560.

[12] Olav Egeland and Tommy Gravdahl. *Modeling and Simulation for Automatic Control*. Marine Cybernetics AS, 2003. ISBN: 82-92356-01-0.

[13] Thor Inge Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011. ISBN: 978-1-1199-9149-6.

[14] T. Fraichard and A. Scheuer. "From Reeds and Shepp's to continuous-curvature paths". In: *IEEE Transactions on Robotics* 20.6 (2004), pp. 1025–1035. DOI: 10.1109/TRO.2004.833789.

[15] Friction and Friction Coefficients. *Calculate distance, bearing and more between Latitude/Longitude points*. URL: https://www.movable-type.co.uk/scripts/latlong.html (visited on June 14, 2020).

[16] Petros Ioannou and Jing Sun. *Robust Adaptive Control*. Jan. 1995.

[17] Reza N. Jazar. *Vehicle Dynamics: Theory and Application*. Springer, 2008. ISBN: 978-0-387-74244-1.

[18] Hassan K. Khalil. *Nonlinear Systems*. Pearson, 2015. ISBN: 0-13-067389-7.

[19] Uwe Kiencke and Lars Nielsen. *Automotive Control Systems*. Springer, 2005. ISBN: 3-540-23139-0.

[20] Nathan Koenig and Andrew Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, Sept. 2004, pp. 2149–2154.

[21] Linly Ku. *How Automation is Transforming the Farming Industry*. 2019. URL: `https://www.plugandplaytechcenter.com/resources/how-automation-transforming-farming-industry/` (visited on May 28, 2021).

[22] Anastasios M. Lekkas. "Guidance and Path-Planning Systems for Autonomous Vehicles". PhD thesis. Trondheim: Norwegian University of Science and Technology, Apr. 2014.

[23] Filipe Marques, Paulo Flores, and Hamid M. Lankarani. "Study of Friction Force Model Parameters in Multibody Dynamics". In: *The 4th Joint International Conference on Multibody System Dynamics*. 2016.

[24] ODE. *ODE Manual*. 2019. URL: `http://ode.org/wiki/index.php?title=Manual` (visited on June 9, 2020).

[25] Hans B. Pacejka. "Chapter 7 - Single-Contact-Point Transient Tire Models". In: *Tire and Vehicle Dynamics (Third Edition)*. Ed. by Hans B. Pacejka. Third Edition. Oxford: Butterworth-Heinemann, 2012, pp. 329–354. ISBN: 978-0-08-097016-5. DOI: `https://doi.org/10.1016/B978-0-08-097016-5.00007-3`. URL: `https://www.sciencedirect.com/science/article/pii/B9780080970165000073`.

[26] "Path Planning and Guidance for Marine Surface Vessels". MA thesis. Trondheim: Norwegian University of Science and Technology, Apr. 2013, p. 237.

[27] J. Peters. "Changing Variables". In: *IEEE Computer Graphics and Applications* 32.3 (2012), pp. 88–93. DOI: `10.1109/MCG.2012.51`.

[28] Georg Rill. "Wheel Dynamics". In: *DINAME* (2007).

[29] Open Robotics. *Customer Stories*. URL: `https://www.openrobotics.org/customer-stories` (visited on June 19, 2020).

[30] *ROS 2 Documentation*. 2020. URL: `https://docs.ros.org/en/foxy/index.html#` (visited on May 2, 2021).

[31] Y.D. Setiawan, T.H. Nguyen, and P.S. et al. Pratama. "Path tracking controller design of four wheel independent steering automatic guided vehicle". In: *Int. J. Control Autom. Syst.* 14 (2016).

[32] Yuhanes Dedy Setiawan et al. “Control System Desifn of Four Wheeled Independent Steering Automatic Guided Vehicles Based On Backstepping Control Method”. In: Nov. 2013.

[33] R. Sinnott. “Virtues of the Haversine”. In: 1984.

[34] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2006. ISBN: 978-0-471-64990-8.

[35] Eric W. Weisstein. *Fermat's Spiral*. URL: https://mathworld.wolfram.com/FermatsSpiral.html (visited on May 6, 2021).

[36] Eric W. Weisstein. *Halley's Method*. URL: https://mathworld.wolfram.com/HalleysMethod.html (visited on Mar. 17, 2021).

[37] C. Canudas de Wit et al. “A New Model for Control of Systems with Friction”. In: (1995).

[38] Yunxiang Ye, Long He, and Qin Zhang. “Steering Control Strategies for a Four-Wheel-Independent-Steering Bin Managing Robot”. In: *IFAC-PapersOnLine* 49.16 (2016). 5th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2016, pp. 39–44. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.10.008. URL: http://www.sciencedirect.com/science/article/pii/S2405896316315695.

[39] Seongjin Yim. “Comparison among Active Front, Front Independent, 4-Wheel and 4-Wheel Independent Steering Systems for Vehicle Stability Control”. In: *Electronics* 9.5 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9050798. URL: https://www.mdpi.com/2079-9292/9/5/798.