Bachelor's thesis

NTNU
Norwegian University of Science and Technology

Eskil Bakken
Eirik Norvoll Bjørnevik
Jørgen Vesterheim Knutsen
Vegard Ohren

Lone Wolf ATV: Dynamic Model and Model Predictive Control

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Irja Gravdahl
May 2024

NTNU
Norwegian University of Science and Technology

KONGSBERG

Eskil Bakken
Eirik Norvoll Bjørnevik
Jørgen Vesterheim Knutsen
Vegard Ohren

Lone Wolf ATV: Dynamic Model and Model Predictive Control

KONGSBERG

Bachelor's thesis in Electrical Engineering - Automation and Robotics
Supervisor: Irja Gravdahl
May 2024

Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Abstract

This bachelor's thesis report presents the final project done by four Electrical Engineering students from the Norwegian University of Science and Technology, specializing in Automation and Robotics.

The thesis involves the design and development of a dynamic model and a Model Predictive Controller (MPC) for an autonomous all-terrain vehicle (ATV). The report details the interdisciplinary approach, integrating principles from mechanical dynamics, control theory, and computer simulations to address the challenges of autonomous navigation and control.

To achieve this, the team developed a simulator with a dynamic model that simulates the ATV's physical behaviors in the XY-plane. This model incorporates advanced tire modeling and powertrain dynamics to accurately reflect the interaction between the vehicle and the terrain.

A significant component of the project involved implementing an MPC framework designed to optimize the vehicle's trajectory and control inputs in real-time. The MPC utilizes a simplified dynamic model to predict the vehicle's future states and calculate optimal control actions that minimize a predefined cost function, which includes terms for tracking error and control effort.

This thesis not only demonstrates the feasibility of using advanced control strategies for autonomous ATV navigation but also lays a foundation for future work to explore more complex scenarios, including obstacle avoidance and adaptive path planning. The project exemplifies the potential of combining theoretical knowledge with practical application in solving real-world engineering challenges.

Sammendrag

Denne rapporten beskriver arbeidet som er utført i forbindelse med bacheloroppgaven til fire elektroingeniørstudenter ved Norges teknisk-naturvitenskapelige universitet (NTNU).

Oppgaven går i hovedsak ut på utviklingen av en dynamisk modell av en firhjuling, som deretter er implementert i en simulator. Det er i tillegg utviklet en modellprediktiv kontroller (MPC), som er testet i simulatoren. Rapporten beskriver de tverrfaglige løsningene og valgene som er tatt i løpet av prosjektarbeidet.

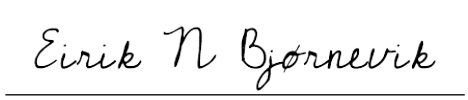
Den dynamiske modellen som er implementert simulerer kjøreegenskapene til en firhjuling i XY-planet. Modellen er utviklet ved bruk av fysiske lover, samt avanserte dekk- og drivverksmodeller.

En viktig del av prosjektet var å implementere en MPC for å optimere firhjulingens inputs for følging av forhåndsdefinerte baner. MPC-en som er implementert bruker en forenklet dynamisk modell til å forutsi firhjulingens fremtidige tilstander. Disse brukes deretter for å minimere en predefinert kostfunksjon.

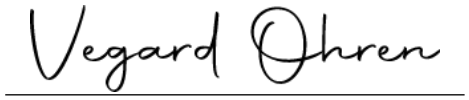
Bacheloroppgaven demonstrerer hvordan avanserte kontrollalgoritmer kan benyttes for autonom navigering av en firhjuling. Den legger også grunnlaget for videre arbeid, for eksempel kontinuerlig baneplanlegging og unngåelse av hindringer. Prosjektet kombinerer teoretisk kunnskap med praktisk, ingeniørfaglig anvendelse for å løse en reell teknisk oppgave.

Preface

This thesis constitutes the capstone project for our Bachelor of Science degree in Electrical Engineering, with a specialization in Automation and Robotics, at the Norwegian University of Science and Technology (NTNU). The project has been immensely interesting and challenging, providing us with invaluable experience and preparation for future challenges in control engineering.

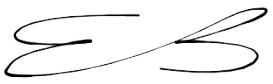


Eirik Norvoll Bjørnevik



Vegard Ohren

Jørgen Vesterheim Knutsen



Eskil Bakken

Trondheim

16th of May 2024

Acknowledgements

We would like to express our sincere gratitude to all those who have supported us throughout our thesis.

Profound thanks are due to our advisor, Irja Gravdahl, for her exceptional dedication and invaluable insights throughout our thesis process. Her constant availability and constructive feedback have been crucial to our success.

Special gratitude is extended to Roger Werner Laug for entrusting us with this project and equipping us with the vital insights and resources needed to excel. Additionally, heartfelt thanks go to Chris André Brombach and Kristian Sandaa for their technical assistance, feedback, and ongoing support throughout the project.

Our appreciation also extends to Professor Damiano Varagnolo, Associate Professor Christian Fredrik Sætre, and Associate Professor Gilbert Joseph Bergna Diaz for their invaluable technical insights.

Lastly, we acknowledge and thank Eline Marie Håve and Glenn Risvold Varhaug for their invaluable assistance during the project. Eline provided guidance in dynamic modeling for autonomous vehicles, significantly enhancing our understanding and approach to the task. Glenn offered valuable help in navigating and understanding the previous work done on this project.

Table of Contents

List of Figures

List of Tables

This page has been left blank intentionally.

Chapter 1

Introduction

This introductory chapter aims to orient the reader about the central theme, objectives, and scope of the research. Moreover, the structure of the thesis is outlined, detailing the focus of each chapter and how they build upon each other to achieve the objectives of the thesis.

1.1 Background on project Lone Wolf

"Project Lone Wolf is a multi-disciplinary project within KONGSBERG's business area, Kongsberg Defence & Aerospace. The project is dedicated to designing and constructing an autonomous all-terrain vehicle (ATV) capable of obstacle avoidance." [16]

Figure 1.1: The project group and the physical Lone Wolf ATV

The Lone Wolf project was initiated in 2019 to bring together students from various disciplines to collaborate on a large-scale project. It primarily serves as a summer project for interns but has also been the subject of academic theses at the Norwegian University of Science and Technology and the University of South-Eastern Norway in recent years.

The physical ATV has been modified and constructed to enable remote steering, gear-shifting, acceleration, and braking control using a remote controller or a connected computer. Several challenges must be overcome to achieve the final goal of an autonomous ATV. This thesis builds upon the previous work done on the Lone Wolf project and aims to contribute to the final goal of the project.

1.2 Project assignment

The original project assignment proposed by KDA, found in Appenix A, describes the overarching goal of fully autonomous operation of the Lone Wolf ATV and the requirements for the project. The key elements of the project assignment include:

- Identify the dynamics of the ATV, and develop a physics model that describe the identified dynamics.
- Implement the model in a ROS2-compatible simulator.
- Implement a control algorithm based on the physics model, as well as available sensor data, for example, MPC.
- Integrate the control algorithm in the current software architecture.

Some additional tasks were proposed, including looking at the necessity of a closed-loop control system for the steering angle, comparing the current control algorithm with the new one, and exploring the possibility of sensor fusion for better state estimation.

1.2.1 Thesis statement

Based on the project assignment, the thesis statement narrows the assignment down to the essentials, which constitutes the overarching goal of the thesis:

"Development of a dynamic model of the Lone Wolf ATV and integration of a complementary control algorithm"

1.3 Thesis objectives

The key elements of the project assignment must be specified and re-phrased to specific goals so the goal achievement of the project can easily be identified. This will also help identify the prerequisites of the project.

Based on the project assignment and thesis statement, these four main objectives were identified to guide the work throughout our thesis:

1. **Identify and model the dynamics of the ATV:** Develop an accurate, nonlinear, and dynamic model that mimics the physical Lone Wolf ATV dynamics.
2. **Implement the dynamic model in a simulator:** The model identified in the first objective must be implemented in a ROS2-compatible simulator.
3. **Develop a Model Predictive Controller:** Develop an MPC to be tested in the simulator. The controller should be based on measurements from sensor data, which resembles the information that can be gathered from the physical Lone Wolf, such as GPS- and IMU data.
4. **Implement the control algorithm into the Lone Wolf system:** The developed control algorithm has to be implemented into the current system- and software architecture of the current Lone Wolf project so that it can easily be used in future projects.

These were the four initial objectives outlined in the preliminary project. However, due to unforeseen circumstances at the Kongsberg test facility[1], conducting tests and collecting data from the physical Lone Wolf ATV was not possible. As a result, the first objective *"Identify and model the dynamics of the ATV"* had to be revised. This change was done in close consultation with the client and supervisor, leading to a redefined objective that replaced the original. The redefined objective states:

1. **Identify and modelling the dynamics of a four-wheel vehicle:** Exploration, analysis, and modeling of a generic four-wheel vehicle's dynamics. Structure the model to allow adjustable parameters, enabling future modifications to closely simulate the physical Lone Wolf ATV dynamics.

[1] *"Lagerbygg har rast sammen i Kongsberg" - VG* [40]

1.3.1 Prerequisites

As stated in the thesis objectives, the focus of this thesis is to identify, simulate, and control a generic four-wheel vehicle. However, the development of autonomous vehicles requires overcoming various technical challenges. Because of this, these prerequisites were defined:

- **Path/trajectory planning:** The path/trajectory the ATV should follow is assumed to be pre-generated. In reality, a path-/trajectory-planning algorithm would need to be implemented to feed a reference to the controller.
- **Signal processing:** It is assumed that the signals delivered to the MPC are noiseless. In reality, sensors and signals will experience noise, which can influence the controller's performance.

1.4 Project overview

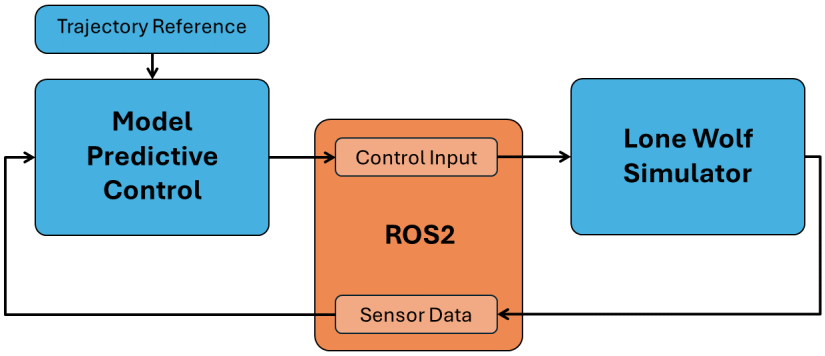


Figure 1.2: Project flowchart

The first part of the project, seen as the blue box labeled *Lone Wolf Simulator* in Figure 1.2, focuses on the detailed development of the dynamic model for the Lone Wolf ATV. This part includes research and analysis of vehicle characteristics to accurately capture the vehicle's nonlinear dynamics. Key activities involve setting up equations of motion and modeling the mechanics of the vehicle. The resulting model is then implemented into the ROS2-compatible simulation environment Simulink.

The second part of the project, seen as the blue box labeled *Model Predictive Control* in Figure 1.2, focuses on developing and integrating the Model Predictive Controller (MPC). Instead of utilizing the advanced dynamic model from the simulator, the MPC is based on an external, simplified model. This approach will facilitate a clearer understanding of the controller parameters and keep computational cost at a reasonable level. The MPC was tested using the simulator developed in the first part of the thesis. This testing phase was crucial for fine-tuning the controller to achieve desired responses.

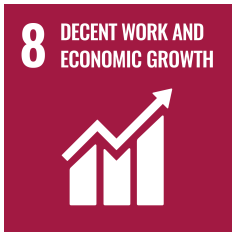
The interaction between the Lone Wolf Simulator and the MPC mimics the ROS2 framework of the actual Lone Wolf system, ensuring that the controller integration is replicable on the real Lone Wolf. This is visualized in the orange box labeled *ROS2* in Figure 1.2.

The flowchart in Figure 1.2 is repeated throughout the beginning of the relevant chapters in this thesis report. In addition, figures and flowcharts are provided where necessary to help give a deeper understanding of the content provided throughout the text in the report.

1.5 Report structure

- **Chapter 2: Robot Operating System** briefly introduces the ROS framework and how it is utilized in this thesis to resemble the architecture of the real-world Lone Wolf.
- **Chapter 3: Dynamic Modeling - Theory** introduces different techniques used in nonlinear modeling and explains the physics, models, and approaches taken in modeling a generic four-wheel vehicle.
- **Chapter 4: Dynamic Modeling - Implementation** describes how the theory from Chapter 3 was used to develop a simulator in Simulink.
- **Chapter 5: MPC - Theory** contains the theory regarding Model Predictive Control, discretization, and optimization.
- **Chapter 6: MPC - Implementation** explains how the MPC was developed, tuned, and implemented in the system.
- **Chapter 7: Results** shows the simulator's and the MPC's performances under various conditions.
- **Chapter 8: Discussion** discusses the methods and results presented throughout the thesis.
- **Chapter 9: Conclusion** summarizes the work done throughout the project.

1.6 Sustainable Development Goals



The Sustainable Development Goals (SDGs) in focus during this thesis are number 8, 9, and 11 [25]. They are the following:

- **SDG 8, Decent Work and Economic Growth:** By automating transport in challenging terrains, autonomous vehicles can increase productivity and support economic growth while ensuring safety in work environments.
- **SDG 9, Industry, Innovation, and Infrastructure:** The development of autonomous vehicles enhances industrial infrastructure and innovation, particularly in agriculture and emergency response, fostering sustainable industrialization.
- **SDG 11, Sustainable Cities and Communities:** Autonomous vehicles improve urban mobility, enhance emergency services, and reduce emissions, supporting the development of inclusive, sustainable urban environments.

The integration of autonomous ATVs aligns with SDGs 8, 9, and 11, promoting innovation, sustainability, and economic growth within industrial and urban frameworks. This project demonstrates the potential of such technologies to advance societal and environmental goals for the future.

1.7 Reader requirements

Before reading this thesis, it is recommended that the reader possesses basic knowledge of physics, calculus, linear algebra, and differential equations. Although not required, the reader would strongly benefit from a basic understanding of control engineering and dynamic systems. All concepts deemed non-trivial to someone with some background in engineering are explained when they arise.

Chapter 2

Robot Operating System

As outlined in Figure 2.1, this chapter is aimed at explaining Robot Operating System (ROS), and the communication architecture for the project. This aims to give the reader an overview of how the full system works together. This chapter focuses on the fourth objective, which states:

4. **Implement the control algorithm into the Lone Wolf system:** The developed control algorithm has to be implemented into the current system- and software architecture of the current Lone Wolf project so that it can easily be used in future projects.

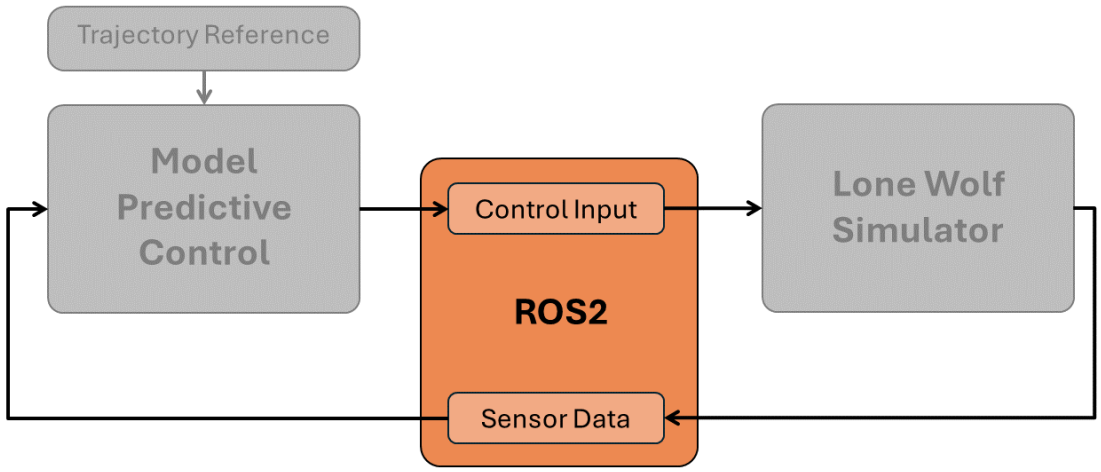


Figure 2.1: Project flowchart: Robot Operating System

2.1 ROS - Theory

Robot Operating System (ROS) is an open-source set of libraries used for robotics applications [34]. Despite its name, it is not an actual operating system, but can rather be described as a framework that enables communication between different software [13].

Robotics is an interdisciplinary field requiring expertise from various fields of study. In addition, robots come in numerous different configurations. Before ROS was created, vast amounts of time were spent configuring communication systems for each new robot [13]. This was an advanced process that made it difficult to contribute to robotics research without specific how-to knowledge. The problem was recognised by the creators of ROS as an impeding factor for progress in development and research [47]. As a result, Robot Operating System was created to make a standard framework for robotics communications. This lowered the barriers to entry in robotics, allowing more time to be spent developing new technology and welcoming people from other fields to contribute to robotics.[34]

ROS allows for the interconnection of various software. This is greatly useful in the Lone Wolf project, because of all the different software that needs to run together smoothly.

2.2 Utilized ROS infrastructure

The ROS framework has a lot of different functionality, but only a few are used in this project. This section describes the features that are utilized in this project.

Node

A node is a process that performs a computation. A robot system often consists of a great number of nodes, each one controlling a different component or process. This can be anything from a microcontroller reading sensor data to pure software performing computations. There can be any number of nodes in a system, only limited by the system running them [31]. An architecture of nodes greatly reduces the complexity of large systems, as there is no need for a direct pipeline for information. In addition, it increases the fault tolerance, as a crash will often be limited to a specific node [33].

Topics

Topics are named communication buses used by nodes to exchange information [35]. These work on a publish and subscribe basis. Any node can publish, or send data, to a given topic. All nodes subscribing to this topic have access to the data sent by the publisher nodes. The data published to a topic must be of a ROS datatype, anything from a simple integer to complex datatypes containing geometric information including covariances and timestamps. [35]

The code in Listion 2.1 demonstrates how a node is initialized as a publisher and a subscriber. The publisher is created by using the inherited method `self.create_publisher()`. The method requires the ROS2 datatype, the specific topic it should publish to, and a message buffer size. Creating a subscriber is the same process, except it requires a callback function, which determines what should happen when a message is published to the topic.

```
1 class ROS2_Node(Node):
2     def __init__(self):
3         self.publisher = self.create_publisher(Int64, '/topic', 10)
4         self.subscriber = self.create_subscription(Float32, '/topic',
5                                             self.callback_function, 10)
```

Listing 2.1: Python code for creating a ROS2 publisher

Messages

ROS messages are a way for different nodes to communicate with each other. They are structured data formats, like position, speed, or simply integers, that nodes in the ROS network send and receive to coordinate actions and share information. Each message type defines a specific set of data fields that are relevant to a particular topic [32].

Figure 2.2 illustrates how messages from Nodes can be distributed via a ROS topic.

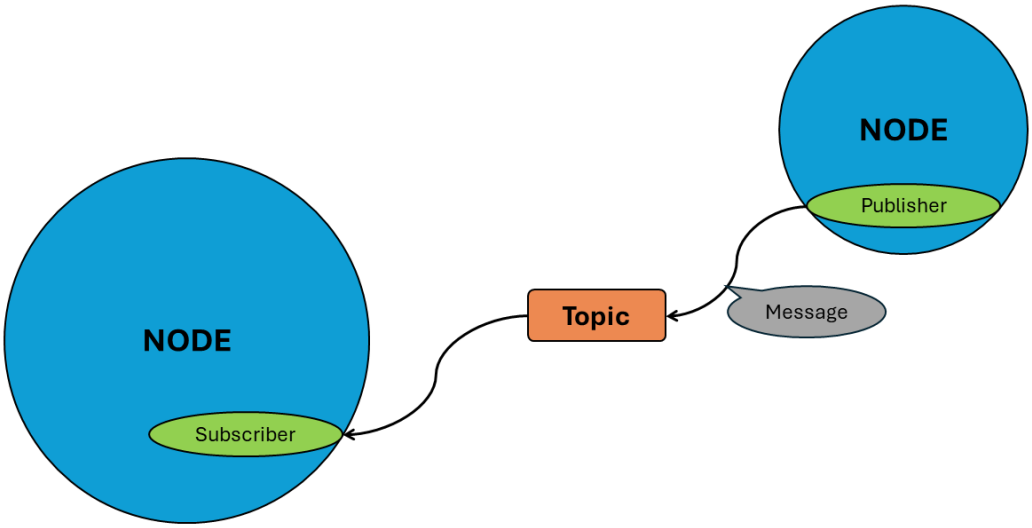


Figure 2.2: ROS: Publishers and subscribers [36]

2.3 ROS2 implementation

The implementation uses the improved ROS2 framework, the second major iteration of ROS. ROS2 is a decentralized system, which greatly improves system robustness, as there is no single point of failure. In the implementation, nodes communicate directly with each other by publishing and subscribing to topics.

Figure 2.3 illustrates the communication architecture created to enable the connection between the MPC and the simulator.

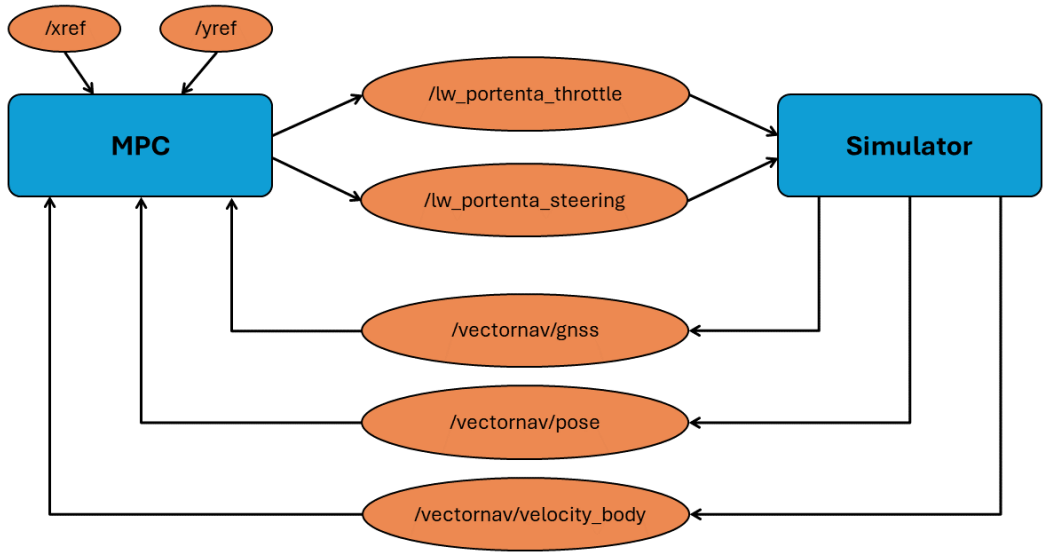


Figure 2.3: ROS: Project architecture

The code in Listing 2.2 demonstrates how the publishers and subscribers are created in the ROS2 implementation. Lines 4 to 5 show the process of initializing a node as a publisher to a specific ROS2 topic. As shown in the code, the publisher `self.u1_publish` uses the ROS2 data type Int64 and publishes to the `/lw_portenta_throttle` topic. The same is done for the `/lw_portenta_steering`. The figure also demonstrates how a specific subscriber to a topic is initialized.

Line 7 to 9 demonstrates how the subscribers are created. The subscriber `self.xy_sub` uses the ROS2 datatype `NavSatFix`, and subscribes to the `/vectornav/gnss` topic. Additionally, it uses the `self.update_states_xy` method to update the X- and Y positions when a message has been published to the topic. The same is done for the `/vectornav/gnss` and `/vectornav/velocity_body` topics. The`/vectornav` topics con-

tain IMU data from the Vectornav fitted on Lone Wolf. The MPC also requires access to a reference. As illustrated in figure 2.3, this is achieved by subscribing to the reference topics, `/xref` and `/yref`.

```
class mpc_ros_controller(Node):
    def __init__(self):
        ...  # Code omitted for brevity
        self.u1_publish = self.create_publisher(Int64,
                          '/lw_portenta_throttle', 10)
        ...  # Code omitted for brevity
        self.xy_sub = self.create_subscription(
                          NavSatFix, '/vectornav/gnss',
                          self.update_states_xy, 10)
        ...  # Code omitted for brevity
```

Listing 2.2: Python code showing snippets the ROS2 topics utilized

The Simulator subscribes to the control input topics. It uses the control inputs and then publishes the current state of the simulated Lone Wolf to the `/vectornav` topics. Additionally, it subscribes to the reference topics for plotting purposes.

Recall that the 4th objective, re-stated in this chapter's introduction, concerns implementing the control algorithm into the existing software architecture. The interface between the MPC and the simulator, as illustrated in figure 2.3, is by design the exact interface the physical Lone Wolf uses. This approach was taken to simplify the implementation process on the real Lone Wolf.

Chapter 3

Dynamic Model - Theory

This chapter briefly introduces non-linear modeling and explains the physics and models used to develop the dynamic model of a generic four-wheel vehicle. The theory explained throughout this chapter is used in the Lone Wolf Simulator, as illustrated in Figure 3.1. The chapter focuses on the first objective, which states:

1. **Identify and model the dynamics of a four-wheel vehicle:** Exploration, analysis, and modeling of a generic four-wheel vehicle's dynamics. Structure the model to allow adjustable parameters, enabling future modifications to closely simulate the physical Lone Wolf ATV dynamics.

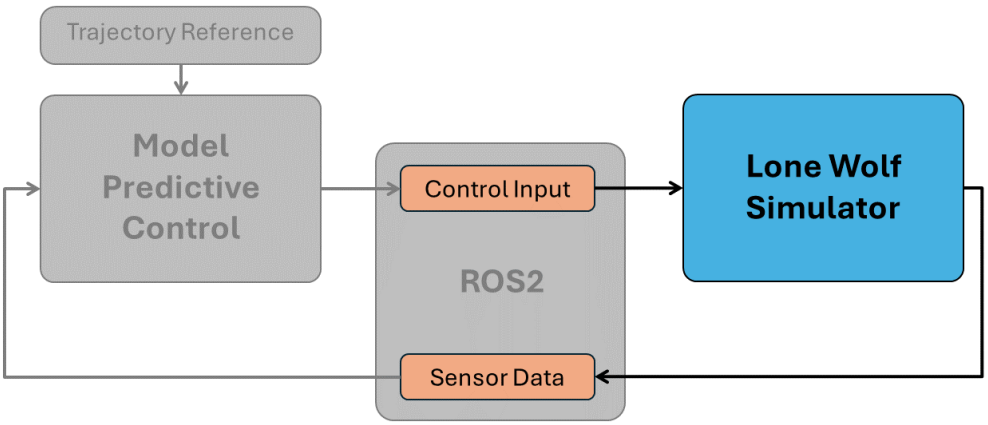


Figure 3.1: Project flowchart: Simulink

3.1 Nonlinear modeling

Nonlinear modeling involves the process of developing mathematical models that accurately represent the behavior of non-linear systems based on observed data [46]. Unlike linear systems, where responses are directly proportional to the inputs, non-linear systems exhibit behaviors such as saturation, hysteresis, or dynamic responses that cannot be captured by simple linear equations [46]. The goal of non-linear system identification is to construct a model that can predict future system outputs or understand system dynamics, which is crucial in fields such as control engineering and other industries. Nonlinear modeling typically involves selecting an appropriate modeling method, estimating model parameters, and validating the model against real-world data to ensure it performs as expected. Three different approaches to non-linear modeling are described in the following subsections.

White box modeling

White box models are completely transparent, with all internal workings and parameters explicitly defined [50]. These models are typically based on first principles, such as Newton's laws, known relationships, or empirical data of the system being modeled. Examples include physical and chemical process models based on differential equations. White box models have high interpretability and the ability to incorporate system knowledge directly into the model in question. However, these models can be limited by the accuracy of the system knowledge and can be strongly affected by assumptions and uncertainties. [50]

Black box modeling

Black box modeling refers to models where the internal structure and the relationships between input and output are not explicitly defined or understood [22]. These models focus solely on input-output mapping without considering the underlying process mechanisms. Examples include neural networks and deep learning models [22]. The advantages of black box modeling include high effectiveness in capturing complex and high-dimensional relationships and working well in situations where the underlying processes are hard to understand. However, as illustrated in Figure 3.2, black box models lack interpretability, which can be critical in fields requiring insight into decision processes. More information

about black box modeling and machine learning within system identification can be found in [22].

Grey box modeling

Grey box modeling strikes a balance between black and white box models, as seen in Figure 3.2 [48]. These models use partial knowledge about the system processes while also employing data-driven approaches to model unknown components or parameters [43]. Grey-boxed models balance flexibility and interpretability. They are useful in situations where some system processes or parameters are known, but others are too complex or unknown. [48]

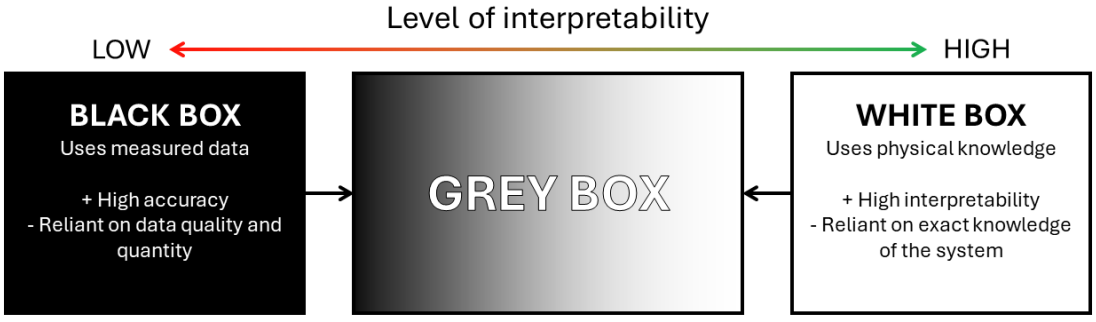


Figure 3.2: Nonlinear modeling methods

3.2 Model overview

All three modeling methods presented in Section 3.1 were considered for modeling the vehicle. Unfortunately, data collection was impossible due to the unforeseen circumstances mentioned in Section 1.3. As a result, black box and grey box methods were ruled out, leading the project group to base the model on white box modeling using first principles and assumptions. The rest of this chapter presents the approaches taken during the modeling of the vehicle and aims to give the reader a clear understanding of the model that is later implemented in the simulator.

Different mechanics and parts of the system are modeled separately before being tied together. Figure 3.3 shows how the different parts of the vehicle model are connected. Each part is explained in their own subsections throughout this chapter. The model takes in the control inputs, throttle and the steering angle δ via the ROS2 interface. The throttle

input is then used in the engine map to calculate the torque produced in the engine. This torque then passes through the drivetrain, which distributes the output torque to the front and rear wheels. In the front and rear wheel blocks, the tire forces, F_{xf}, F_{yf}, F_{xr} and F_{yr} are calculated individually using the Pacejka tire model. The front wheel takes in the control input δ to calculate the front wheel forces and transform them to the vehicle reference frame. The tire forces are then used in the dynamic equations of the vehicle body before the output signal data is transmitted via ROS2.

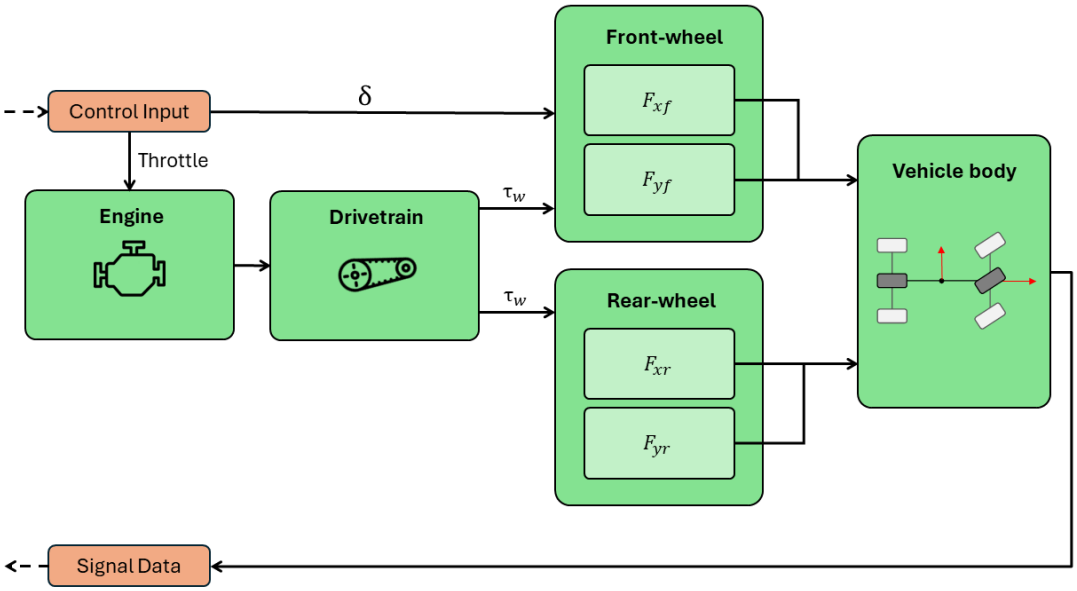


Figure 3.3: Dynamic model overview

At relatively low speeds, a vehicle model can be derived kinematically, as explained for the simplified model used for the MPC in Section 6.3. At higher speeds, the vehicle will experience slipping forces on the tires, leading to what is commonly known as drifting. Due to this, a more complex dynamic model, including these tire forces, has been implemented.

Terms and definitions

All terms and definitions used in modeling the vehicle are done in appliance with the "Road vehicles - Vehicle dynamics and road-holding ability"-standard provided by the International Organization for Standardization (ISO) [14]. The variables and parameters mentioned in this section are explained in Tables 3.2 and 3.3 at the end of the section. They are also mentioned in the text where relevant. When a variable or parameter is denoted by i, it represents either $i = f$ for the front or $i = r$ for the rear.

Degrees of freedom

The dynamic model described in this section has 3 degrees of freedom (DOF): movement in the longitudinal direction, x, movement in the lateral direction, y, and rotation about the vehicle's z-axis, Ψ. While more complex models like the 14 DOF [38] model were investigated, the simplicity of the 3 DOF model was preferred for initial simulator. This approach means that the simulator operates solely within the XY-plane and does not account for any variations in terrain, such as hills or tilting of the vehicle. The extension of degrees of freedom is further discussed in Section 8.1.

Inputs and outputs

The model contains two inputs: throttle, given in percent [0, 100], and steering angle, δ, given in radians $[-\frac{\pi}{4}, \frac{\pi}{4}]$. A lot of variables in the model could be considered outputs of the system. In this thesis, however, to simulate the physical Lone Wolf as closely as possible, the outputs were carefully chosen to resemble those measurements available in real life. These are the global X- and Y positions, the yaw-angle, ψ, and velocity, v.

Bicycle model

To simplify the equations and derivation of the model, the bicycle model has been used. The modification combines the two front wheels into a single wheel positioned at the center of the front axle and, similarly, merges the two rear wheels into one wheel located at the center of the rear axle, as illustrated in Figure 3.4 [4]. The variables of the figure are further explained throughout this chapter. The bicycle model assumes that the steering angle of the two front wheels is the same when, in reality, there is a slight difference due to the vehicle's track width.

3.3 Frames of reference

A frame of reference is defined as *"...a means by which an observer specifies positions and describes the motion of bodies"* [30]. Reference frames may operate relative to each other. Earth is often used as the inertial, non-moving reference frame when considering a moving reference frame, such as a moving vehicle. When considering the movement of Earth, the

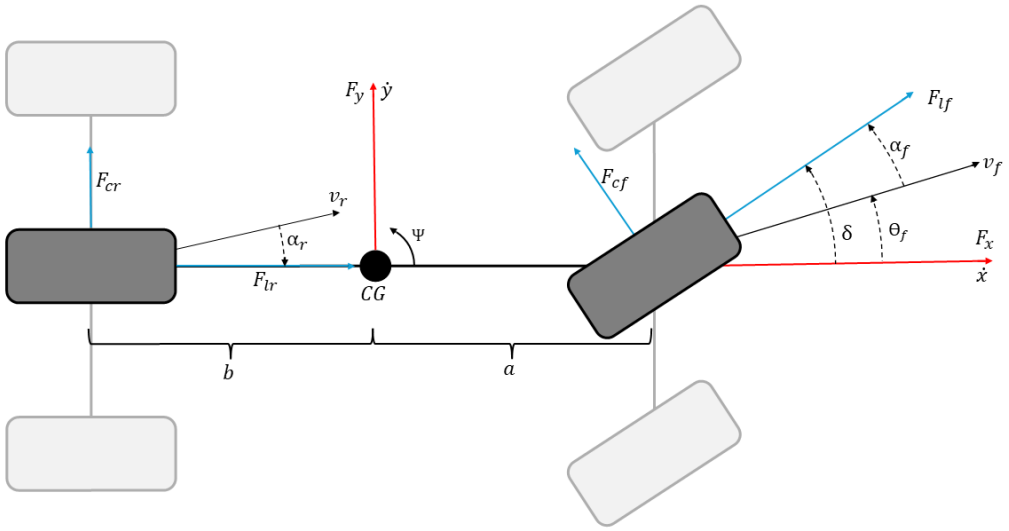


Figure 3.4: Bicycle model

solar system may be considered this inertial reference frame.

The modeling of the vehicle dynamics has been derived from the vehicle's reference frame. as seen in Figure 3.5. This frame originates in the vehicle's center of gravity (CG). The x-axis points in the forward longitudinal direction, while the y-axis points in the lateral direction, positively defined to the left side of the vehicle. The yaw, Ψ, refers to the angle between the vehicle's longitudinal direction and the X-axis of the inertial reference frame. The inertial reference frame is stationary and can be thought of as the world the vehicle is moving in, where the position is specified by the global X and Y coordinates. The position and movement of the vehicle in the inertial reference frame are of interest as outputs. The reference frames are illustrated in Figure 3.5, with the black axes depicting the inertial reference frame and the red axes depicting the vehicle reference frame.

As the dynamic equations of the system are derived in the vehicle's reference system, the vehicle's behavior in the inertial reference frame requires some calculations. These calculations can be derived geometrically as:

$$\dot{X} = \dot{x}\cos\Psi - \dot{y}\sin\Psi \tag{3.1}$$

$$\dot{Y} = \dot{x}\sin\Psi + \dot{y}\cos\Psi \tag{3.2}$$

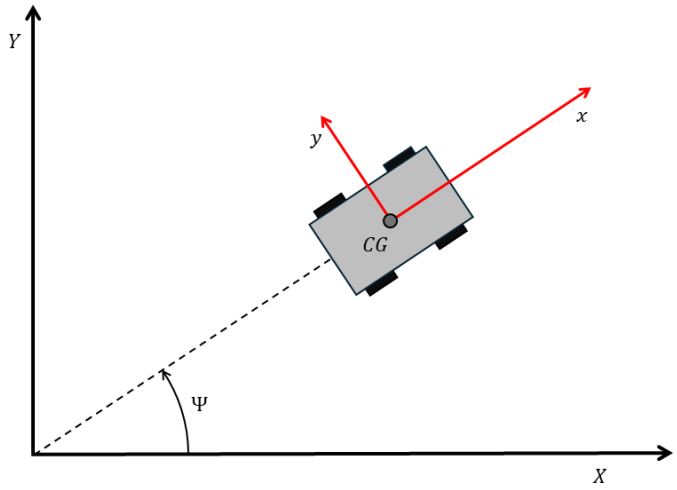


Figure 3.5: Frames of reference

3.4 Dynamic equations of the vehicle body

The derivation of the dynamic equations of the vehicle body is based on first principle physics. Inspiration and help for understanding the physics of the system have been collected from [21], [19], and [29].

Newton's second law states that

$$\sum F = ma \tag{3.3}$$

where F is the total force applied to the body, m is the total mass and a is the acceleration of the body. This law also applies to rotational motion;

$$\sum \tau = I\alpha \tag{3.4}$$

where τ is the total torque, I is the moment of inertia and α is the angular acceleration.

This first principle is the foundation for deriving the equations in the dynamic model. The total force F can be broken down into longitudinal and lateral components, denoted by F_x and F_y. These terms affect the acceleration of the vehicle body in the x-direction and y-direction, respectively. Equations 3.5, 3.6, and 3.7 make up the dynamic model of the vehicle body. For an explanation of the variables, see Table 3.2 at the end of the chapter. Each term of the equations is further explained in the following subsections.

$$F_x = m\ddot{x} = m\dot{y}\dot{\Psi} + F_{xf} + F_{xr} + F_{x,ext} \tag{3.5}$$

$$F_y = m\ddot{y} = -m\dot{x}\dot{\Psi} + F_{yf} + F_{yr} + F_{y,ext} \tag{3.6}$$

$$I_z\ddot{\Psi} = mF_{yf} - mF_{yr} + M_{z,ext} \tag{3.7}$$

Centrifugal forces

The first terms in Equations 3.5 and 3.6, $m\dot{y}\dot{\Psi}$ and $-m\dot{x}\dot{\Psi}$, are due to the centrifugal forces that occur when turning. The centrifugal forces are often referred to as fictitious forces because they are not apparent in the inertial reference frame [6]. When considering the dynamics in a rotating/moving reference frame however, as done in this thesis, these forces are felt by the mass related to the rotating/moving reference system and must be included in the equations of motion [6].

Imagine the vehicle is moving in the positive x-direction and starts to turn left. When the vehicle turns, there is a longitudinal velocity and a counter-clockwise yaw rate when viewed from above. The product of the vehicle mass, m, longitudinal velocity, $\dot{x}$, and the yaw rate, $\dot{\Psi}$, results in a force acting to the right of the vehicle. This is the centrifugal force you feel pushing you outward during a turn. This centrifugal force acts negatively in the y-direction in the vehicle's reference frame.

Likewise, the product of the vehicle mass, m, the longitudinal velocity, $\dot{y}$, and the yaw rate, $\dot{\Psi}$, represent the centrifugal force in the longitudinal direction. This force acts in the positive x-direction.

3.5 Tire modelling

Acceleration and steering of the vehicle are done by altering the force between the tires and the surface on which the vehicle is driving. Because of this, the modeling of the tires and the force applied to the tires are of high importance. As of now, tire modeling is the weakest part of the vehicle simulation industry [49]. Figure 3.6 shows a variety

of approaches to tire modeling, from solely mathematical expressions based on empirical data on the left side to models derived using first principles and complex physics on the right side.

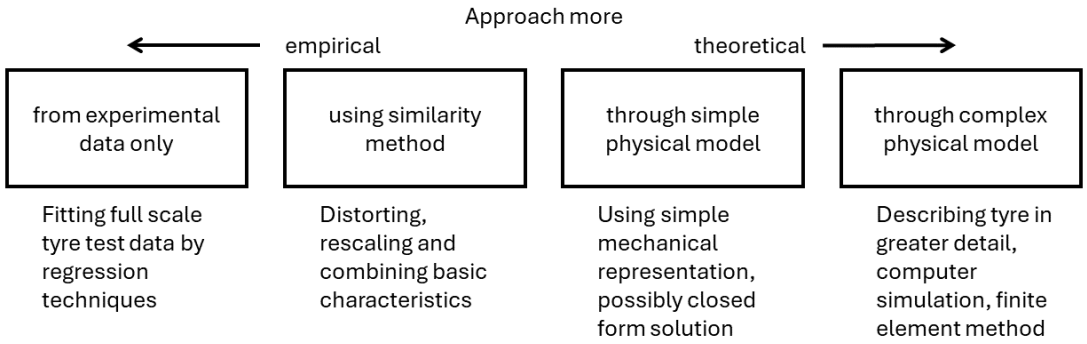


Figure 3.6: Tire modeling methods [27]

3.5.1 Pacejka tire model

Several tire models were considered, and reasoned by a review done by [20], the "Pacejka tire model" was the preferred choice due to its ease of construction and application, high accuracy, and broad and continuous development. It is also one of the most used models in the industry [37]. The Pacejka tire model is a semi-empirical, mathematical calculation of the forces between the tire and the road. It was developed by Hans B. Pacejka, who has also written the literature on the tire modeling theory used in this section [27]. The general form of the model function states

$$y(x) = D\sin\left(C\arctan\left(Bx - E(Bx - \arctan\left(Bx\right))\right)\right) \tag{3.8}$$

where y represents either the longitudinal tire-road forces, F_l, or the cornering tire-road forces, F_c. The variable x represents the longitudinal slip ratio, κ, or the slip angle, α respectively. This function creates a graph shaped like the one seen in Figure 3.7. The interpretation of the parameters B, C, D, and E can be seen in Table 3.1 [27]. A comparison of different values for the Pacejka parameters can be seen in the results in Section 7.1. These parameters can be adjusted to fit empirical data, as further discussed in Section 8.1.

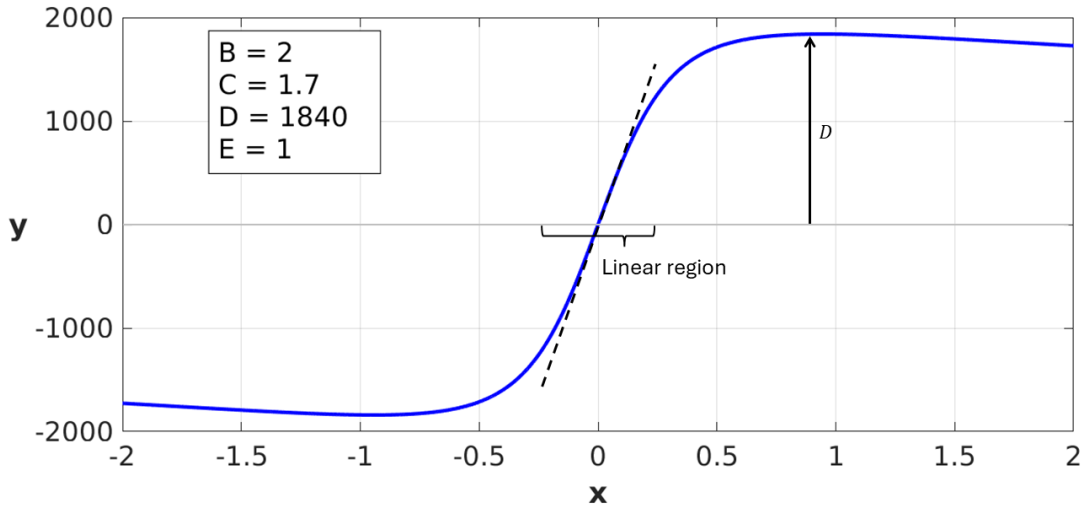


Figure 3.7: Pacejka tire model curve

Parameter	Description
B	Stiffness factor of tire
C	Curve factor
D	Max force ($\mu \cdot m \cdot g$, where μ is friction, m is mass and g is gravity)
E	Stretch/compression of the curve
BCD	Cornering stiffness (slope of curve in linear region)

Table 3.1: Parametres of the Pacejka tire model

3.5.2 Longitudinal tire forces

The rotational tire dynamics in the tire's longitudinal direction, visualized in Figure 3.8, can be described as

$$I_w \dot{\omega} = \tau_w - F_{li} r_w - C_{rr} \cdot m \cdot g \tag{3.9}$$

where $\dot{\omega}$ is the angular acceleration of the wheel, τ_w is the torque applied from the engine to the wheel, r_w is the radius of the wheel, I_w is the moment of inertia of the wheel, and F_{li} is the longitudinal force between the tire and the road surface. C_{rr} is a dimensionless rolling resistance coefficient, m is the weight on the wheel and g is gravity.

The longitudinal forces acting between the tire and the surface, F_{li}, depend on the slip

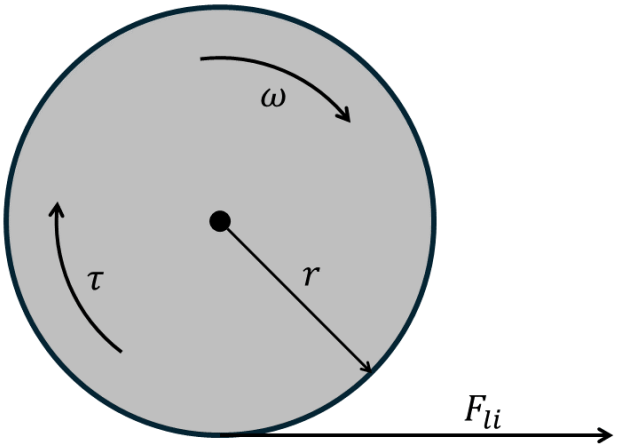


Figure 3.8: Longitudinal tire dynamics

ratio, κ. This slip ratio is defined as the difference between the wheel's angular velocity, ω and the vehicle body's longitudinal velocity, $\dot{x}$ [27]. The slip ratio, κ, is calculated as

$$\kappa = -\frac{\dot{x} - \omega r_w}{\dot{x}}. \tag{3.10}$$

The sign of κ is chosen such that when increasing the velocity, $F_{li} > 0$, and while slowing down, $F_{li} < 0$.

Inserting κ (given in percent) into Equation 3.8, we get

$$F_l(\kappa) = D_l \sin\left(C_l \arctan\left(B_l\kappa - E_l(B_l\kappa - \arctan\left(B_l\kappa\right))\right)\right) \tag{3.11}$$

where B_l, C_l, D_l and E_l refer to the Pacejka parameters for the longitudinal tire forces.

3.5.3 Lateral tire forces

The lateral tire forces, F_{l_i}, depend on the slip angle, α. This slip angle is defined as the difference between the steering angle, δ, and the actual direction of the axle velocity, θ, illustrated in Figure 3.4 [27]. As only the front axle is used for steering on Lone Wolf, the steering angle of the rear wheels will always be zero. As a result, α_r will always equal the direction of the velocity of the rear axle, $-\theta_r$. α_f and α_r are calculated as

$$\alpha_f = \delta - \theta_f = \delta - \arctan\left(\frac{\dot{y} + a\dot{\Psi}}{\dot{x}}\right) \tag{3.12}$$

$$\alpha_r = -\theta_r = -\arctan\left(\frac{\dot{y} - b\dot{\Psi}}{\dot{x}}\right) \tag{3.13}$$

where δ is the steering angle, a is the distance from the vehicle center of gravity (CG) to the front axle, and b is the distance from the vehicle CG to the rear axle. θ_f and θ_r are derived from Figure 3.3 using trigonometric identities.

Inserting α into Equation 3.8, the lateral tire forces can be stated as:

$$F_c(\alpha) = D_c \sin\left(C_c \arctan\left(B_c\alpha - E_c(B_c\alpha - \arctan\left(B_c\alpha\right))\right)\right) \tag{3.14}$$

where B_c, C_c, D_c and E_c refers to the Pacejka parameters for the lateral tire forces.

3.5.4 Combined tire forces

The formulas described for longitudinal and lateral forces in Sections 3.5.2 and 3.5.3, refer to situations of pure slip. This means the slip angle, $\alpha = 0$ for the longitudinal forces and the slip ratio, $\kappa = 0$ for the lateral forces. In other words, only one type of slip occurs at a time. While steering and accelerating the vehicle at the same time, the tires will experience both types of slip. This will affect the tire's characteristics, which must be accounted for. This is done by normalizing the slip ratio, κ, and slip angle, α [27]. The normalized slips are given by

$$\kappa_{\text{norm}} = \frac{\kappa}{\kappa_{\text{peak}}} \tag{3.15}$$

$$\alpha_{\text{norm}} = \frac{\alpha}{\alpha_{\text{peak}}} \tag{3.16}$$

where

- κ is the actual slip ratio.
- κ_{peak} is the slip ratio at peak longitudinal tire force.
- α is the actual slip angle.
- α_{peak} is the slip angle at peak lateral tire force.

Using the normalized slip values, the total slip S is calculated as

$$S = \sqrt{(\kappa_{\text{norm}})^2 + (\alpha_{\text{norm}})^2} \tag{3.17}$$

which can be used to calculate the modified slip values;

$$\kappa_{\text{mod}} = S \cdot \kappa_{\text{peak}} \tag{3.18}$$

$$\alpha_{\text{mod}} = S \cdot \alpha_{\text{peak}} \tag{3.19}$$

Finally, weighting the normalized slips and multiplying them by Equations 3.11 and 3.14 as functions of κ_{mod} and α_{mod}, we get the resulting tire forces during combined slip:

$$F_{l_{\text{combined}}} = \frac{\kappa_{\text{norm}}}{S} \cdot F_l(\kappa_{\text{mod}}) \tag{3.20}$$

$$F_{c_{\text{combined}}} = \frac{\alpha_{\text{norm}}}{S} \cdot F_c(\alpha_{\text{mod}}) \tag{3.21}$$

Tire forces to vehicle reference frame

F_{xi} and F_{yi} in Equations 3.5 and 3.6 are the forces acting on the vehicle from the tires, which are calculated as described in the previous subsections. These forces are calculated in the tire reference frame and have to be transferred to the vehicle reference frame to be used in the dynamic equations of the vehicle body. This is calculated geometrically as

$$F_{xf} = F_{lf} \cos\delta - F_{cf} \sin\delta \tag{3.22}$$

$$F_{yf} = F_{lf} \sin\delta + F_{cf} \cos\delta \tag{3.23}$$

where l equals the tire longitudinal direction, c equals the tire lateral (cornering) direction, and δ equals the steering angle. This is illustrated in Figure 3.9. As the Lone Wolf has front axle steering only, this change of reference frame is only needed for the front wheel. The rear wheel stays aligned with the vehicle reference frame and can be used directly in the dynamic equations.

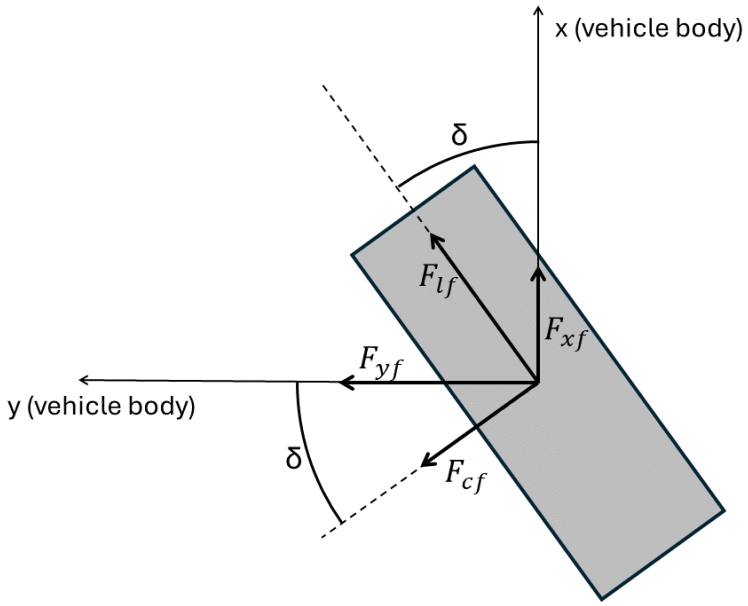


Figure 3.9: Tire forces to vehicle reference frame

3.6 Powertrain modeling

The powertrain consists of everything between the throttle and output force of the wheels [12]. On the Lone Wolf, the main components of the powertrain are the engine, transmission, final drive, and differential, as seen in Figure 3.10. This system is difficult to model based on physics because of its complexity. Therefore, an empirical model based on the powertrain dynamics is the best solution [15]. The limiting factor of modeling with this approach is the need for precise data on the powertrain, which has proven to be hard to acquire from the producers.

3.6.1 Engine modeling

Modeling the torque output of a combustion engine using physics-based methods presents significant challenges, as it requires a broad set of data specific to the engine being modeled [8]. This data, known as the engine map, which is the output torque based on the engine RPM and throttle input, plays a crucial role in determining the powertrain dynamics. The engine's RPM must be calculated in this model, while the throttle is an input variable.

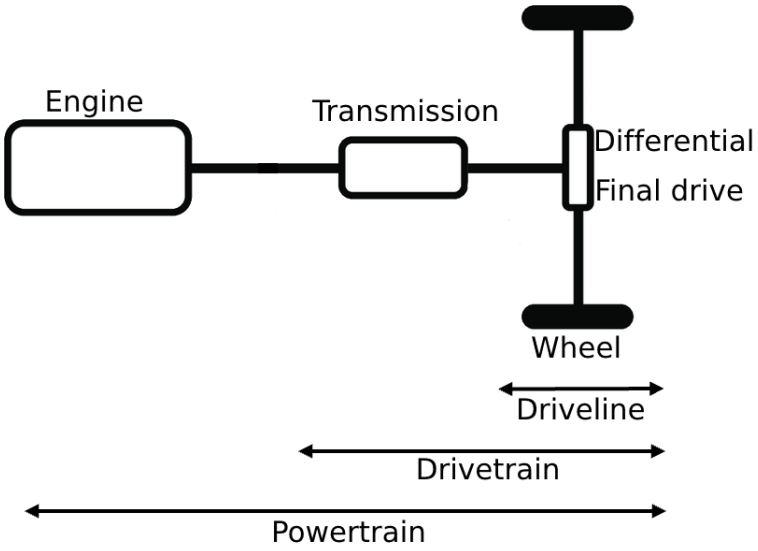


Figure 3.10: Powertrain flowchart [3]

3.6.2 Drivetrain modeling

As seen in Figure 3.10, the drivetrain is everything in the powertrain except the engine [12]. This constitutes the transmission and the driveline.

CVT modeling

The Lone Wolf ATV has a Continuously Variable Transmission (CVT). Modeling the CVT is done based on Olav Aaen's literature [1]. The modeling of the CVT can be made where the gear ratio is approximated based on the vehicle's velocity and the transmission characteristics, as seen in Figure 3.11. The dotted line graphs the engine RPM as a function of the velocity of the vehicle. The CVT characteristics can be simplified into four stages, which are:

- **The Engagement stage:** During this stage, the CVT is not fully engaged yet, and the torque on the secondary side, the wheel, is minimal during this stage.
- **The Shift RPM stage:** During this stage, the RPM of the engine rises while the CVT is locked to the low ratio.
- **The Straight shift stage:** During this stage, the gear ratio changes from the low to the high ratio, and during this, the engine RPM is relatively unchanged.

- **The Shift Out stage:** During this stage, the gear ratio is constant at the high ratio while the rpm rises to the max RPM of the engine.

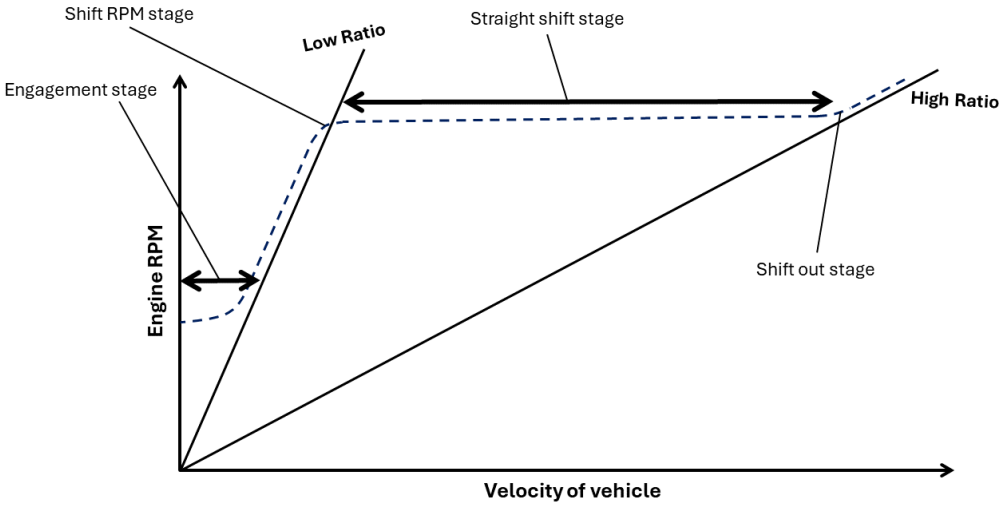


Figure 3.11: Common characteristics of a CVT [1]

Driveline modeling

The main parts of a driveline for an offroad vehicle are the final drive and the differential. The final drive gear ratio is the final gearing of the drivetrain before the torque is transmitted by the wheels. This gear is placed on the front and back axle with a constant gear ratio [9]. In this model, it is unnecessary to model the differential because the ATV model is based on a bicycle model, meaning each axle has only one tire. Approximations of these parts combined can make the simplified driveline model. This driveline can then be combined with the CVT to model the complete drivetrain based on the ideal formula for converting engine torque to wheel torque [52]. This is stated as

$$\tau_w = \frac{i_x \cdot i_o \cdot \tau_e}{n_w} \tag{3.24}$$

where τ_w is the torque output at the wheel, i_x is the gearbox ratio, i_0 is the final drive ratio, τ_e is engine torque and n_w is the number of wheels connected to the transmission. This is the ideal, no-loss-based model for the gearing between an engine and the wheels of a vehicle. Combined with the gear ratio characteristics from figure 3.11, this makes an ideal approximation of a generic CVT combined with a driveline.

Variable	Description
$x, \dot{x}, \ddot{x}$	Longitudinal position, velocity, acceleration (vehicle ref. frame)
$y, \dot{y}, \ddot{y}$	Lateral position, velocity, acceleration (vehicle ref. frame)
$X, \dot{X}$	X position, velocity (world ref. frame)
$Y, \dot{Y}$	Y position, velocity (world ref. frame)
$\psi, \dot{\psi}, \ddot{\psi}$	Yaw angle, yaw rate, yaw acceleration
F_{l_f}, F_{l_r}	Force in tire longitudinal direction (tire ref. frame)
F_{c_f}, F_{c_r}	Force in tire lateral direction (tire ref. frame)
F_{xf}, F_{xr}	Longitudinal tire forces (vehicle ref. frame)
F_{yf}, F_{yr}	Lateral tire forces (vehicle ref. frame)
$F_{x,ext}, F_{y,ext}$	External forces (longitudinal, lateral)
$M_{z,ext}$	External moment
θ_f, θ_r	Vehicle velocity angle
α_f, α_r	Slip angle
κ	Slip ratio
δ	Steering angle
$\omega, \dot{\omega}$	Angular velocity, angular acceleration of tires
τ_w	Torque applied from to tire at final drive
τ_e	Torque applied to tire from the transmission

Table 3.2: Table of variables (f = front, r = rear)

Parameter	Description
m	Total vehicle mass
a	Distance from CG to front axle
b	Distance from CG to rear axle
Iz	Moment of inertia of vehicle body
r_w	Radius of wheels
I_w	Moment of inertia of wheels
B_l, C_l, D_l, E_l	Pacejka parameters (longitudinal)
B_c, C_c, D_c, E_c	Pacejka parameters (lateral)
i_x	Gearbox ratio
i_o	Final drive ratio
n_w	Number of wheels connected to transmission

Table 3.3: Table of parameters

Chapter 4

Dynamic Model - Implementation

This chapter explains how the dynamic model from Section 3 was implemented for simulation in Simulink, how the simulator communicates with the ROS2-system, and describes the graphical user interface (GUI) that was made to visualize the simulation in real time. The chapter aims to satisfy the second and fourth objective, stating:

2. **Implement the dynamic model in a simulator:** The model identified in the first objective must be implemented in a ROS2-compatible simulator.

4. **Implement the control algorithm into the Lone Wolf system:** The developed control algorithm has to be implemented into the current system- and software architecture of the current Lone Wolf project so that it can easily be used in future projects.

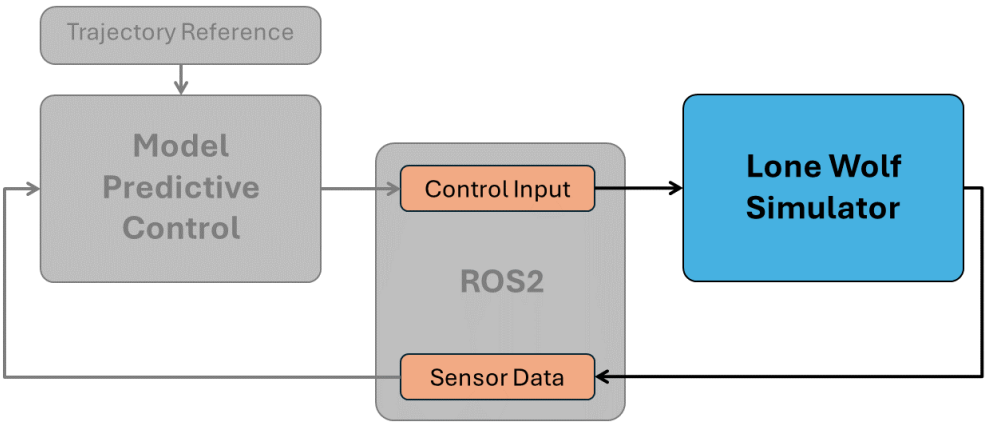


Figure 4.1: Project flowchart: Simulink

4.1 Previous work

As the Lone Wolf has been subject to summer projects and academic theses over several years, challenges have been approached in various ways. The bachelor thesis carried out at NTNU in 2022, *ROS Simulated World for ATV with SLAM*, also developed a simulator for their thesis [13]. Their main objective was to perform "Simultaneous Localization and Mapping" (SLAM), and the actual behavior of the ATV was not prioritized in the simulator. The bachelor thesis from NTNU in 2023, *Graph-based Path Planning in a Simulated Rough Terrain*, also states that *"driving the ATV sometimes feels unnatural and the vehicle does not behave as one would expect"* [17]. This previous simulator was created utilizing the software Gazebo, which was also considered for this thesis. However, due to difficulties regarding the modeling of the tire-road physics in Gazebo, another software was chosen in consultation with the client. MathWorks software Simulink was preferred, as it allows the user to build the dynamic system from the beginning and does not predefine any physics. It also has support for ROS2 communication, which is important for mimicking the actual software architecture of the Lone Wolf system.

4.2 Simulink introduction

Simulink, developed by MathWorks, is a graphical programming environment for modeling, simulating, and analyzing dynamic systems [24]. Its primary interface is a block diagramming tool and a customizable set of block libraries. Simulink is widely used in engineering for various applications, ranging from dynamic system simulation to embedded system design. This software supports system-level design, simulation, and automatic code generation. It provides an interactive graphical environment and a customizable block library set that lets users design, simulate, implement, and test various time-varying systems, including communications, controls, signal processing, and image processing [24]. In this thesis, Simulink is used to simulate the ATV, and all the dynamics discussed in Chapter 3 are implemented using Simulink. In addition to this, the GUI is developed and runs in Simulink, utilizing the real-time capabilities of the software to show the movement of the ATV during simulation.

4.3 Model implementation

Recall the first thesis objective stating to *"Structure the model to allow adjustable parameters, enabling future modifications to closely simulate the physical Lone Wolf ATV dynamics."* Because of this, the model has been structured to include adjustable parameters, such as the Pacejka parameters explained in Section 3.5.1. These can be fine-tuned using the grey box method mentioned in Section 3.1 once real-world data from the physical Lone Wolf is available.

The simulator consists of two main files. The main file, `LoneWolfSimulator.slx`, which contains the dynamic model and executes the simulations. The second file, `LoneWolfParams.m`, is used to declare all the constant parameters of the model, seen in Table 3.3.

The physics and theory of the model implemented are explained in Chapter 3. Figure 4.2 depicts the first view when opening the `LoneWolfSimulator.slx` file in Simulink. The system is split into designated sub-systems, and closely resembles the flowchart provided in Figure 3.3. This resemblance is intentional, designed to clarify the system's theoretical foundations and streamline the learning process for individuals who will engage with the system in the future.

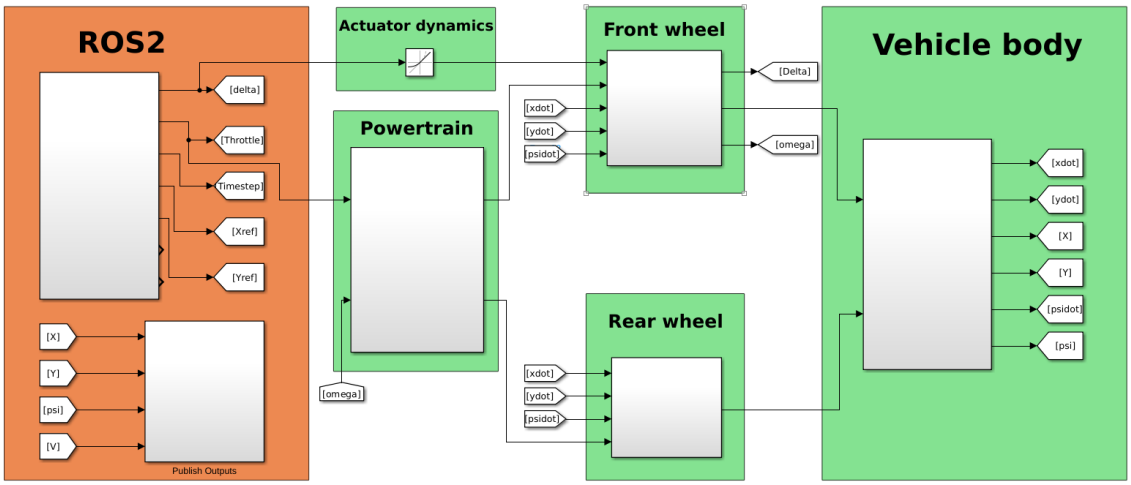


Figure 4.2: Simulink: Screenshot of dynamic model

4.3.1 Simulink subsystem: Vehicle body

The dynamic equations of the vehicle body, described in Section 3.4, were built using basic Simulink features, such as summation blocks, gain blocks, and integrator blocks. Figure 4.3 depicts the inside of the subsystem labeled "Vehicle body" in Figure 4.2. The subsystem has the tire forces as inputs, and provides $\dot{x}$, $\dot{y}$ and $\dot{\Psi}$ as outputs.

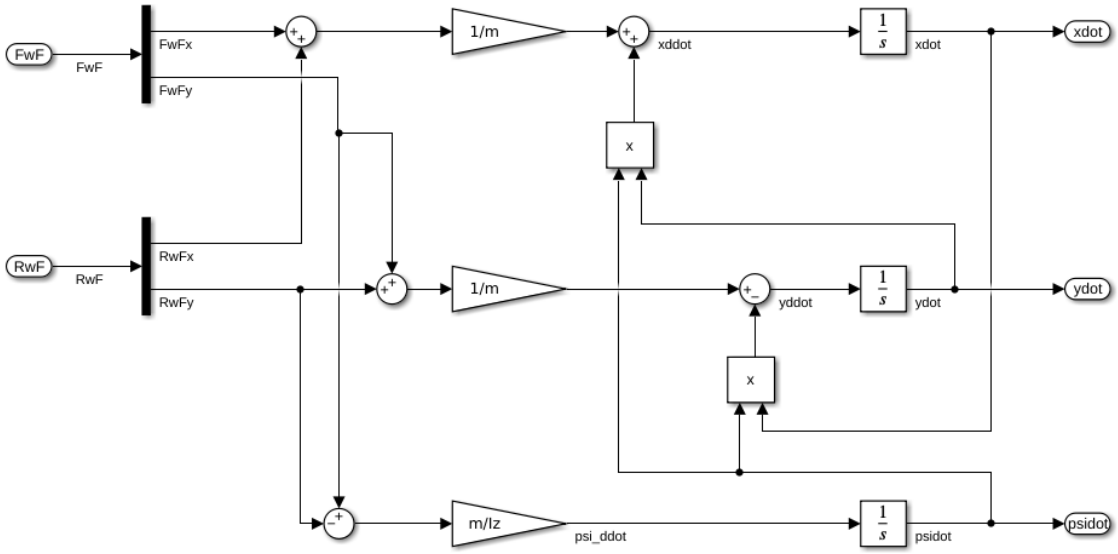


Figure 4.3: Simulink: Dynamic equations of vehicle body

4.3.2 Simulink subsystem: Front wheel and Rear wheel

The tire forces are calculated as described in Section 3.5. Figure 4.4 shows the inside of the subsystem labeled "Rear wheel" in Figure 4.2. The two grey boxes in the figure contain the Pacejka tire model for longitudinal and lateral rear tire forces. These forces are then put inside an array, which is the input of the "Vehicle body" subsystem. The subsystem labeled "Front wheel" in Figure 4.2 contains the corresponding functions for the front tire forces.

4.3.3 Simulink subsystem: Powertrain

Due to the lack of relevant data for modeling the combined transmission and engine from an engine map, they have been modeled as separate parts. Figure 4.5 is an overview of how the different parts of the powertrain are connected in the Simulink model.

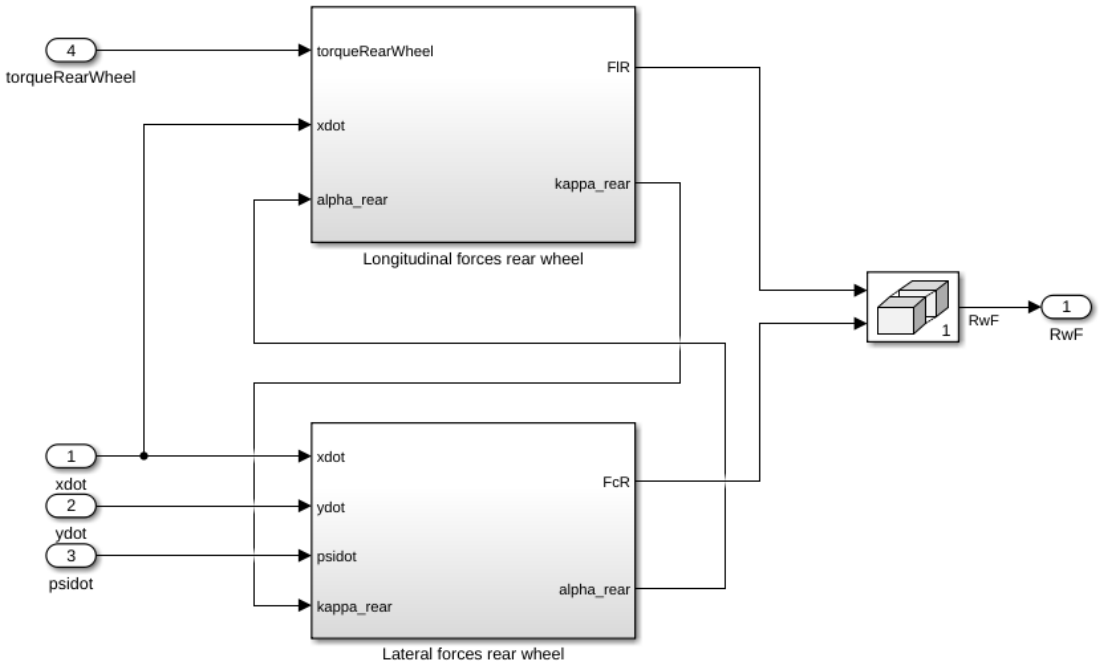


Figure 4.4: Simulink: Tire model implementation

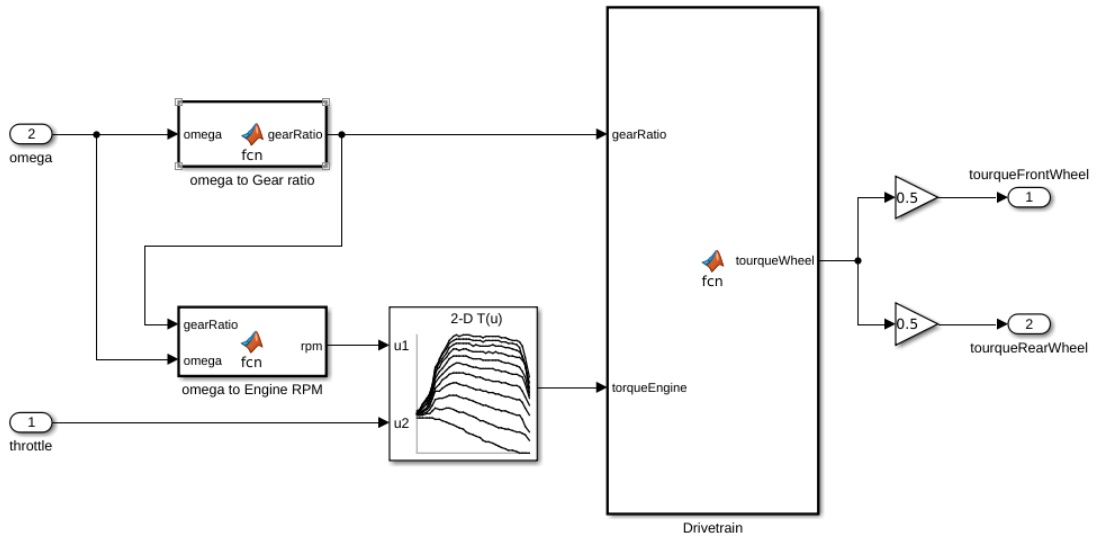


Figure 4.5: Simulink: Powertrain implementation

Engine

Given the absence of relevant data from the engine in Lone Wolf, a substitute engine map was employed to establish correlations between the throttle, RPM, and torque. The substitute map was obtained from an article modeling an internal combustion engine [8] and is of an Audi 3.2l V6 TSI engine, which is a 4-stroke V6 petrol engine [7]. As a result

of the difference between the actual engine of the ATV and the engine map obtained, some modifications had to be made to the model after using this engine map. This includes a reduction constant inside the gearbox, tuned for the simulated vehicle to match the steady state of 60 km/h at full throttle on the real Lone Wolf. For the RPM calculations, the wheel's angular velocity has been used in combination with the estimated final drive value and the current ratio of the Continuously Variable Transmission (CVT) to find the actual RPM of the engine. This value, together with the throttle input, will then be used in the 2-D lookup table of the engine map, seen in Figure 4.6, to calculate the engine's output torque.

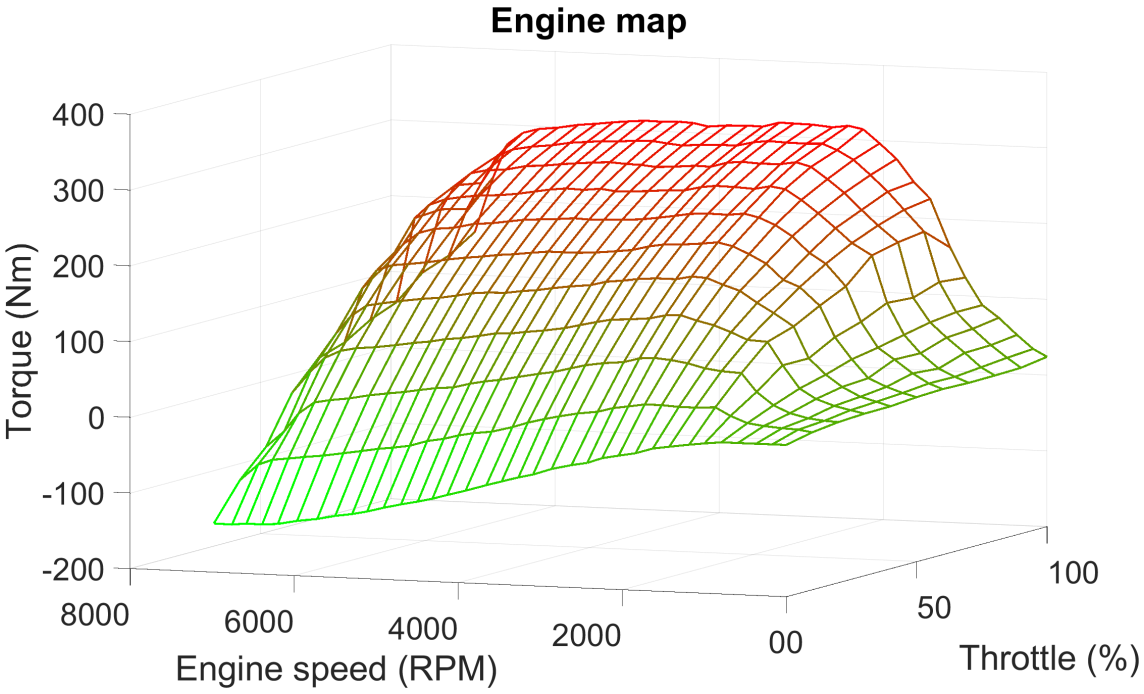


Figure 4.6: Simulink: Engine map

Drivetrain

The solution for implementing the transmission model utilizes a CVT modeled after the CVT system on Lone Wolf [51], detailed in Figure 3.11. This model reflects typical CVT behavior, where the gear ratio linearly adjusts between two fixed ratios based on the wheel's angular velocity. For model simplification, the gear ratio does not fall below the lowest set ratio during engagement.

To model the final drive ratio, the maximum velocity of the ATV was set at 60 km/h. Based on this velocity, the final drive gear ratio was calculated to ensure that the maximum RPM

of the substitute engine corresponded to this velocity. To address discrepancies between the actual engine and the one depicted in the substitute engine map, see Figure 4.6, a reduction constant was introduced. Initially, this constant was set to match the peak torque differences, but subsequent adjustments lowered it to represent an ATV's natural acceleration curve more realistically. The reduction constant was manually fine-tuned until the model produced satisfactory results, matching the maximum velocity of 60 km/h at full throttle.

4.3.4 Simulink subsystem: Actuator dynamics

Three actuators are available on the physical Lone Wolf: throttle, brake, and steering. In practice, only the throttle and steering are used. The brake currently behaves as a binary switch and is used only for emergency stops. During normal driving, the engine braking and rolling resistance are utilized to de-accelerate the vehicle and lower the speed. Because of this, only the steering input and throttle input have been included in the model for the simulator.

Steering

The steering angle, δ, is one of the two inputs to the system. The steering is controlled using a servo motor connected to a double bar linkage, which are then connected to where the handlebar of the ATV would usually be. As the servo has a limit on its rate of change, the steering input given to the system passes through a rate limiter constrained at $\Delta\delta_{max}$.

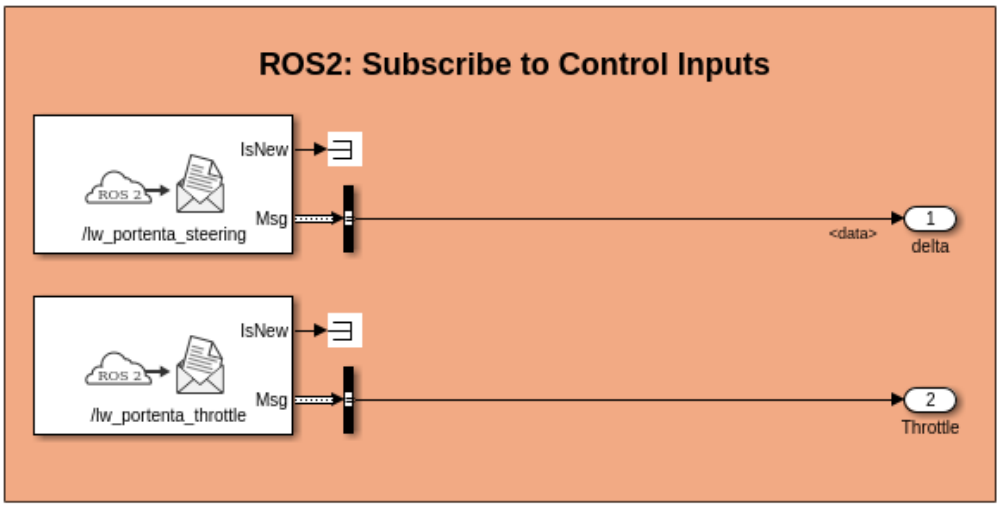
Throttle

The throttle input is connected directly to the electrical throttle signal box on the ATV. Therefore, the actuation of the throttle input for the system is seen as instant, and no input dynamics from the throttle have been applied.

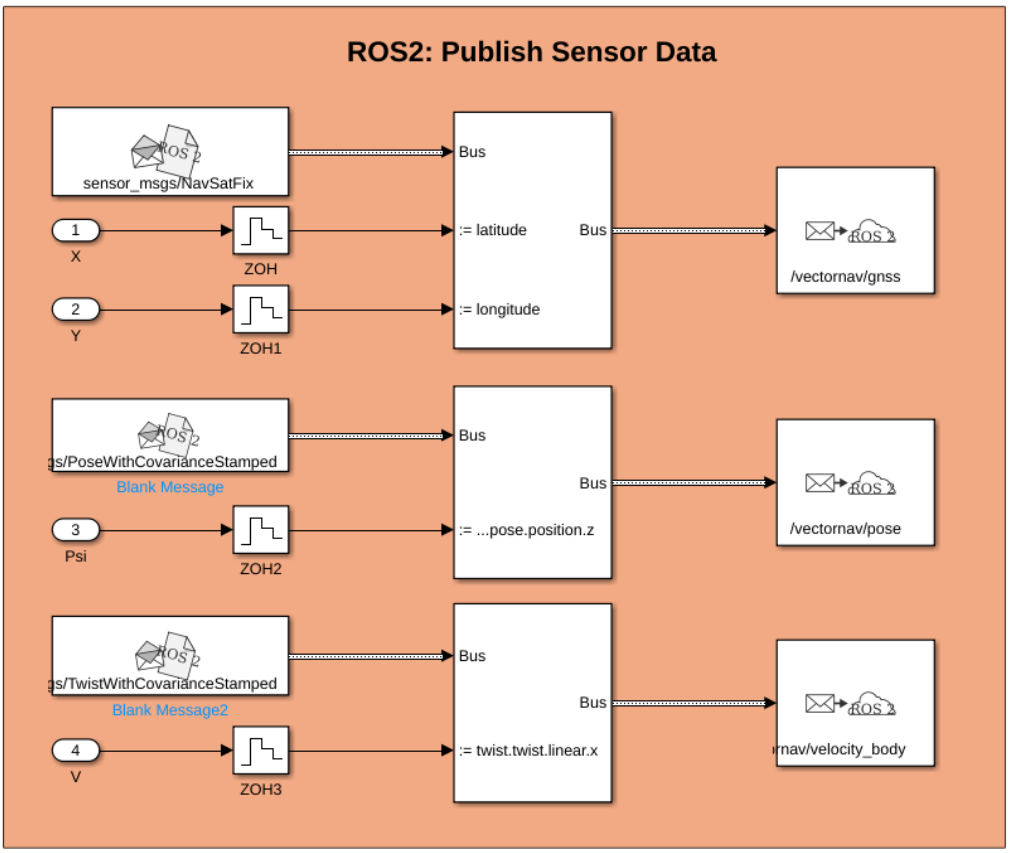
4.3.5 Simulink: ROS2

The fourth thesis objective is to replicate the interface for the physical Lone Wolf as closely as possible. The simulator should also mimic how the physical Lone Wolf communicates

with the ROS2 system. Figures 4.7a and 4.7b shows how Simulink's integrated ROS2 function blocks are utilized to subscribe to the control inputs and publish the sensor data.



(a) ROS2 in Simulink: Subscribing to control inputs



(b) ROS2 in Simulink: Publishing sensor data

Figure 4.7: ROS2 in Simulink

4.4 Real time simulation

Implementing real-time simulation is crucial for accurately testing and validating the Model Predictive Control (MPC) system developed for the Lone Wolf ATV. To achieve this, the simulation utilizes MATLAB's Real-Time Kernel, which allows the execution of the Simulink model at speeds close to real-time. The kernel is installed by typing the following line in the MATLAB Command Window:

```
>> sldrtkernel -install
```

Listing 4.1: Installing Real-Time Kernel in MATLAB

The primary objective of using MATLAB's Real-Time Kernel in this thesis is to ensure that the MPC's computational demands meet realistic time constraints. This approach simulates the system's behavior under operational conditions, where the control algorithms must respond within strict time frames typical of real-world scenarios.

4.5 Simulator GUI

The simulator's graphical user interface (GUI), depicted in Figure 4.8, is organized into two main sections. On the left, the GUI graphically represents the ATV position as a red box with wheels. This includes the reference trajectory, shown as a blue dotted line, and the actual path taken by the ATV, represented by a black solid line. The right section of the GUI displays various state parameters for the vehicle plotted against time across three separate plots:

- **Velocity | Throttle:** The top plot shows the vehicle's velocity in km/h with a green solid line. Alongside this, the throttle input from the MPC is plotted as a percentage, indicated by a grey dotted line.
- **Steering angle:** The middle plot shows the vehicle's steering angle in radians as a solid blue line. In addition, the steering input is plotted as a grey dotted line.
- **Position error:** The bottom plot shows the position error of the vehicle, measured as an absolute value in meters and represented by a solid red line. The error is calculated as the distance from the vehicle's center of gravity to the closest point on the reference trajectory.

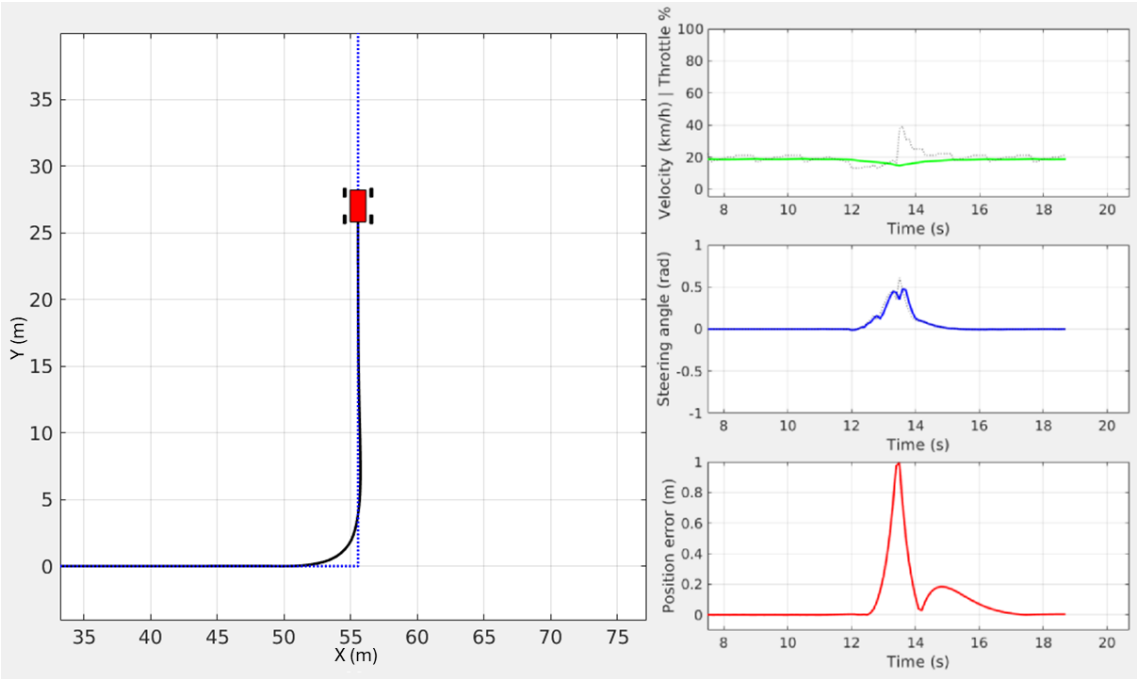


Figure 4.8: Simulink: GUI screenshot while in use

Chapter 5

Model Predictive Control - Theory

This chapter introduces the concept of Model Predictive Control (MPC) and explains the mathematical framework behind it. The theory explained throughout this chapter is used in the MPC, as illustrated in Figure 5.1. The chapter focuses on the third objective, which states:

3. **Development of a Model Predictive Controller (MPC):** Develop an MPC to be tested in the simulator. The controller should be based on measurements from sensor data, which resembles the information that can be gathered from the physical Lone Wolf, such as GPS- and IMU data.

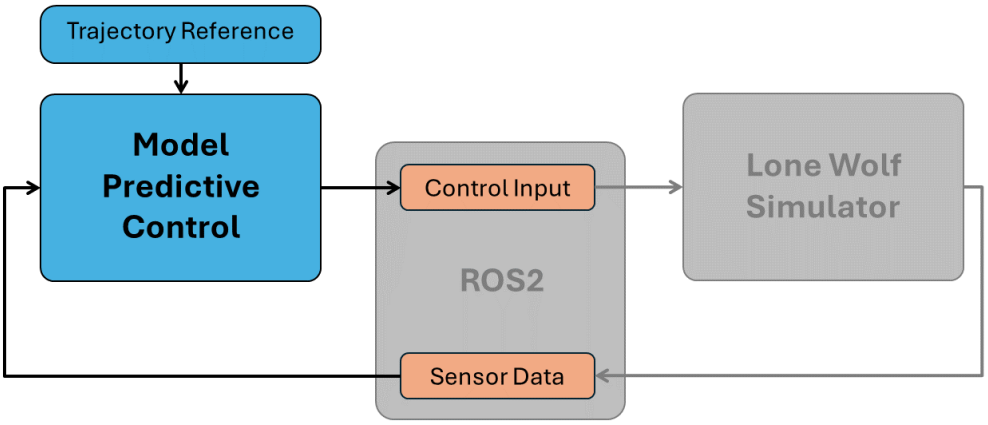


Figure 5.1: Project flowchart: MPC

5.1 Model Predictive Control

Model Predictive Control (MPC) is an advanced, optimal control algorithm that combines dynamic optimization with feedback control. It can be used to control different dynamic systems while satisfying a set of constraints. As the name implies, a mathematical dynamic model of the system is required to build such a controller. The principle of MPC is shown in Figure 5.2. At every sampling instant, the controller calculates the solution of a finite horizon optimal control problem, considering the system's state at the sampling time t' [11]. The controller calculates the optimal sequence of inputs over the Prediction Horizon N, and minimizes the cost function. Only the first input calculated will be applied to the system, shown as a thick, red line in fig 5.2. The whole calculation is done again at the next sampling time. [11]

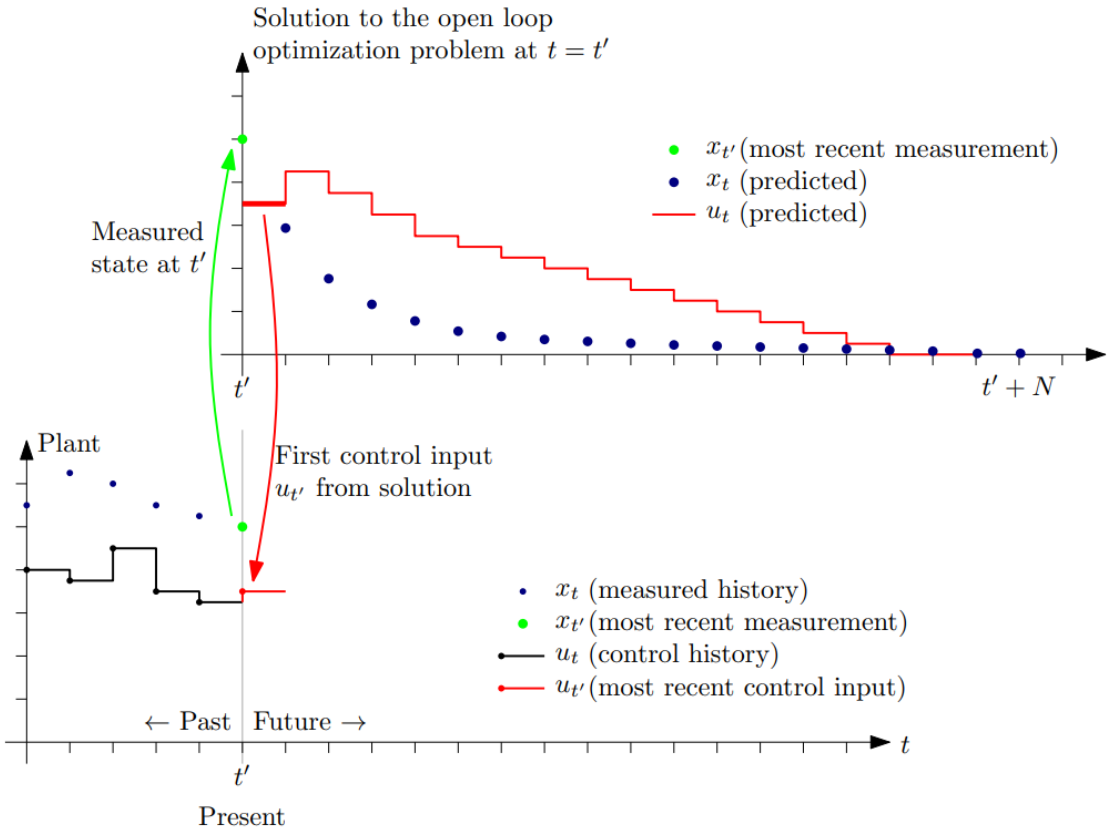


Figure 5.2: Illustration of the MPC principle [11]

For this project, due to the significant non-linearities found in the dynamic model for Lone Wolf, a non-linear MPC (NMPC) was chosen. The non-linear dynamics are described in detail in Chapter 3 and 6.

5.2 Mathematical framework for the non-linear MPC

Mathematically, the non-linear MPC algorithm can be formulated in the following way:

$$\min_{z\in\mathbb{R}^n} f(z) = \sum_{k=0}^{N-1} \frac{1}{2}\left[(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{ref}})^\top Q_{t+1}(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{ref}}) + \Delta\mathbf{u}_k^\top R_{\Delta t}\Delta\mathbf{u}_k\right] \tag{5.1}$$

Subject to the constraints:

$$x_{k+1} = g(x_t, u_t) \tag{5.2}$$

$$x^{\text{low}} \leq x_t \leq x^{\text{high}} \tag{5.3}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}} \tag{5.4}$$

where

$$Q_t \succeq 0 \tag{5.5}$$

$$R_{\Delta t} \succeq 0, \tag{5.6}$$

and x_0 and u_{-1} are given.[11]

In equation 5.1, z is given as

$$z = (x_1^\top, x_2^\top, \ldots, x_N^\top, x_1^{\text{ref}\,\top}, x_2^{\text{ref}\,\top}, \ldots, x_N^{\text{ref}\,\top}, \Delta u_1^\top, \Delta u_2^\top, \ldots, \Delta u_N^\top), \tag{5.7}$$

meaning that the cost function takes all in arguments which are the vectors representing the physical state of the system, x_k, reference, x_k^{ref}, and the change in control input, Δu_k. All the values are real numbers.

The constraint stated in equation 5.2 states that

$$x_{k+1} = g(x_t, u_t),$$

which means that all the possible x_k are constrained to some state allowed by the the discrete dynamic model, based on the previous state and control input. This constrains the optimization to only consider states that are deemed "possible", based on understanding of the system given by the dynamic equations.

Equations 5.3 and 5.4 refer to physical constraints. Equation 5.3 states that

$$x^{\text{low}} \leq x_t \leq x^{\text{high}},$$

meaning that the state of the system, x_t, must stay within the physical limits x^{low} and x^{high}. The limits are based on the physical limitations of Lone Wolf. For example, the steering angle is limited between $\pm 45°$.

Equation 5.4 states that

$$u^{\text{low}} \leq u_t \leq u^{\text{high}},$$

which constrains the control inputs to be the values accepted by the actuators' control logic. This means that the controller is not allowed to set the control input higher, or lower, than the limits. For example, on Lone Wolf, the throttle is limited between 0 to 100.

Equations 5.5 and 5.6 refers to the structure of the Q_t and $R_{\Delta t}$ matrices. The operator $\succeq$, in the context of matrices, means that the matrix is at positive definite, or positive semi-definite. This means that

$$Q_t \succeq 0, \text{ if } x^\top \cdot Q_t \cdot x \geq 0 \text{ for all } x \in \mathbb{R}^n, \text{ where } Q_t \in \mathbb{R}^{nxn}. \tag{5.8}$$

This constraints ensures that there is no tuning of the Q and R matrices that allows some states to incorrectly lower total cost calculated in the cost function for some state x_t.

Essentially, all this mean that the function to be minimized is constrained to the predicted states, governed by the discretized dynamic model, and by the physical limitations of the system. The optimal sequence of control is found as the sequence that minimizes the cost function, tuned by the Q_t and $R_{\Delta t}$ matrices. Only the first set of control inputs are applied to the system, and the calculations are repeated for all following time-steps.

5.3 Discretizing dynamic systems

Recall that MPC is a discrete-time controller. Often, the dynamic equations for a system is derived in continuous time. Therefore, the dynamic system must be rewritten to a discrete form. Discretizing a system means taking a dynamic system from the form

$$\dot{y} = g(x_t, u_t) \tag{5.9}$$

to

$$y_{k+1} = g(x_k, u_k), \tag{5.10}$$

which deals in discrete time steps, k.

There are many different methods for completing the discretization process. Appendix C contains a discussion of three methods: Forward Euler, Runge-Kutta of the 2nd order, and an exact analytical approach. The appendix covers the rest of the theory for discretizing systems.

5.4 Defining the Q and R matrices

Recall that the cost function was defined as in equation 5.11.

$$\min_{z\in\mathbb{R}^n} f(z) = \sum_{k=0}^{N-1} \frac{1}{2}\left[(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{ref}})^\top Q_{t+1}(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{ref}}) + \Delta\mathbf{u}_k^\top R_{\Delta t}\Delta\mathbf{u}_k\right] \tag{5.11}$$

This cost function squares the difference between input and reference, and squares the changes in control inputs. Equation 5.13, demonstrates the multiplication done in the $R_{\Delta t}$ of the component, with the example situation where the $R_{\Delta t}$-matrix is the matrix in equation 5.12. This is done for clearer demonstration of the concept.

$$R_{\Delta t} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \tag{5.12}$$

$$\Delta\mathbf{u}_k{}^T \cdot R_{\Delta t} \cdot \Delta\mathbf{u}_k = \begin{bmatrix} \Delta u_1[k] & \Delta u_2[k] \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} \Delta u_1[k] \\ \Delta u_2[k] \end{bmatrix} = \Delta u_1[k]^2 + 2 \cdot \Delta u_2[k]^2 \tag{5.13}$$

It is clear to see that the weighting of the $R_{\Delta t}$ matrix is what determines how much each of the components in $\Delta\mathbf{u}_k$ is weighted in the cost function. The same principle holds for the Q matrix. Changing the entries in these two matrices define how the different states, or inputs, are weighted in the cost function.

5.5 Optimization problem

The optimization problem is to minimize the cost function restated in equation 6.27. The function is minimized when the optimal sequence of control inputs are identified, which are ones that gives the lowest value for $f(z)$. How the different states and control inputs

are weighted in the cost function is described in section 5.4. Minimizing the cost function is generally solved by utilizing numerical methods because of the complexity and time-limits posed in MPC problems. Optimization is a broad and complex field, and further discussion is beyond the scope of this thesis.

5.6 Defining the reference

When discussing control of moving robots and autonomous vehicles, the terms "path" and "trajectory" are often used. While they are closely related, they refer to different concepts:

A *path* describes where an object will travel in space and can be represented as a sequence of coordinates in a plane or in a three-dimensional space. The focus is on the sequence and shape of movements, such as turns, straight lines, or curves. Importantly, a path does not account for time, making it purely geometric.

On the other hand, a *trajectory* not only describes a path but also adds the dimension of time. It specifies the coordinates of where the vehicle should go and when it should be at each point along the sequence of points. This addition of time implicitly generates a velocity reference at all given points along the path, which is crucial for the precise control and timing needed in autonomous navigation [2].

In this thesis, it is assumed that the reference provided to the controller is a trajectory, meaning a point on the XY-plane that changes over time.

Chapter 6

Model Predictive Control - Implementation

This chapter explains the design and implementation of the MPC. This includes how the dynamic model in the MPC is derived, and the logic behind the MPC. It should be noted that the dynamic model developed in this chapter is fully independent from the model derived in Chapter 3. As outlined in Figure 6.1, this chapter describes the contents of the MPC, and focuses on the third and fourth objective, which states:

3. **Development of a Model Predictive Controller (MPC):** Develop an MPC to be tested in the simulator. The controller should be based on measurements from sensor data, which resembles the information that can be gathered from the physical Lone Wolf, such as GPS- and IMU data.

4. **Implement the control algorithm into the Lone Wolf system:** The developed control algorithm has to be implemented into the current system- and software architecture of the current Lone Wolf project so that it can easily be used in future projects.

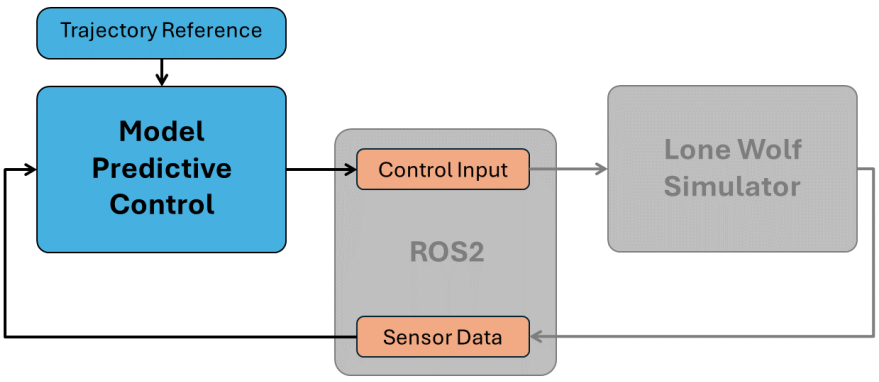


Figure 6.1: Project flowchart: MPC

6.1 MPC implementation

Based on the theoretical framework presented in section 5, the following sections will describe how the MPC has been implemented in this project. Figure 6.2 illustrates the MPC algorithm implemented.

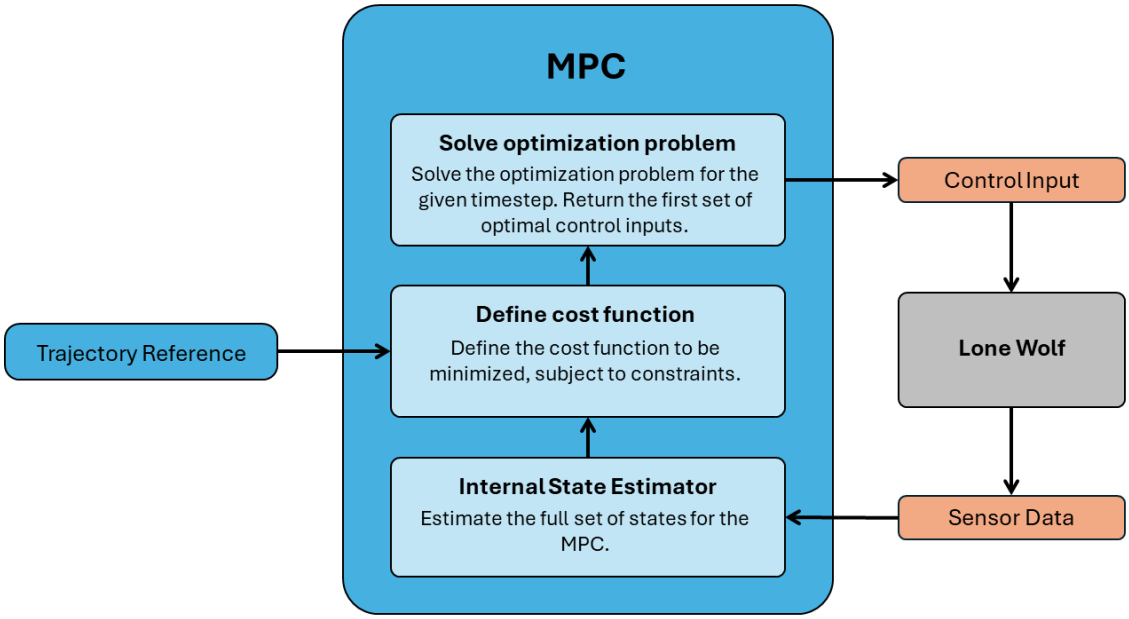


Figure 6.2: MPC flowchart

6.2 Software implementation - Python

The MPC was developed in Python. This choice was made because of the easy syntax compared to some other languages, like C++, and because it has many highly relevant MPC and optimization libraries easily available.

The implementation of the MPC has been done in two separate files: `MPC.py` and `MPC_node.py`. The file `MPC.py` contain the MPC class, and `MPC_node.py` creates the MPC ROS2 node.

The MPC was written in an object oriented style, which allows for easy use in the MPC ROS2 node, as can be seen in Listing 6.1. After line 6, the MPC is ready for use.

```python
from MPC import MPC_Controller

class mpc_ros_controller(Node):
    def __init__(self):
        ...  # Code omitted for brevity
        self.mpc = MPC_Controller()
```

Listing 6.1: Python code showing how the MPC object is created

6.3 Deriving the dynamic model for the MPC

As stated in Section 5.2, a discrete dynamic model is required for predicting the states for the optimization problem. First, a continous-time dynamic model must be derived. The dynamic model implemented in the MPC attempts to describe the dynamics of the ATV in the XY-plane. This 2-dimensional approach was taken to keep the computational costs down. The kinematic bicycle model was implemented, with an assumption of no slip.[39]. The kinematic bicycle model is shown in Figure 6.3. The physical states that describe the dynamics are the inertial coordinates X- and Y, the velocity, v, and the yaw-angle, ψ. In addition, there have been implemented three extra states, F, δ, and $\dot{\delta}$, to describe the dynamics of the engine and steering. The three extra states are internal to the MPC logic, and are calculated based on the model. Inspiration for deriving the equations are drawn from these sources: [4, 10, 39].

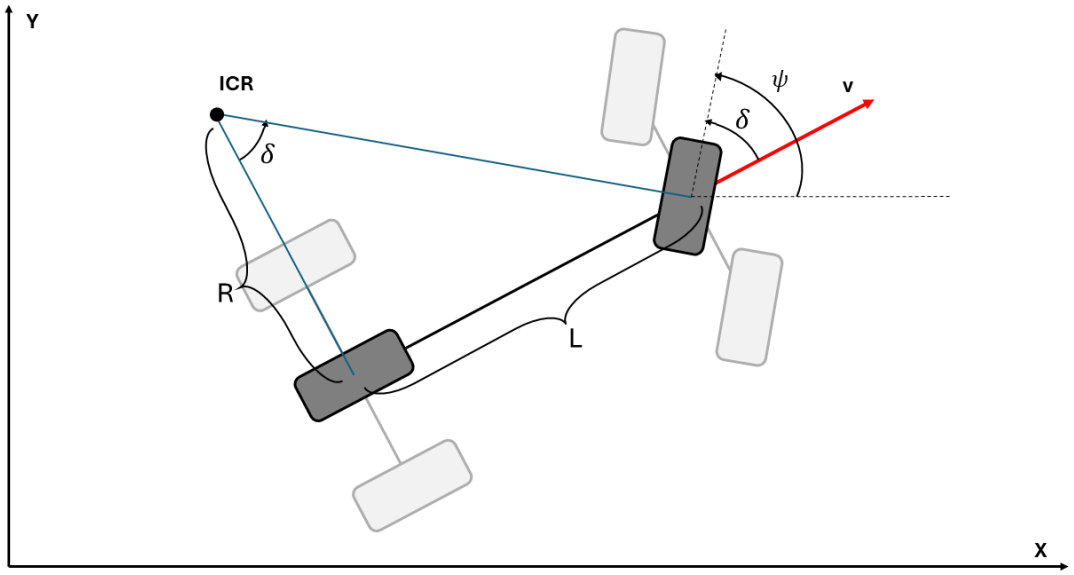


Figure 6.3: Kinematic bicycle model with the assumption of no slip.

Physical states

Equations 6.1 and 6.2 describe velocities in the X- and Y-direction by decomposing the general velocity into their respective directional components.

$$\dot{X} = v \cdot \cos(\psi) \tag{6.1}$$

$$\dot{Y} = v \cdot \sin(\psi) \tag{6.2}$$

Equation 6.4 is based on Newton's second law of motion,

$$\sum F = ma, \tag{6.3}$$

where $\sum F$ is the sum of the forces, m is the mass, and a is the acceleration. The sum of the forces includes the force from the engine, F_e, in the positive direction of travel, and the force from the engine brake and other resistive forces, F_r, in the opposite direction of F_e. The force F_r was added to give more realistic dynamics to the acceleration, and after testing it was kept because of good results. Finally, the equation can be states as

$$\dot{v} = a = \frac{\sum F}{m} = \frac{1}{m}\big(F - F_r \cdot v\big), \tag{6.4}$$

to explicitly describe the velocity-dynamics.

Using the kinematic bicycle model with the assumption of no-slip, shown in figure 6.3, the dynamic equation describing the yaw-angle can be found.

The relationship between linear- and angular velocity, v and $\dot{\psi}$ respectively, is such that

$$v = \dot{\psi} \cdot R, \tag{6.5}$$

where R is the turn radius. The angular velocity, $\dot{\psi}$, is in this case around the Instantaneous Center of Rotation (ICR). The rotation around the ICR is the same as the rotation around the z-axis of the ATV.

From inspecting Figure 6.3, the trigonometric relationship between the length of the wheelbase, L, and the turn radius, R, can be used to find an equation for the angle, δ, as

$$\tan(\delta) = \frac{L}{R}. \tag{6.6}$$

Re-arranging and combining Equations 6.5 and 6.6, the equation for $\dot{\psi}$ can be found as shown in equation 6.7.

$$v = \dot{\psi} \cdot R$$

$$R = \frac{v}{\dot{\psi}}$$

$$\tan(\delta) = \frac{L}{R} = \frac{L \cdot \dot{\psi}}{v}$$

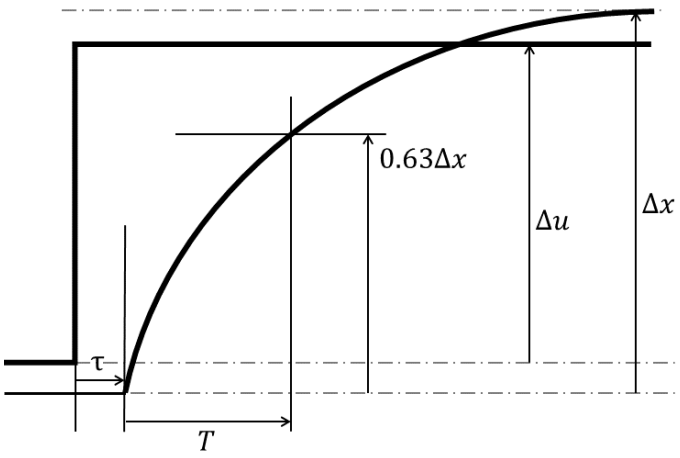
$$\dot{\psi} = \frac{v}{L} \cdot \tan(\delta) \tag{6.7}$$

Equations 6.1, 6.2, 6.4, and 6.7 make up the physical states of the dynamic model. This is summarized in Equation 6.8.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\psi) \\ v \cdot \sin(\psi) \\ \frac{v}{L} \cdot \tan(\delta) \\ \frac{1}{m}(F_e - F_r \cdot v) \end{bmatrix} \tag{6.8}$$

Internal states

As mentioned previously, the states F_e, δ, and $\dot{\delta}$ were included to describe the dynamics of the engine and steering. These states are calculated and kept track of internally in the control algorithm. This is discussed further in Section 6.7. The state F_e describes the



$$h_{ux}(s) = \frac{x(s)}{u(s)} = \frac{K}{1+Ts} \cdot e^{-\tau s}$$

$$K = \frac{\Delta x}{\Delta u}$$

Figure 6.4: Step response of first order low pass filter [5].

force from the motor, and is estimated as a first-order low-pass filter. The results proved to be satisfying with this estimation, so the estimation was kept.

A low-pass filter can be implemented in the form

$$H(s) = \frac{K}{\tau s + 1}, \tag{6.9}$$

where the gain, K, represents the input-to-output gain and the time-constant τ, represent how long it takes for the output value to reach 63% of its maximum output value [41]. The response shape of a first-order low-pass filter, as well as a visual description of the time-constant can be seen in Figure 6.4.

Equation 6.10 explains that the dynamics from the input signal $U_1(s)$ to the output $F_e(s)$ in the is described by $H(s)$ in the s-domain.

$$\frac{F_e(s)}{U_1(s)} = H(s) \tag{6.10}$$

$$F_e(s) \cdot \tau s + F_e(s) = K \cdot U_1(s) \tag{6.11}$$

Performing the inverse Laplace transform results in the time-domain equation for F_e, shown in equation 6.12.

$$\dot{F} \cdot \tau + F = K \cdot u_1 \tag{6.12}$$

Solving for $\dot{F}$ gives the final equation for the motor dynamics

$$\dot{F} = \frac{1}{\tau} \cdot (-F + u_1 \cdot T_{gain}), \tag{6.13}$$

where the gain, K, is changed to T_{gain} to be easily distinguishable.

Finally, the dynamics from the steering input to the steering angle the δ, are estimated as a second-order filter, which uses both δ and $\dot{\delta}$. A servo act like a rate-limiter to a target state. To implement a rate-limiter, a lot of if-statements are required. A second-order low-pass filter is easier to implement as an ODE and provides dynamics that behaves similarly enough to a rate-limiter, as illustrated in Figure 6.5.

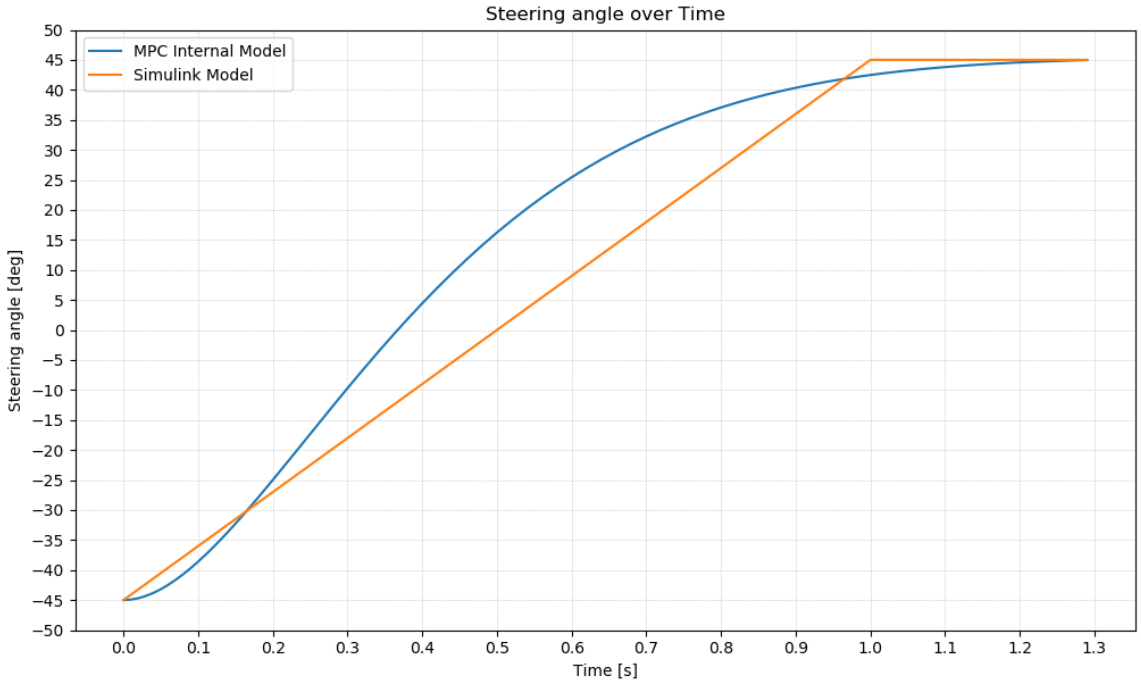


Figure 6.5: Comparison: Integrator and second order filter

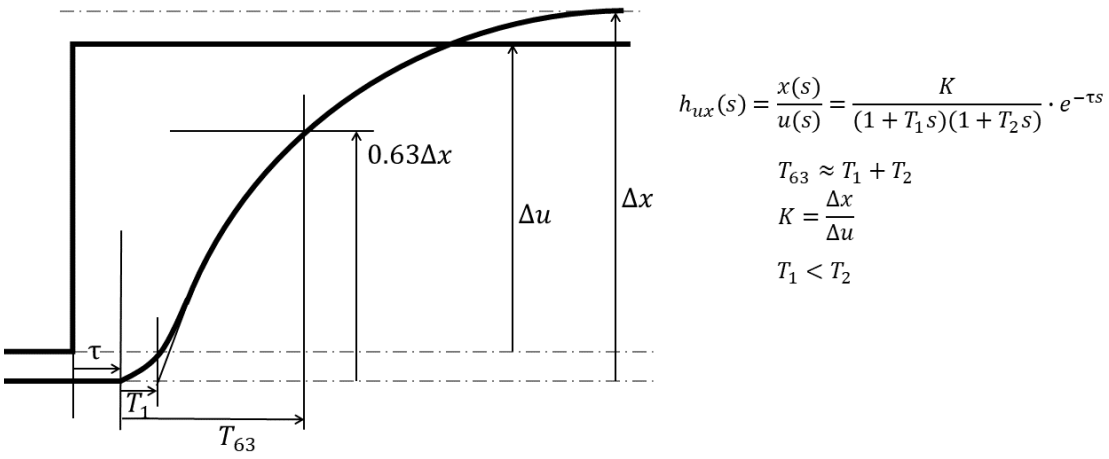


Figure 6.6: Two real pole second order low-pass filter [5].

$$H(s) = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)}, \tag{6.14}$$

where the "1" in the numerator signifies a unit gain and no time-delay. The τ_1 and τ_2 in the denominator refers to the time constants, as described in Figure 6.6. This implementation is based on two distinct real poles. Rewriting from τ_1 and τ_2 to ζ and ω,

$$H(s) = \frac{1}{(\tau_1 s + 1)(\tau_2 s + 1)} = \frac{1}{\frac{1}{\omega^2} \cdot s^2 + \frac{2\zeta}{\omega} \cdot s + 1}, \tag{6.15}$$

gives the more general expression. This form grants more control in tuning the model, as this enables overshooting the target state by introducing the opportunity of complex-conjugate poles. In this case, ζ represents the damping of the response, and ω describes the frequency of the response.

Further, the transfer-function can be applied to the relationship between the steering input and the steering angle

$$\frac{\delta(s)}{U_2(s)} = H(s), \tag{6.16}$$

and expanded to

$$\frac{1}{\omega^2}\delta(s)s^2 + \frac{2\zeta}{\omega}\delta(s)s + \delta(s) = U_2(s), \tag{6.17}$$

resulting in the s-domain dynamics. Performing the inverse Laplace transform to this system results in the time-domain equations given in equation

$$\frac{1}{\omega^2}\ddot{\delta} + \frac{2\zeta}{\omega}\dot{\delta} + \delta = u_2. \tag{6.18}$$

Solving for $\ddot{\delta}$ gives the final equation in the system

$$\ddot{\delta} = -2\zeta\omega\dot{\delta} - \omega^2(\delta - u_2). \tag{6.19}$$

6.4 Brakes in the MPC model

As is apparent, there are no control input for brakes included in the model. The brakes on the physical Lone Wolf has a harsh on/off characteristic, due to how the brake control is realized. Therefore used primarily for full-stops. Additionally, the engine brakes and other resistive forces are often seen as sufficiently good for the maneuvers Lone Wolf will undertake. The knowledge of the brake characteristics and the lack of data on brake dynamics led to the decision to omit the brakes in the model.

6.5 MPC model

The complete dynamic model is made up of equations 6.1, 6.2, 6.4, 6.7, 6.13, and 6.19. These are recapped in equation 6.20. A table containing a description of the parameters can be found in Table 6.1.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \dot{v} \\ \dot{F}_e \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\psi) \\ v \cdot \sin(\psi) \\ \frac{v}{L} \cdot \tan(\delta) \\ \frac{1}{m}(F_e - F_r \cdot v) \\ \frac{1}{\tau}(u_1 \cdot T_{\text{gain}} - F_e) \\ \dot{\delta} \\ -2\zeta\omega\dot{\delta} - \omega^2(\delta - u_2) \end{bmatrix} \tag{6.20}$$

where:

- X, Y are the positions in the inertial reference frame
- $\dot{X}, \dot{Y}$ are velocities in their respective directions in the inertial reference frame
- ψ is the angle of the vehicle relative to the static X-axis
- δ is the relative angle of the wheels compared to ψ
- $\dot{\delta}$ is the rate of change of the steering
- F_e is the force output from the engine
- u_1 is the control input for throttle
- u_2 is the control input for steering
- L is the length of wheelbase
- m is the mass
- T_{gain} is the scaling factor from control signal to force
- F_r is the friction parameter encapsulating rolling resistance and engine braking
- ζ is the damping factor in the second-order low-pass filter used in steering dynamics
- ω is the frequency of the second-order low-pass filter
- τ is the time constant in the first-order low-pass filter used in the engine dynamics

6.6 Discrete dynamic model

Recall that the constraints related to the dynamic model are based on discrete time, therefore the model has to be discretized. In Appendix C, three approaches to discretizing dynamics systems has been explored: an exact analytical approach, forward Euler, and 2nd order Runge-Kutta. Based on those findings, Forward Euler was chosen as the method for discretizing the model, because of the simplicity of its implementation, and satisfying results during the tests [42].

When discretizing the model for the MPC, the control inputs u_1 and u_2 are not changed between time-steps and thus are denoted as $\bar{u_1}$ and $\bar{u_2}$. The final discrete implementation is shown in Equation 6.21, where dt denotes the time step.

$$\begin{aligned}
X[k+1] &= X[k] + dt \cdot \big(v[k] \cdot \cos(\psi[k])\big) \\
Y[k+1] &= Y[k] + dt \cdot \big(v[k] \cdot \sin(\psi[k])\big) \\
\psi[k+1] &= \psi[k] + dt \cdot \left(\frac{v[k]}{L} \cdot \tan(\delta[k])\right) \\
v[k+1] &= v[k] + dt \cdot \left(\frac{1}{m}(F_e[k] - F_r \cdot v[k])\right) \\
F_e[k+1] &= F_e[k] + dt \cdot \left(\frac{1}{\tau_1}(\bar{u}_1 \cdot T_{\text{gain}} - F[k])\right) \\
\delta[k+1] &= \delta[k] + dt \cdot \dot{\delta}[k] \\
\dot{\delta}[k+1] &= \dot{\delta}[k] + dt \cdot \Big(-2\zeta\omega\dot{\delta}[k] - \omega^2(\delta[k] - \bar{u}_2)\Big)
\end{aligned} \tag{6.21}$$

The set of equations in 6.21 is the discrete set of equations governing the constraints in the MPC,

$$x_{k+1} = g(x_t, u_t), \tag{6.22}$$

as described by Equation 5.2 in Section 5.2.

The value of dt was chosen to be 10% of the desired close loop response time of the fastest dynamics of the system [23]. As the steering angle, δ, takes 1 second to go from $[-\frac{\pi}{4}, \frac{\pi}{4}]$ rad, the value was chosen to be 0.1s.

The code in Listing 6.2 shows parts of the `mpc_generate_step` method, and specifically how the discrete time model has been implemented in the MPC to define the cost function. As seen from line 9 to out, for every iteration over the prediction horizon, `self.N`, the difference between the current state and the reference, and the change in control input are

added to the cost function. Then, as seen in line 16 to 26 the next state is calculated and added to a list for use in the next iteration. In this manner, the optimization problem is constrained to the states deemed possible by the discrete dynamic model.

```
1  def mpc_generate_step(self, physical_states, reference_list_x,
2                                                 reference_list_y):
3      #Set initial states
4      initial_states = self.internal_state_estimator(physical_states)
5      ...  # Code omitted for brevity
6      cost = 0
7      g = []
8      ...  # Code omitted for brevity
9      for k in range(self.N):
10         ...  # Code omitted for brevity
11         state_diff = state - target_state
12         cost += 0.5*ca.mtimes([state_diff.T, self.Q, state_diff]) +
13                 0.5*ca.mtimes([delta_control.T, self.R, delta_control])
14
15         # #discrete dynamic model
16         x_next = self.x[k] + self.dt* self.v[k]*ca.cos(self.psi[k])
17         y_next = self.y[k] + self.dt * self.v[k]*ca.sin(self.psi[k])
18         psi_next = self.psi[k] + self.dt* self.v[k] /
19                               self.L * ca.tan(self.delta[k])
20         v_next = self.v[k] + self.dt * (self.F[k]/self.M -
21                               self.D/self.M*self.v[k])
22         ...  # Code omitted for brevity
23         if k < self.N-1:
24             g.append(self.x[k+1] - x_next)
25             g.append(self.y[k+1] - y_next)
26             ...  # Code omitted for brevity
```

Listing 6.2: Python code showing the discrete dynamic model implemented

6.7 Internal state estimator

The states F, δ, and $\dot{\delta}$ are internal to the MPC logic, and they are implemented in the MPC class. The states, X, Y, ψ, and v, are physical states obtained from the sensors, but all seven states are used for calculating the predicted states. Therefore the four states, including previous control inputs, are passed through the state estimation function. This uses the same equations as the dynamic model, restated in equation 6.23, to update the internal states, and then return the full set of states for use in the model implemented in the MPC.

$$\begin{aligned} F[k+1] &= F[k] + dt \cdot \left(\frac{1}{\tau_1} (\bar{u_1} \cdot T_{\text{gain}} - F[k]) \right) \\ \delta[k+1] &= \delta[k] + dt \cdot \dot{\delta}[k] \\ \dot{\delta}[k+1] &= \dot{\delta}[k] + dt \cdot \left(-2\zeta\omega\dot{\delta}[k] - \omega^2(\delta[k] - \bar{u_2}) \right) \end{aligned} \tag{6.23}$$

Line 4 of the code Figure 6.2 uses the internal state estimator. Every time the `mpc_generate_step` method is called, which is one every 0.1 seconds, the internal states must be re-calculated. The four physical states X, Y, ψ, and v are read from the `/vectornav` ROS2 topics, and passed through the `internal_state_estimator` method, as shown in the code in Listion 6.3. The method takes in the physical states, and returns the full set of states used in the dynamic model. The full set of states functions as the initial conditions for the cost function calculations.

```
1  def internal_state_estimator(self, physical_states):
2      x, y, psi, v = physical_states
3
4      self.F_state += self.dt *( 1/self.tau * (-self.F_state +
5                      self.throttle_state * self.throttle_gain))
6      self.delta_state += self.dt*(self.delta_dot_state)
7      self.delta_dot_state += self.dt*(-2*self.zeta*self.omega
                            *self.delta_dot_state - self.omega**2 *
8                          (self.delta_state - self.steering_state))
9
10     return [x, y, psi, v, self.F_state, self.delta_state,
11                                      self.delta_dot_state]
```

Listing 6.3: Python code showing internal state estimator implemented

6.8 Physical constraints on the system

Recall that the physical limitations on the system, i.e. the constraints for the optimization problem, are given as

$$x^{\text{low}} \leq x_t \leq x^{\text{high}} \tag{6.24}$$

$$u^{\text{low}} \leq u_t \leq u^{\text{high}}, \tag{6.25}$$

as described in section 5.2.

These are implemented as

$$\begin{aligned} -\frac{\pi}{4} \leq\ & \delta\ \leq \frac{\pi}{4} \\ 0 \leq\ & u_1\ \leq 100 \\ -\frac{\pi}{4} \leq\ & u_2\ \leq \frac{\pi}{4}, \end{aligned} \tag{6.26}$$

which describes the physical limitations of the steering angle, δ, throttle input, u_1, and steering input, u_2.

6.9 Optimization problem

The optimizer solves the optimization problem

$$\min_{z \in \mathbb{R}^n} f(z) = \sum_{k=0}^{N-1} \frac{1}{2} \left[\left(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{ref}} \right)^\top Q_{t+1} \left(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{ref}} \right) + \Delta \mathbf{u}_k^\top R_{\Delta t}\, \Delta \mathbf{u}_k \right], \tag{6.27}$$

subject to the constraints given by the discrete dynamic model in equation 6.21, and the physical constraints described in section 6.8. The optimization problem must be solved for every time step in the MPC. The calculations are done by using the Python-library *IPOPT*, which is short for Interior-Point Optimizer [45]. Interior-point methods are shown to be successful for non-linear optimization problems and are considered powerful algorithms for large-scale non-linear programming [26]. When tested, IPOPT gave satisfactory good results, both in terms of computation-speed and reference tracking. The reference tracking is discussed more in chapter 7. Details on how this method works is beyond the scope of this thesis, however, the reader may explore more about these optimization techniques in these references: [44, 45, 26].

When all the cost function and the constraints have been correctly defined, the code to solve the optimization problem is compact, as can be seen in line 5 of the code in Listing

6.4. The variable `sol` now contains the optimal control inputs. In line 1, the IPOPT options has been declared, which contains a max CPU timer. This variable restricts the amount of time the solver is allowed to use searching for an optimal solution. The time of 0.07 seconds was chosen because it is 70% of the time-step, dt, which is 0.1 second. This allows for overhead, and ensures that there will always be a solution ready at each time-step.

```
1 ipopt_options = {'print_level': 0, 'max_cpu_time': 0.07}
2 ...  # Code omitted for brevity
3 solver = ca.nlpsol('solver', 'ipopt', nlp, {'ipopt': ipopt_options})
4 ...  # Code omitted for brevity
5 sol = solver(x0=initial_conditions, lbx=self.lbx, ubx=self.ubx, lbg=0, ubg
    =0)
```

Listing 6.4: Python code showing internal state estimawtor implemented

As mentioned earlier, the cost function, $f(z)$, is defined over the prediction horizon. The prediction horizon, `self.N = 30`, because of computational speed. This value ensures that most of the optimal solution is found within the CPU time-limit.

6.10 Tuning the MPC

The behavior of the MPC can be changed, or tuned, in two different ways: by changing the cost function with the Q and $R_{\Delta t}$ matrices, and/or by changing the parameters of the dynamic model. As soon as the dynamic model is sufficiently accurate, only the Q and $R_{\Delta t}$ matrices should be used to change the behavior of Lone Wolf.

6.11 Tuning of the cost function

The matrices implemented in the solution are the Q_t and $R_{\Delta t}$ matrices shown in equation 6.28.

$$Q_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad R_{\Delta t} = \begin{bmatrix} 0.02 & 0 \\ 0 & 8 \end{bmatrix} \tag{6.28}$$

The Q_t configuration focuses exclusively on position compared to the reference. This is

done because of the main concern of following the planned path.

The $R_{\Delta t}$ matrix punishes changes in the throttle- and steering inputs. These have very different scales, as their values are significantly different in range of possible input values: $u_1 \in [0, 100]$ and $u_2 \in [-\frac{\pi}{4}, \frac{\pi}{4}]$. Punishing the change in control restrains big changes in control inputs, improving longevity of actuators in the physical system. A balance must be made, because overpunishing the control inputs will lead to poor path-following.

As discussed in the Chapter 7, this combination of tuning led to good results.

6.12 Tuning of the Model

To achieve optimal performance, the model used in Model Predictive Control (MPC) must closely resemble the actual dynamics of the system it controls.

The MPC model incorporates two fixed constants: The length of the wheelbase, L, and the mass, m. Additionally, the model includes five adjustable parameters, of which three are crucial for controlling velocity: T_{gain}, F_r, and τ. The combination of T_{gain} and F_r influences the steady state value as well as the rise and fall shape of the velocity curve, while τ affects only the rise and fall shape.

Regarding steering dynamics, as referenced in Equation 6.19, the steering dynamics are modeled using a second-order filter characterized by two parameters, ζ and ω. These are chosen to mimic a rate limiter with a slope equivalent to the servo's steering rate.

For detailed guidance on selecting these parameters based on empirical data, the reader is referred to Appendix B.

6.13 Distinguishing trajectory and path following

Recall from Section 5.6 where the difference between a path and a trajectory was stated. To clarify, a path consists of a sequence of spatial points, whereas a trajectory is similar but includes specified times for reaching each point. For obstacle avoidance, closely following the designated path of the trajectory reference can be more crucial than adhering strictly to the trajectory, especially when the intended trajectory demands physically impractical accelerations or velocities. To address this, a "Closest point" mechanism has

been implemented. With this feature, the initial point of the trajectory is always set to the closest point on the path, slightly prioritizing points that are closer in time to the real-time reference, this is to ensure that Lone Wolf stays on the correct trajectory if the trajectory crosses itself. This approach effectively transforms the system into a path follower rather than a trajectory follower if the trajectory is unfeasible.

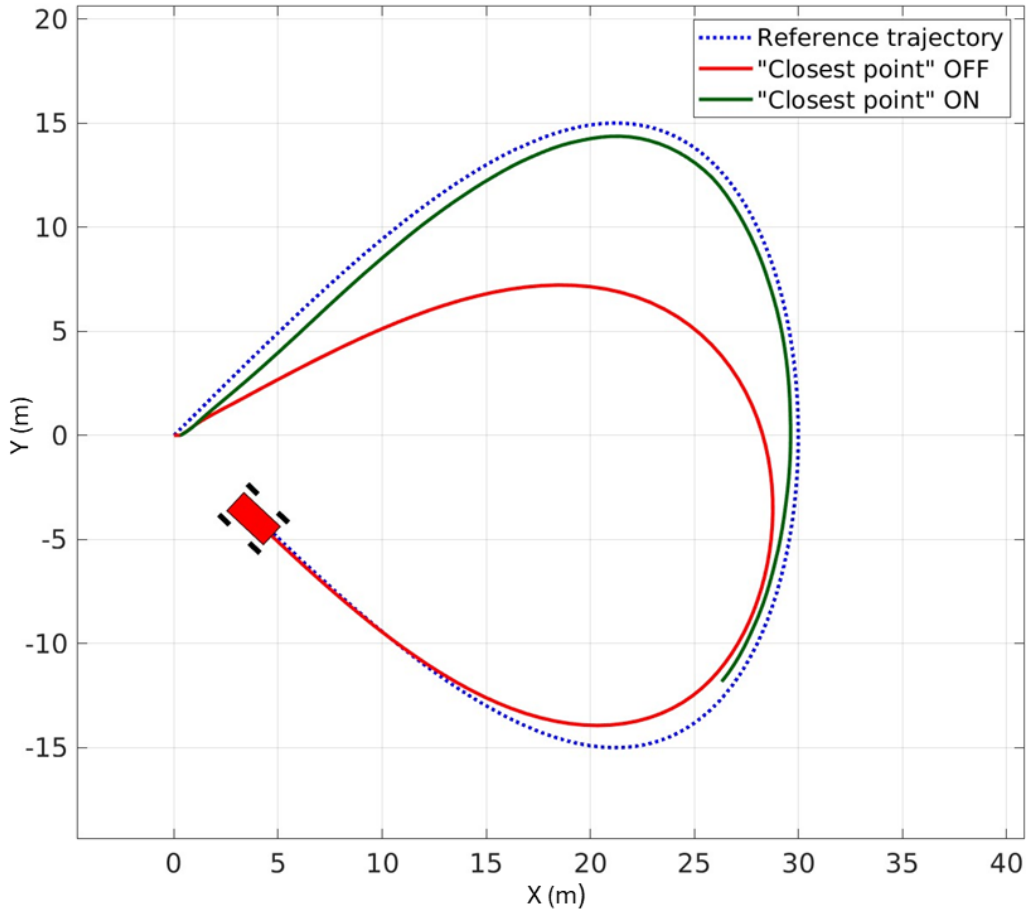


Figure 6.7: "Closest point" mechanism

Observe in Figure 6.7 how the MPC behaves differently based on the "Closest point" setting. When this feature is disabled, the MPC minimizes error over time by cutting corners, it continually aims for the furthest tip of the dynamically drawn reference point. In contrast, when "Closest point" is enabled, the MPC targets the nearest previously established reference point. This approach ensures adherence to a more feasible path, rather than pursuing an unfeasible trajectory.

6.14 Integrating the controller

The MPC has been integrated into the existing software architecture by utilizing the existing ROS2 interface on Lone Wolf, as illustrated in figure 6.8. This means that for every iteration, the physical states X, Y, ψ, and v are read from the relevant sensors via the `/vectornav` ROS2 topics. When these values have been processed by the MPC and the optimal control input has been calculated, the first set of inputs are published to their respective ROS2 topics. The throttle input is published to `lw_portenta_throttle` and the steering input is published to `/lw_portenta_steering`. As mentioned in Section 2.3, the choice to read and send data this way was made to match the current system on the physical Lone Wolf, as per objective 4 stated in this chapter's introduction.

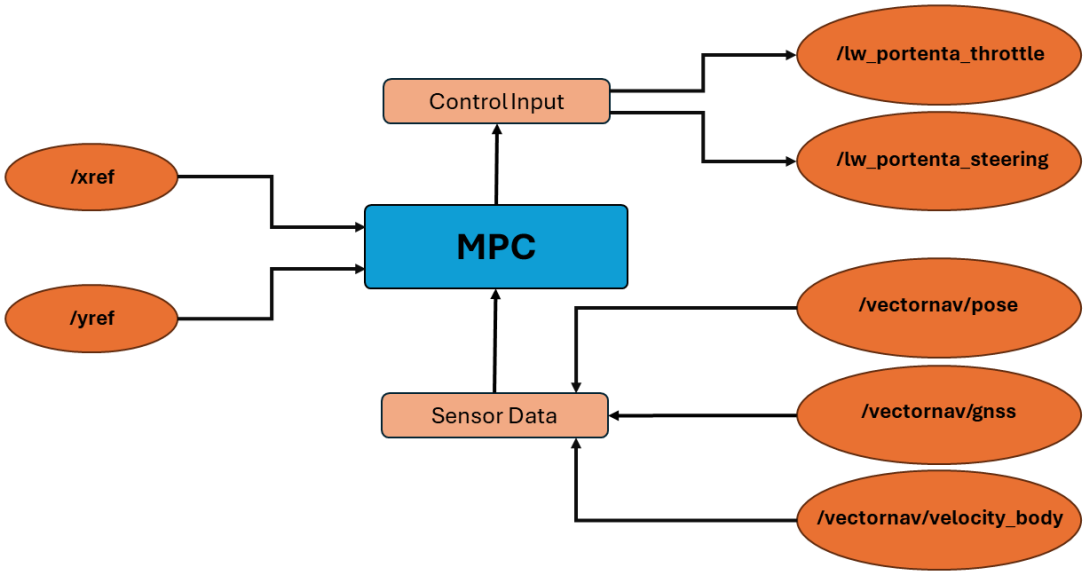


Figure 6.8: MPC: ROS2 integration

The code in Listing 6.5 shows a shortened version of the code implemented. In line 4, the MPC generates the control inputs. This is done by using the MPC class method `mpc_generate_step`, which takes in the physical states retrieved from the `/vectornav` topics, and the X- and Y references. Afterward the data is converted to the correct format, before it is published to the `/lw_portenta_throttle` topic as seen in line 11.

```python
def generate_and_publish_inputs(self):
        ...  # Code omitted for brevity
        u_t = self.mpc.mpc_generate_step(physical_states, xref_n, yref_n)
        u_throttle = float(u_t[0])
        ...  # Code omitted for brevity
        msg_u1 = Int64()
        ...  # Code omitted for brevity
        msg_u1.data = int(u_throttle)
        ...  # Code omitted for brevity
        self.u1_publish_.publish(msg_u1)
```

Listing 6.5: Python code publishing control inputs

6.15 MPC summary

The overview from the start of the chapter is shown in Figure 6.15, which outlines the implementation of the MPC.

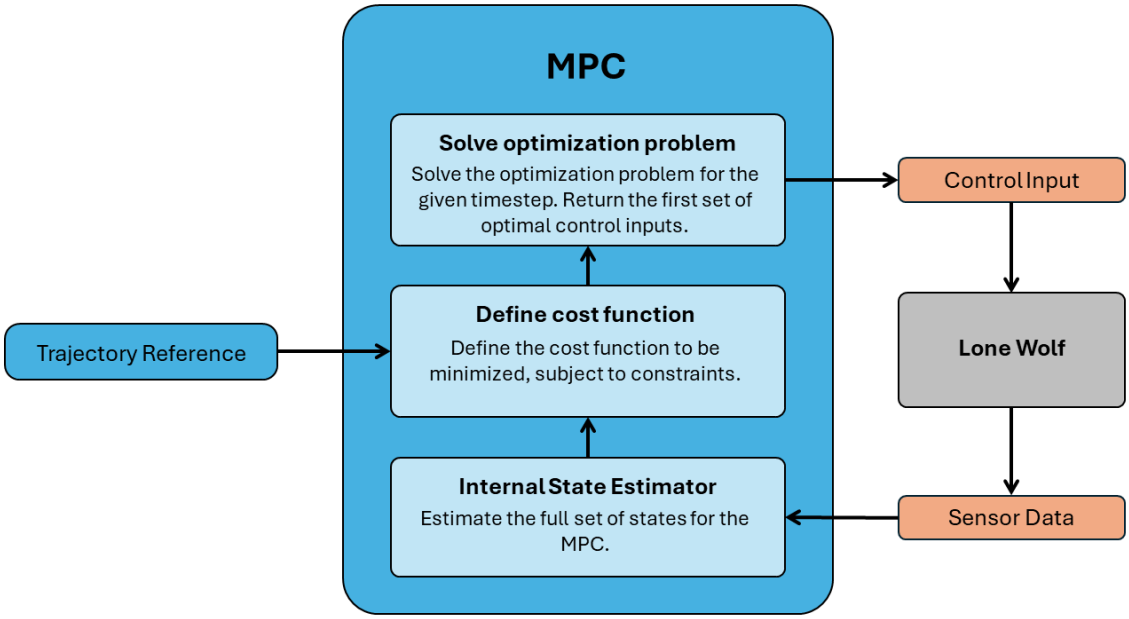


Figure 6.9: MPC flowchart - revisited

The MPC is integrated in the software architecture as described in Section 6.14. For every time-step, it reads the current states from the relevant ROS2 topics. This data is fed through the Internal State Estimator, described in Section 6.7, which returns the full set of states. The full set of states are used in the discrete dynamic model in Equation 6.21, which describes the constraints on possible outcomes based on the given control inputs over the given prediction horizon. This, combined with the imposed physical constraints described in Equation 6.26, are used by IPOPT, the non-linear solver used to determine the optimal sequence of control inputs. Only the first set of control inputs are applied to the system, by publishing these to the their respective ROS2 topics. This process is repeated for every time-step, which constitutes the non-linear MPC implemented.

Parameters used in the dynamic model for the MPC

Parameter	Description
X, Y	Positions in the inertial reference frame
$\dot{X}, \dot{Y}$	Velocities in their respective directions in the inertial reference frame
ψ	Angle of the vehicle relative to the static X-axis
δ	Relative angle of the wheels compared to ψ
$\dot{\delta}$	Rate of change of the steering
F_e	Force output from the engine
u_1	Control input for throttle
u_2	Control input for steering
L	Length of the wheelbase
m	Mass
T_{gain}	Scaling factor from control signal to force
F_r	Friction parameter encapsulating rolling resistance and engine braking
ζ	Damping factor in the second-order low-pass filter used in steering dynamics
ω	Frequency of the second-order low-pass filter
τ	Time constant in the first-order low-pass filter used in the engine dynamics

Table 6.1: Table of variables and parameters used in the dynamic model

Chapter 7

Results

This chapter presents the current functionality of the simulator and the Model Predictive Controller (MPC). It includes demonstrations of the simulator's behavior and how the MPC responds to various scenarios, such as different driving surfaces, modeling inaccuracies, infeasible trajectories, and a variety of tracks.

The different tests and results aims to show how different parameter values and changes in tuning of the models affect the simulator and the MPC developed in the thesis.

7.1 Test 1: Pacejka parameters

Purpose: Test how different values in the Pacejka parameters, described in Section 3.5.1, impact the behavior of the simulated Lone Wolf. The expected behavior is good grip on dry tarmac, some slip on wet tarmac, and a lot of slip on icy conditions. The test was done in an open-loop configuration, meaning that the input signals, δ and throttle, were applied manually and not affected by any controller or reference trajectory.

Conditions:

- Closest point feature set to ON, as detailed in Section 6.13.
- A short steering maneuver applied 13 seconds into the test, as seen in Figure 7.2a.
- Throttle constant at 50%, as seen in Figure 7.2b.
- Three sets of Pacejka parameters were used, as seen in Table 7.1 [28]. Their corresponding Pacejka curves can be seen in Figure 7.1.

Observations: The results from the test can be seen in Figure 7.3, showing the vehicle's travelled path in the XY-plane as well as it's velocity.

- Slower acceleration on icy conditions due to the tires slipping. This also means that Lone Wolf travels a shorter distance before the steering input is applied.
- A loss of velocity during the turn in icy and wet conditions due to tires sliding.
- Negligible loss of velocity during the turn in dry conditions due to the high grip of the tires.
- The vehicle is able to make sharper turns in conditions with better grip.

Conclusion: This test effectively demonstrated that the Pacejka parameters in the simulation influence the simulated Lone Wolf as expected.

Surface\Parameter	**B**	**C**	**D**	**E**
Dry tarmac	10	1.9	1840	0.97
Wet tarmac	6	2.3	1048	1
Ice	4	2	184	1

Table 7.1: Pacejka parameters for different surfaces

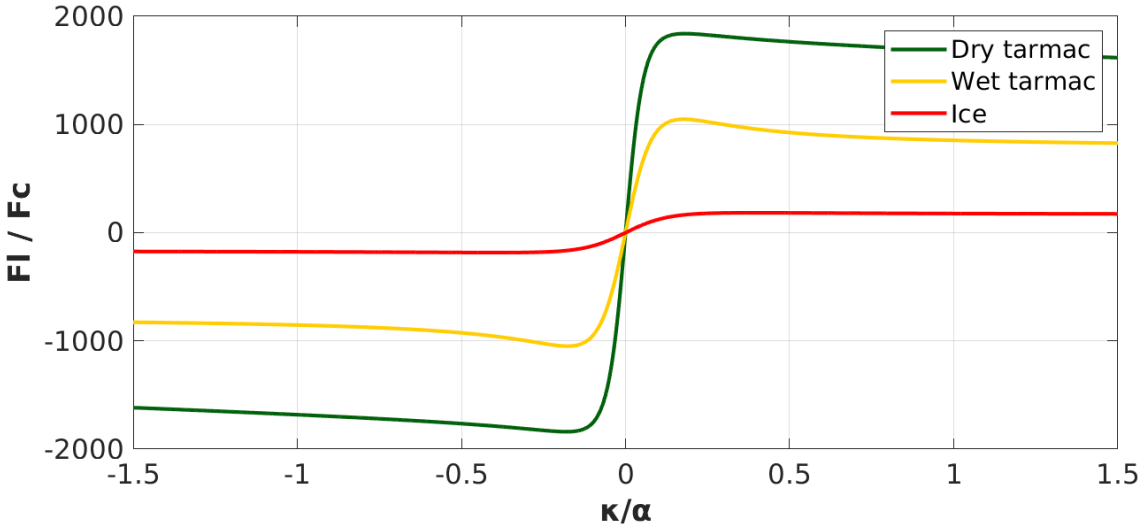
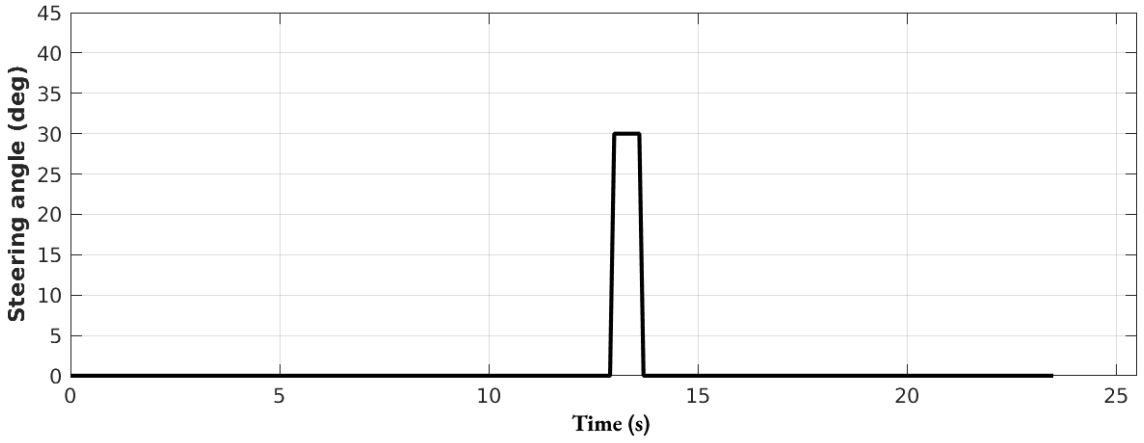
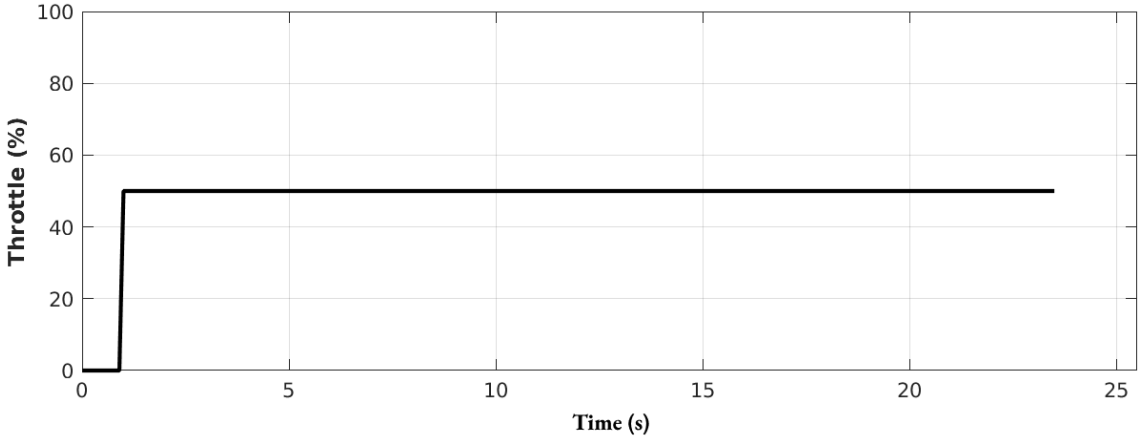


Figure 7.1: Pacejka curves for different surfaces

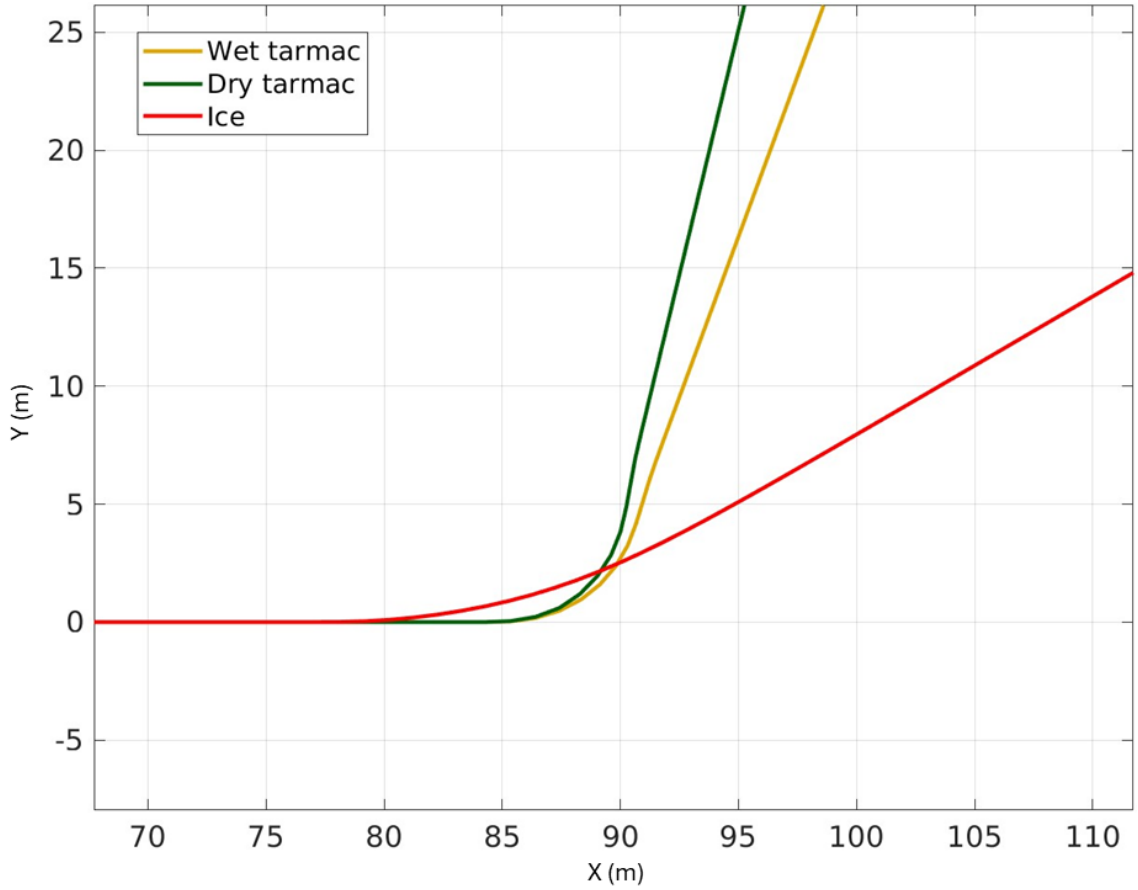


(a) Steering input

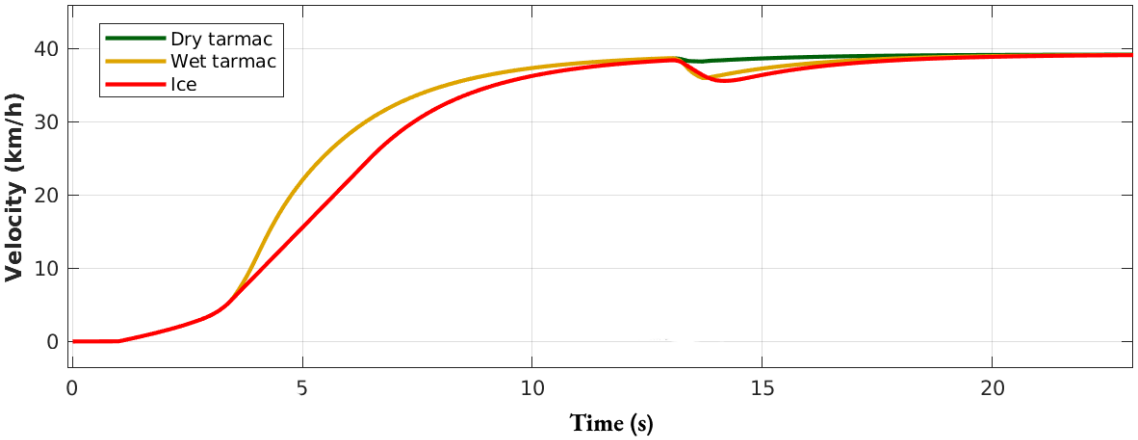


(b) Throttle input

Figure 7.2: Test: Pacejka inputs



(a) Pacejka: XY-graph



(b) Pacejka: Velocity

Figure 7.3: Results: Pacejka

7.2 MPC specific tests

This section details tests designed to evaluate the MPC's performance under various conditions and trajectory references. Comparisons are intended to highlight the impacts of tuning and modeling precision on the MPC's effectiveness. The comparisons are quantitatively assessed using two key metrics: Root Mean Square Error ($RMSE$) and maximum error, $E_{\max}$.

Momentaneous error at simulation iteration i is measured as the distance from the vehicle's center of gravity (CG) to the closest point on the reference trajectory using the formula

$$e_i = \sqrt{(X - X_{\text{ref}})^2 + (Y - Y_{\text{ref}})^2} \ , \tag{7.1}$$

where X and Y represent the vehicle position in the inertial reference frame, and X_{ref} and Y_{ref} represent the closest point on the reference trajectory.

RMSE is calculated using the formula

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{e_i^2}{n}} \ , \tag{7.2}$$

where e_i is the momentaneous error measured at iteration i during the simulation and n is the total amount of momentaneous errors calculated. RMSE gives more weight to larger errors because it squares the error values before averaging them. This means that larger errors have a disproportionately large effect on RMSE, making it sensitive to outliers and large errors. This is chosen as one of the comparison metrics because it is desirable to follow the reference trajectory as closely as possible at all times.

Maximum error simply highlights the maximum deviation from the path during the simulation and can be stated mathematically as

$$E_{\max} = \max_{1 \leq i \leq n} e_i \ , \tag{7.3}$$

where e_i is the momentaneous error measured at iteration i during the simulation and n is the total amount of momentaneous errors calculated. $E_{\max}$ is chosen as the other comparison metric, as it highlights these large deviations clearly.

This page has been left blank intentionally.

7.2.1 Test 2: Straight Line

Purpose: Test the functionality of the MPC with a basic trajectory reference. The expected behavior is for Lone Wolf to make a quick turn to the left and then follow the trajectory with minimal error and no oscillations.

Conditions:

- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the reference trajectory is set to 20km/h.

For this and all following tests, the initial values for Lone Wolf are

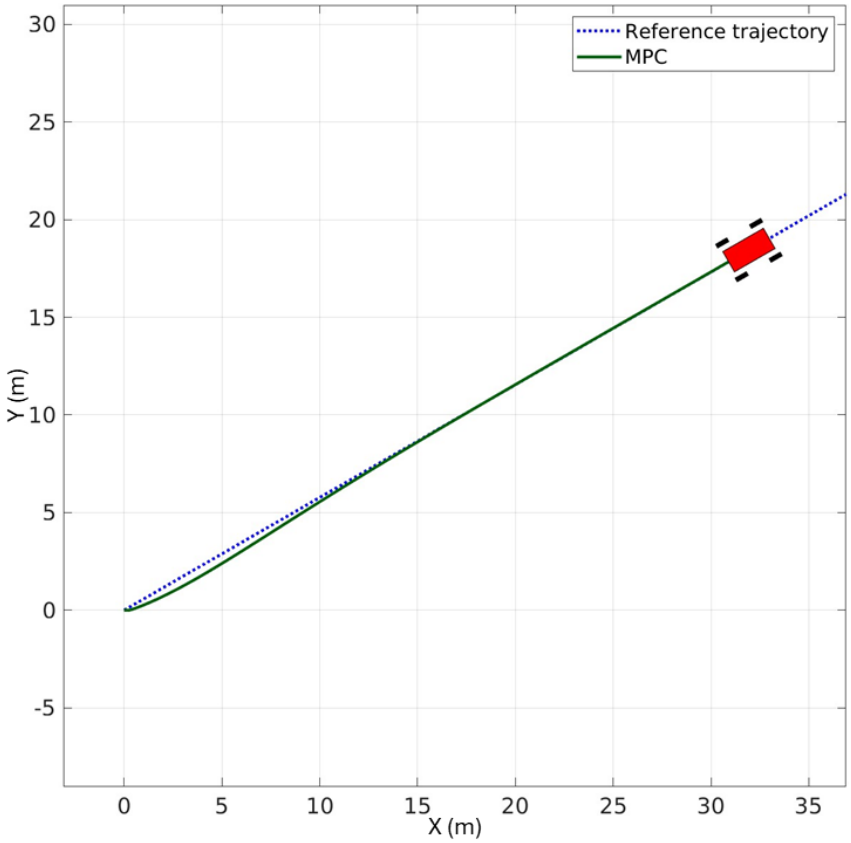
$$\begin{bmatrix} X \\ Y \\ \psi \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{7.4}$$

The closest point feature is also set to *ON*, as detailed in Section 6.13.

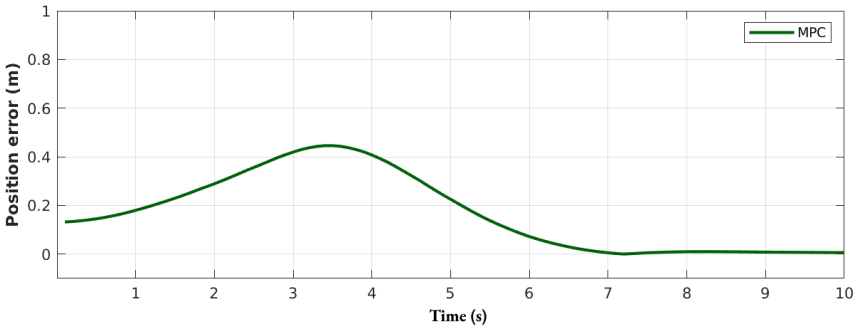
Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.4. Notable observations from these results include:

- The position error growing at the start of the test. This is because the initial heading, ψ, differs from the angle of the trajectory.
- The driven trajectory has no oscillations.

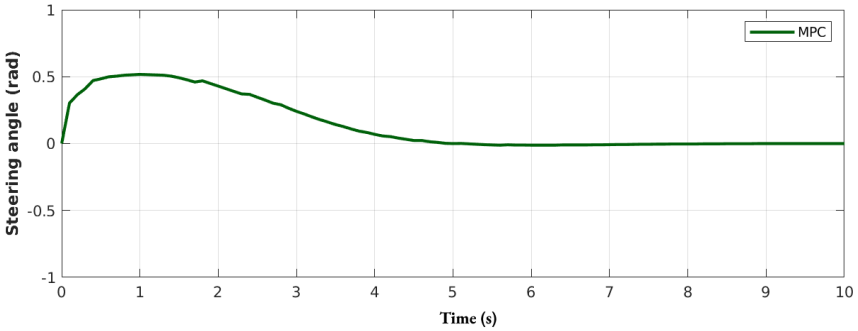
Conclusion: The test confirmed that the MPC effectively adjusts the heading of Lone Wolf and successfully follows this simple trajectory.



(a) Straight line: XY-graph



(b) Straight line: Error



(c) Straight line: Steering angle

Figure 7.4: Results: Straight line

7.2.2 Test 3: 90-degree turn

Purpose: Test the functionality of the MPC with a trajectory reference that is not physically feasible to follow due to the physical limitations of the steering angle. The expected behavior is for the ATV to make a smooth turn with minimal overshoot and no oscillations.

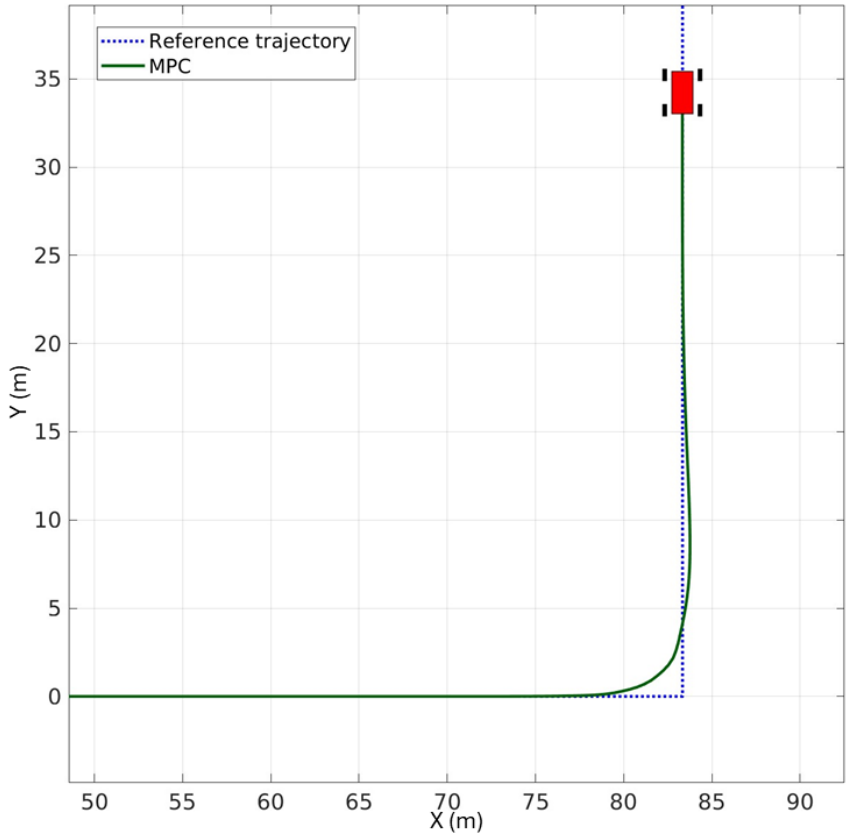
Conditions:

- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the reference trajectory is set to 20km/h.

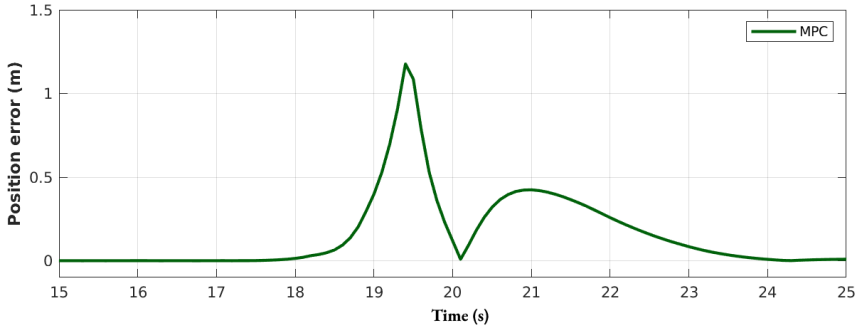
Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.5. Notable observations from these results include:

- Lone Wolf starts to turn before the corner and has minimal overshoot.
- The driven trajectory has no oscillations.

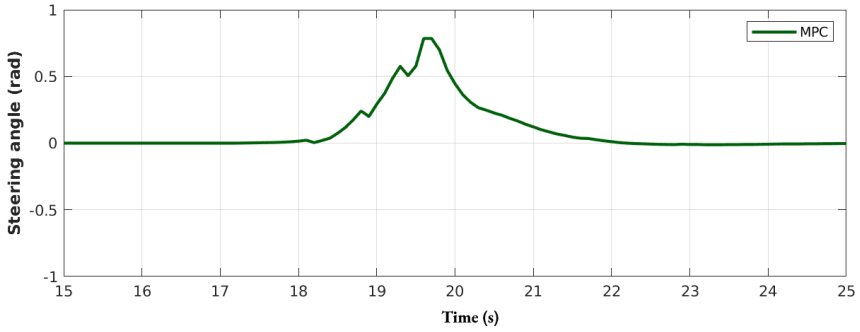
Conclusion: The test confirmed that the MPC effectively makes decisions that minimizes error through physically unfeasible references.



(a) 90-degree turn: XY-graph



(b) 90-degree turn: Error



(c) 90-degree turn: Steering angle

Figure 7.5: Results: 90-degree turn

7.2.3 Test 4: Big Infinity Track

Purpose: Test the functionality of the MPC with a trajectory reference that is smooth and expectedly easy to follow. The expected behavior is for Lone Wolf to closely adhere to the reference trajectory with minimal error.

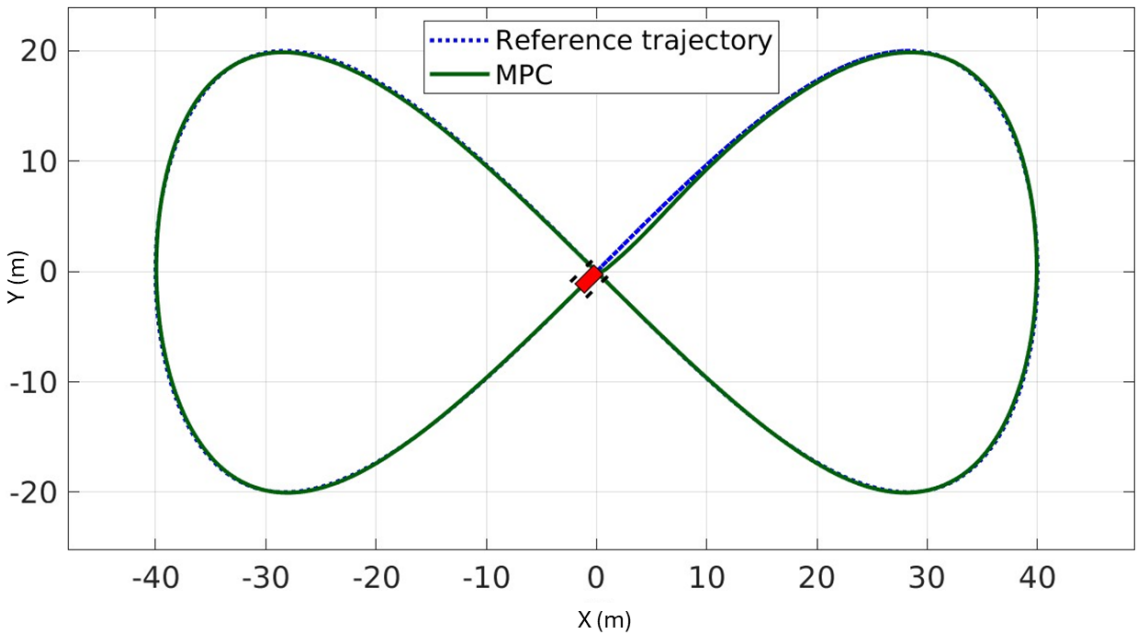
Conditions:

- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the reference trajectory ranges from 15km/h to 35km/h.

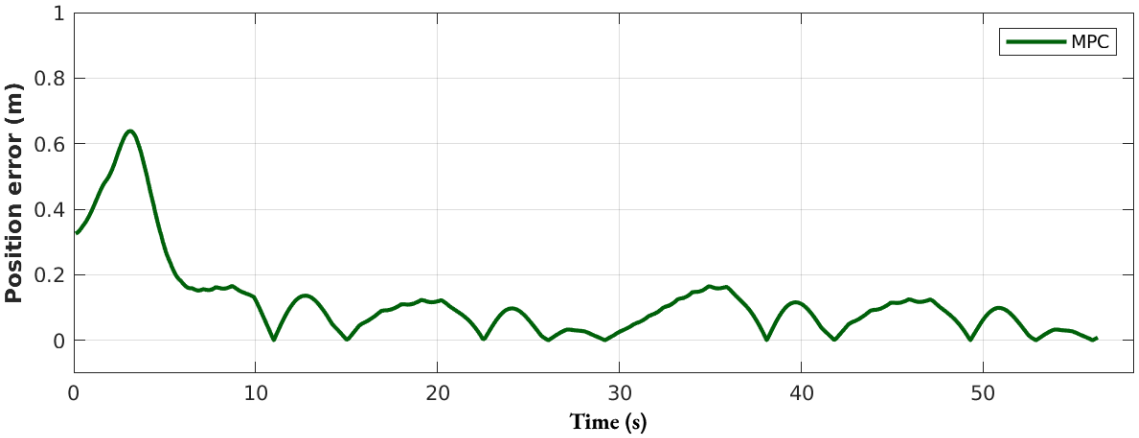
Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.6. Notable observations from these results include:

- The ATV quickly adjusted to the reference trajectory and followed it with little to no error.
- The position error growing at the start of the test. This is because the initial heading, ψ, differs from the initial angle of the trajectory.

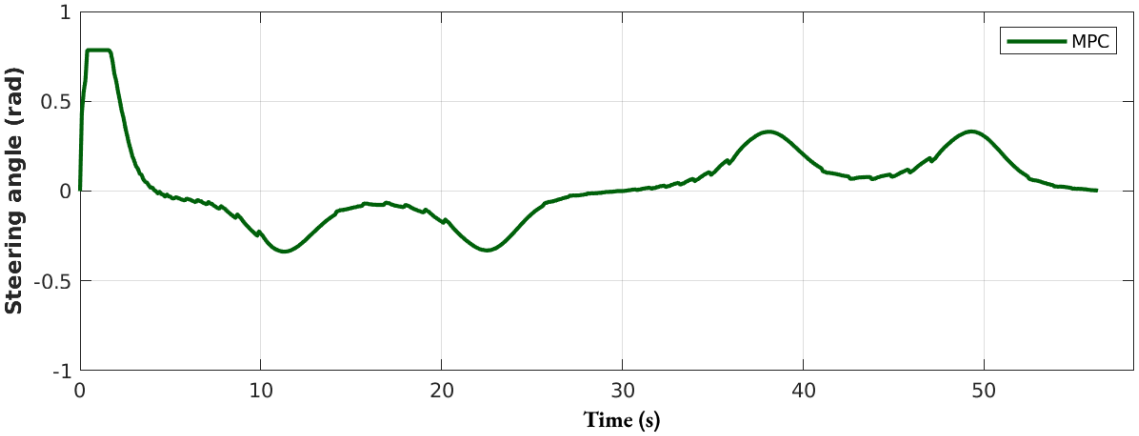
Conclusion: The test confirmed that Lone Wolf is capable of closely following a smooth and relatively simple path with minimal error.



(a) Big infinity: XY-graph



(b) Big infinity: Error



(c) Big infinity: Steering angle

Figure 7.6: Results: Big infinity

7.2.4 Test 5: Small infinity

Purpose: Test the functionality of the MPC with a trajectory reference that is physically impossible to follow. This will check how it deals with problematic situations like too-tight corners and difficult references. The expected behavior is for Lone Wolf to drive in a pattern that resembles the reference.

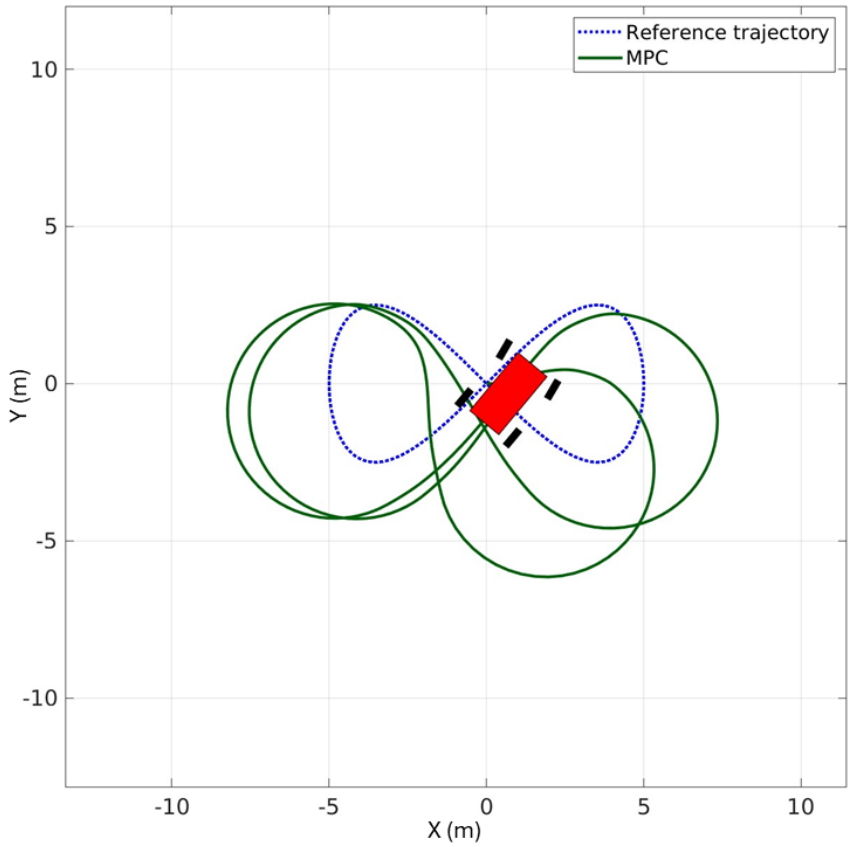
Conditions:

- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the reference trajectory ranges from 10km/h to 20km/h.

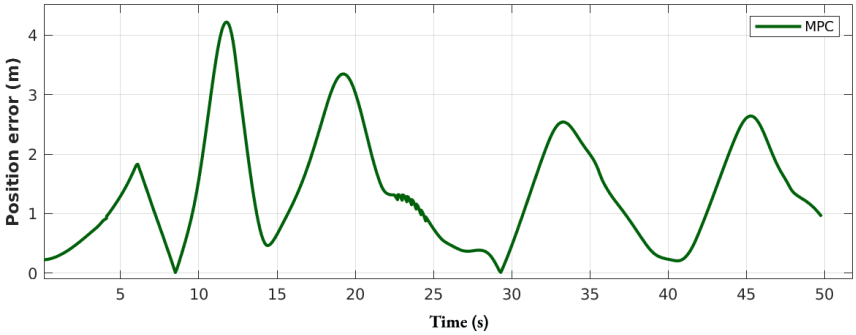
Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.7. Notable observations from these results include:

- The first lap follows a different curve then the following laps due to the initial heading of Lone Wolf.
- Lone Wolf made a valid attempt to follow the unfeasible reference trajectory, adjusting its trajectory in an acceptable manner.
- Figure 7.7c shows that the steering angle saturates as the MPC tries to do the impossible turns.

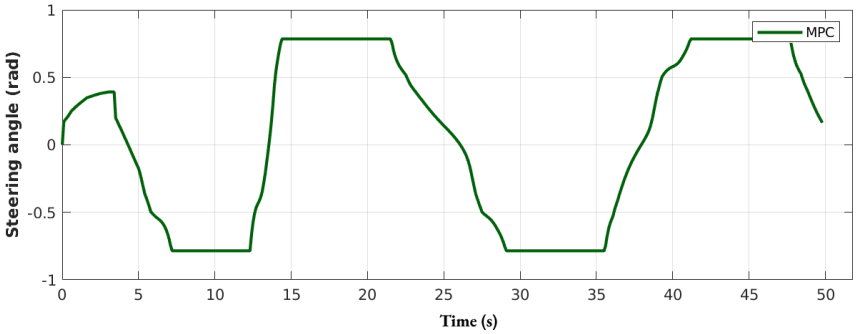
Conclusion: The test demonstrated that the MPC is capable of effectively managing situations where the reference trajectory is not physically feasible, making intelligent adjustments to minimize error over time.



(a) Small infinity: XY-graph



(b) Small infinity: Error



(c) Small infinity: Steering angle

Figure 7.7: Results: Small infinity

7.2.5 Test 6: Racetrack

Purpose: Test the functionality of the MPC with a varied trajectory reference containing a wide turn, a loop, a sine wave, and a 90-degree turn.

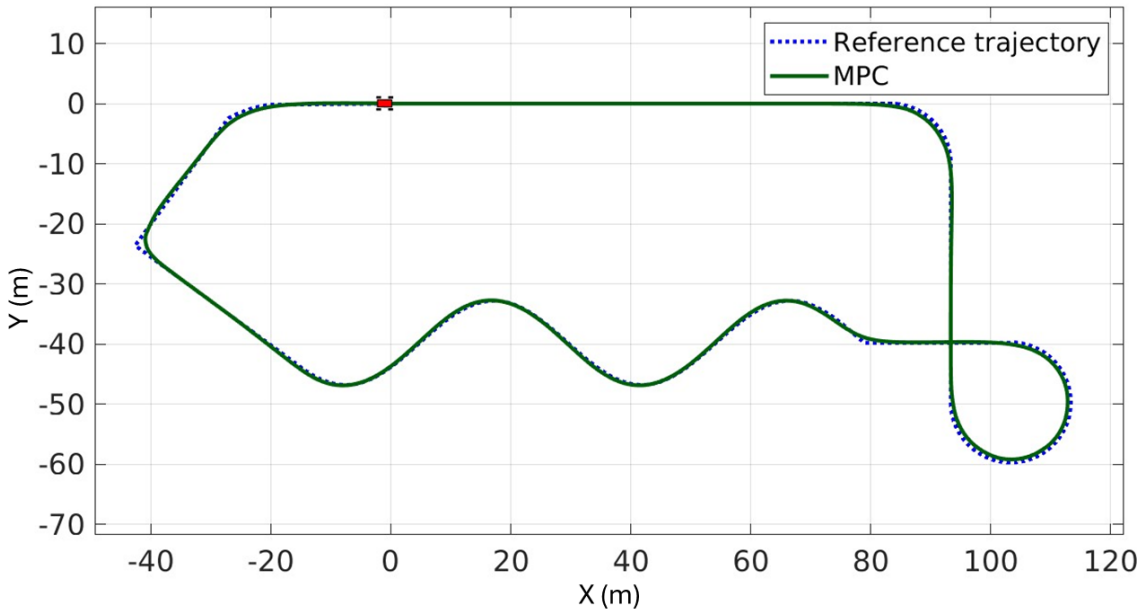
Conditions:

- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the trajectory reference ranges from 10km/h to 30km/h.

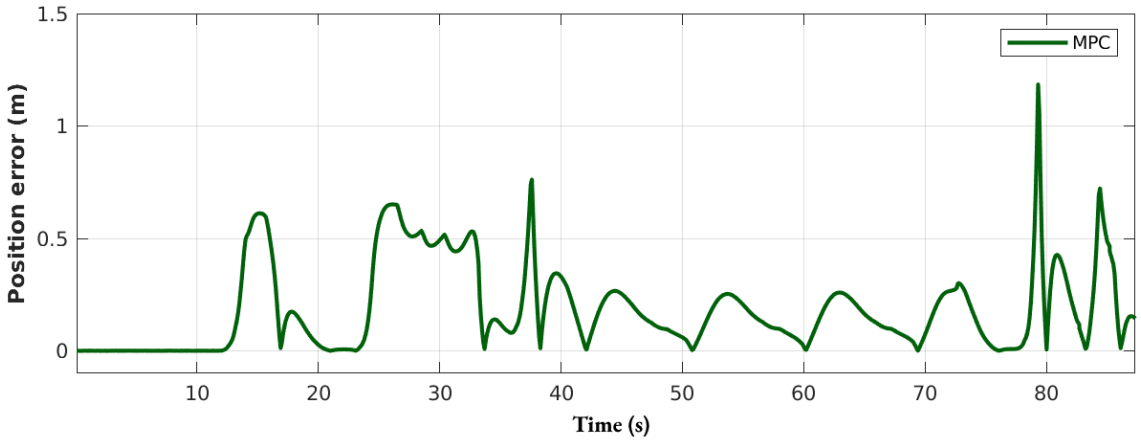
Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.8. Notable observations from these results include:

- The MPC is able to follow the racetrack efficiently.
- The error is generally low.

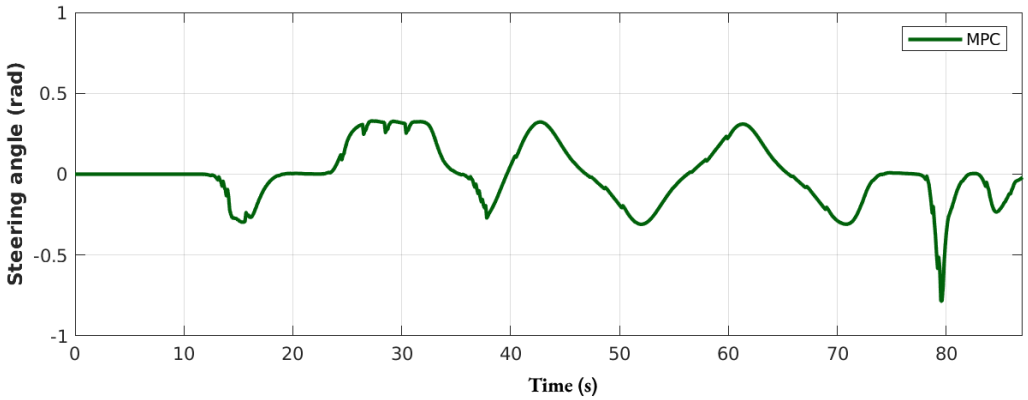
Conclusion: The test demonstrated that the MPC exhibits satisfactory behavior for a range of different trajectory references.



(a) Racetrack: XY-graph



(b) Racetrack: Error



(c) Racetrack: Steering angle

Figure 7.8: Results: Racetrack

7.2.6 Test 7: MPC in different road conditions

Purpose: Test how the MPC behaves when the road conditions change. This tests whether the no-slip assumption made in the MPC model is valid.

Conditions:

- Pacejka parameters set to *dry tarmac* and *ice*, as detailed in Table 7.1.
- Velocity of the trajectory is set to 20km/h.

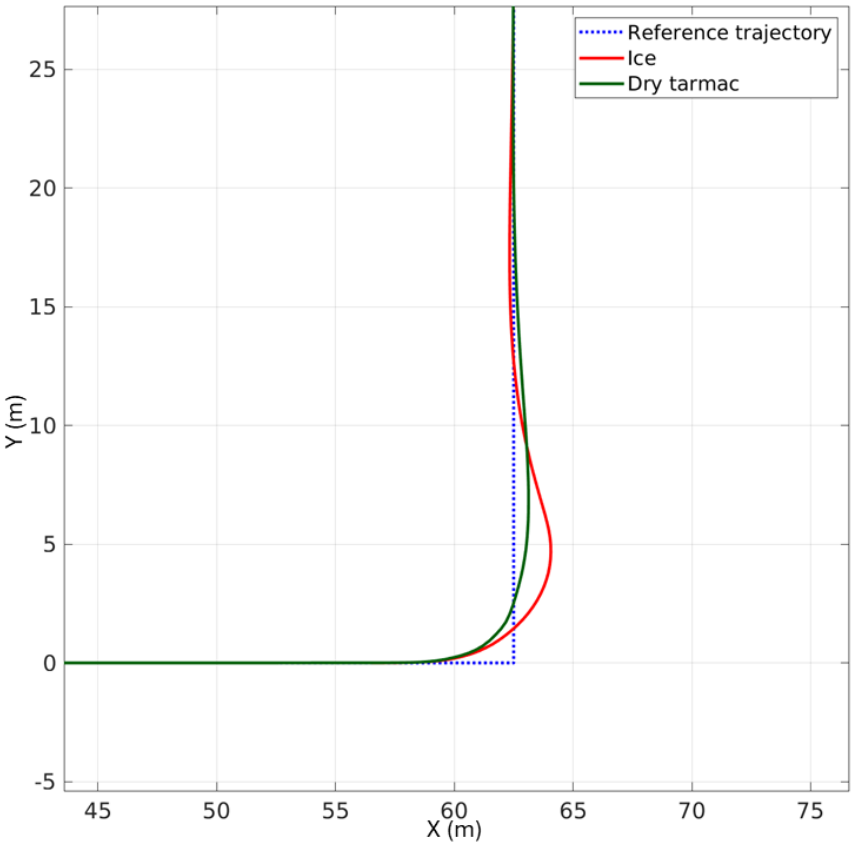
Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.9. The simulated vehicle's error metrics can be found in Table 7.2. Notable observations from these results include:

- Lone Wolf starts to drift under icy conditions but applies a corrective steering input to avoid over-turning.
- Error metrics are larger for icy conditions compared to dry tarmac.

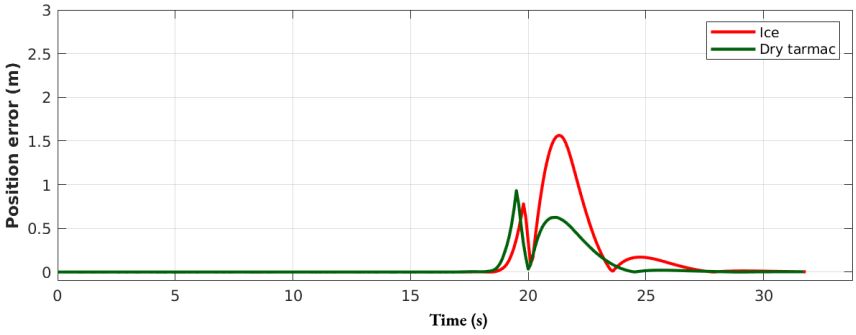
Conclusion: Slippery conditions leads to worse trajectory following and larger error metrics, however, the MPC is still able to follow the trajectory. This signals that the no-slip assumption is valid, because the slip of the simulated vehicle is not enough to cause detrimental errors.

Surface\Error	$\boldsymbol{RMSE}$	$\boldsymbol{E_{max}}$
Dry tarmac	0.1733m	0.9329m
Ice	0.3407m	1.5651m

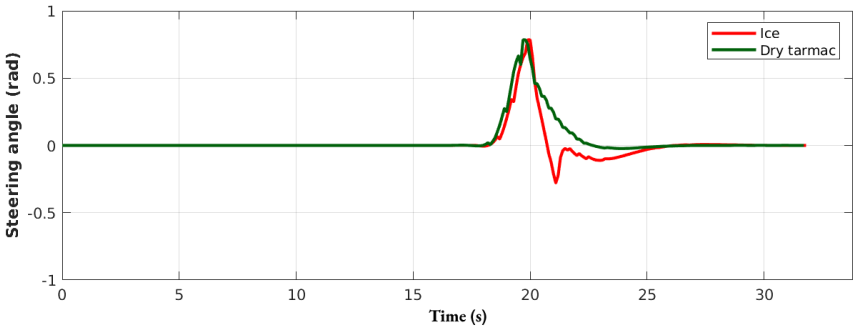
Table 7.2: MPC in different conditions: $RMSE$ and E_{max}



(a) MPC in different conditions: XY-graph



(b) MPC in different conditions: Error



(c) MPC in different conditions: Steering angle

Figure 7.9: Results: MPC in different road conditions

Test 8: Tuning of MPC

Purpose: Test evaluates the $R_{\Delta t}$-matrix tuning in the MPC, using a racetrack reference that incorporates elements from other tests. It compares three $R_{\Delta t}$-matrices for under-punished, balanced, and over-punished settings, detailed in Table 7.3. "Punished" denotes the $R_{\Delta t}$ matrix coefficients affecting the MPC's reluctance to change control outputs. Only the $R_{\Delta t}$-matrix is changed, because it is the ratio between the Q- and $R_{\Delta t}$-matrices that matters, and therefore changing the Q-matrix at the same time would lead to multiple aspects changing at the same time. The expected outcome is that the underpunished setting exhibits the lowest error metrics, despite potentially erratic steering.

Conditions:

- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the trajectory ranges from 10km/h to 30km/h.

Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.10. The simulated vehicle's error metrics can be found in Table 7.4. Notable observations from these results include:

- The under-punished MPC exhibits poorer path adherence than the balanced MPC.
- The over-punished MPC demonstrated longer and slower turns.
- Figure 7.10b and the zoomed in Figure 7.10c also reveal that the under-punished MPC showed erratic steering behavior.

Conclusion: As expected, the underpunished settings resulted in erratic steering behavior, as shown in Figure 7.10c. However, these settings did not yield the lowest performance metrics. This discrepancy may stem from the internal MPC model being less accurate for higher frequencies.

Overpunished settings led to overly cautious and delayed responses to path deviations, resulting in the largest error metrics.

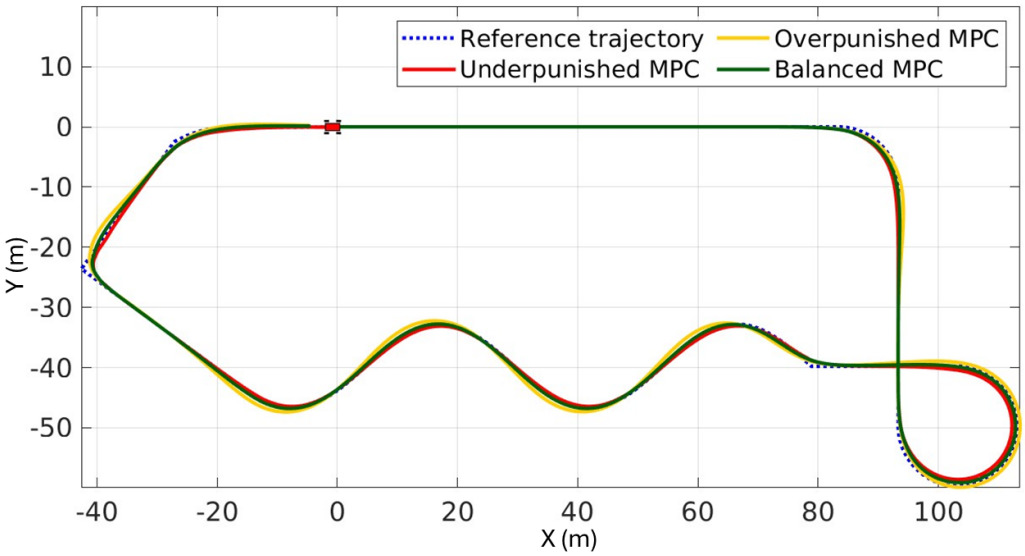
Given these outcomes, the balanced MPC tuning was selected for implementation, providing robust path following and sustainable performance without the drawbacks of either extreme.

$R_{\Delta t}$-matrix	Throttle	Steering
Balanced	0.02	8
Overpunished	0.02	32
Underpunished	0.02	1

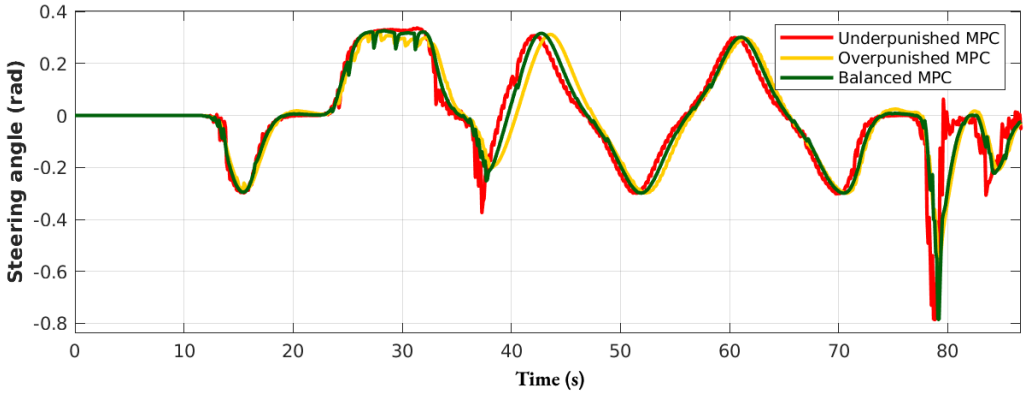
Table 7.3: MPC tuning: R-matrices

$R_{\Delta t}$-matrix	$RMSE$	E_{max}
Balanced	0.2953m	1.3402m
Overpunished	0.5346m	1.4244m
Underpunished	0.4088m	1.3585m

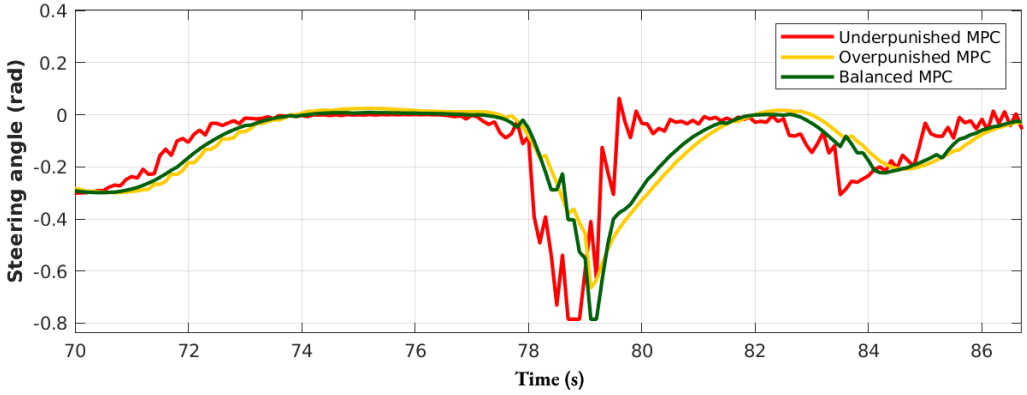
Table 7.4: MPC tuning: $RMSE$ and E_{max}



(a) MPC tuning: XY-graph



(b) MPC tuning: Steering input



(c) MPC tuning: Steering input - zoomed

Figure 7.10: Results: MPC tuning

This page has been left blank intentionally.

7.2.7 Test 9: Modelling errors

Purpose: Test both the tuning of the MPC and how well the MPC handles internal modelling errors. The expected outcome is that the well-tuned MPC model efficiently follows the trajectory reference, while the poorly tuned MPC still follows the track but with worse performance.

Conditions:

- Different internal MPC models as shown in Table 7.5
- Pacejka parameters set to *dry tarmac*, as detailed in Table 7.1.
- Velocity of the trajectory ranges from 10km/h to 30km/h.

Observations: The simulated vehicle's path, position error, and steering angle over time can be seen in Figure 7.11. Interesting observations from these results include:

- A poorly tuned MPC model will lead to less accurate trajectory-following. It struggles to make well-timed turns because of the discrepancy between the simulator model and the model used in the MPC.
- A poorly tuned MPC may still be able to follow the reference trajectory.
- A well tuned MPC significantly decreases the value in the error metrics.

Conclusion: The poorly tuned model is still able to follow the path. However, if the model is sufficiently wrong, the MPC will be unable to accurately predict its trajectory, resulting in unstable behavior. This is not illustrated in the images.

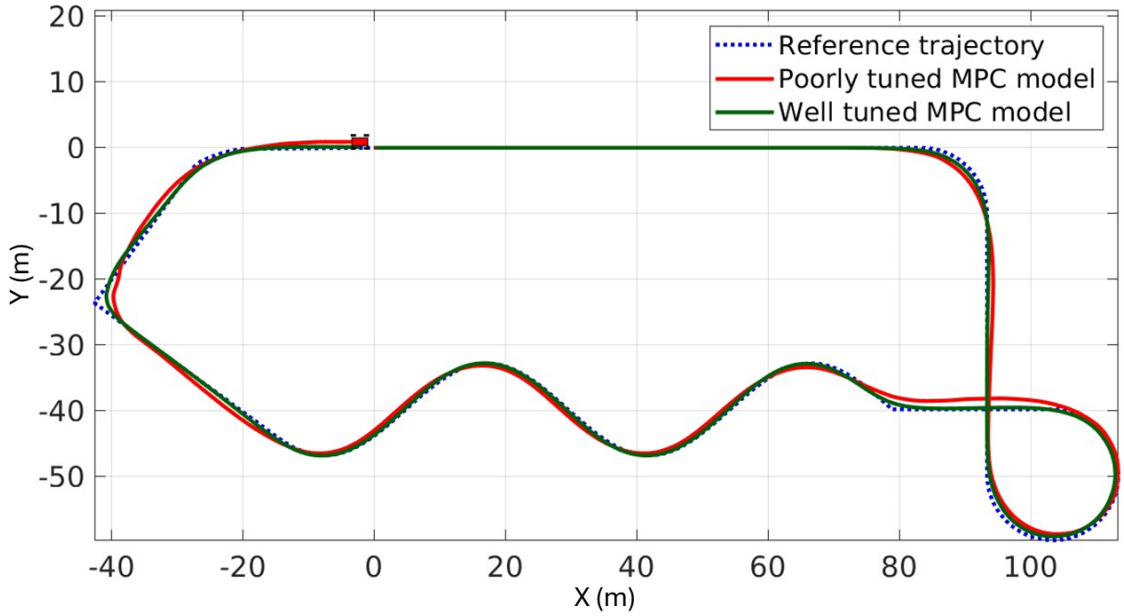
Figure 7.11a shows the paths taken by both MPCs. As can be seen in Figure 7.11b, the error is significantly higher for a poorly tuned MPC model compared to a well-tuned MPC model. Therefore, a sufficiently accurate MPC model is required for good reference-tracking.

MPC model	$\boldsymbol{\zeta}$	$\boldsymbol{\omega}$	$\boldsymbol{T}_{\text{gain}}$	$\boldsymbol{F_r}$
Well tuned	0.88	8.56	35	225
Poorly tuned	1.49	2.57	24.5	157.5

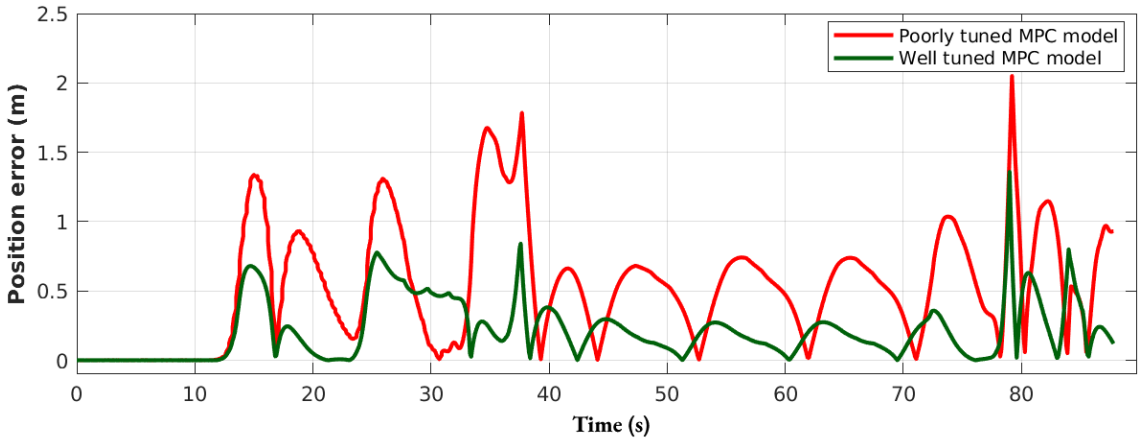
Table 7.5: MPC model tuning parameters

MPC model	$\boldsymbol{RMSE}$	$\boldsymbol{E}_{\max}$
Well tuned	0.2965m	1.3594m
Poorly tuned	0.5262m	2.0786m

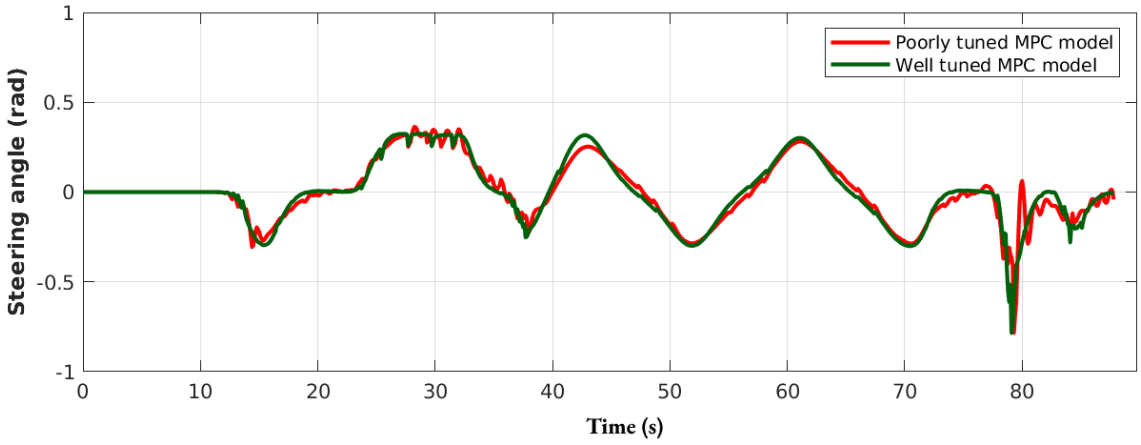
Table 7.6: MPC model tuning: $RMSE$ and $E_{\max}$



(a) MPC model tuning: XY-graph



(b) MPC model tuning: Error



(c) MPC model tuning: Steering Results

Figure 7.11: Results: MPC model tuning

7.3 Summary

This section summarizes the results of the tests and the conclusions drawn.

Test 1: Pacejka parametes shows that different Pacejka parameter values significantly impact vehicle behavior, with slower acceleration and greater velocity loss in icy and wet conditions, and better turning performance on dry surfaces.

Test 2: Straight line shows that the MPC successfully follows a basic trajectory with minimal error and no oscillations.

Test 3: 90-degree turn shows that the MPC effectively minimizes error despite infeasible trajectory references.

Test 4: Big infinity track demonstrates that the MPC closely adheres to a smooth and simple trajectory with minimal error.

Test 5: Small infinity shows that the MPC manages impossible trajectory references by making intelligent adjustments to minimize error.

Test 6: Racetrack shows that the MPC satisfactorily handles varied trajectories with different types of turns and curves.

Test 7: MPC in different road conditions validates the MPC model's no-slip assumption is generally adequate, though icy conditions leads to larger error metrics, as is expected.

Test 8: Tuning of MPC shows that a balanced $R_{\Delta t}$-matrix tuning provides the best performance, avoiding the drawbacks of both under-punished and over-punished settings.

Test 9: Modelling errors demonstrates that accurate tuning of the MPC model is essential for effective reference-tracking, as significant errors in the model leads to unstable behavior. Despite a small model error, the MPC is able to track the reference, although with a larger error.

The tests demonstrates that the MPC and the simulator effectively works together, which results in an efficient way of testing the MPC. Additionally, the results of the tests demonstrates the effectiveness of MPC in the context of reference tracking in autonomous vehicles.

This page has been left blank intentionally.

Chapter 8

Discussion

This chapter examines the modeling and simulation strategies implemented in the project. It discusses the approaches taken and the results of these. It also discusses the project's relevance to the Lone Wolf project, further work, and finally, how this project addresses the Sustainable Development Goals outlined in Chapter 1. The aim of this chapter is to justify the decisions made during the project in the context of its past developments and future directions.

8.1 Modeling methods

This thesis used an analytical approach to identify the dynamics of a generic four-wheel vehicle operating within the two-dimensional XY-plane. This approach has produced satisfying outcomes, presented in Chapter 7. It is important to note that although the model is based on some known specifications from the physical Lone Wolf, the simulation results presented are not validated, and probably diverge from real-world behavior. This is due to the lack of real-world data.

Various approaches were considered for modeling the ATV's engine, as detailed in Section 3.6. Ideally, obtaining a comprehensive power map from Lone Wolf, including the engine and transmission, would provide the most accurate model. However, a substitute engine map was used instead due to the challenges of acquiring this data from manufacturers. This alternative map has considerable discrepancies compared to the one for the actual engine. These discrepancies were addressed by applying a scaling constant to the torque values. After this adjustment, the engine dynamics were deemed satisfactory.

When modeling the drivetrain of the ATV, the main objective was to accurately represent the vehicle's dynamics while keeping the simulator's complexity manageable. This was accomplished by focusing on the most critical components of the drivetrain, the Continuously Variable Transmission's (CVT's) gear ratio, and the final drive. As discussed in Section 4.3.3, the model does not account for transmission losses and simplifies the dynamics of the CVT. Although these simplifications introduce some discrepancies compared to the actual Lone Wolf ATV, the overall results are still considered satisfactory.

However, the first thesis objective stated to *"Structure the model to allow adjustable parameters, enabling future modifications to closely simulate the physical Lone Wolf ATV dynamics"*. By integrating adjustable parameters such as the Pacejka parameters, the model can be modified at a later stage utilizing grey box modeling methods described in Section 3.1. As seen in Section 7.1, a change in these parameters significantly affects the behavior of the simulated vehicle. It is not proven that such an approach would give good enough results, but the foundation has been laid. Due to this, the first thesis objective is considered fulfilled.

8.2 Simulator implementation

According to the second thesis objective, *"Implement the dynamic model in a simulator"*, along with the fourth thesis objective, *Implement the control algorithm into the Lone Wolf system*, a ROS2-compatible software had to be chosen for the simulator. During previous work, Gazebo has been used for simulation. The vehicle simulated in Gazebo had small opportunities for customization, and it was deemed too hard to implement the advanced tire and powertrain models. As a result, Simulink was chosen as the preferred software. Simulink supports ROS2 communication and makes it possible to build dynamic systems from the bottom up. In this way, everything that happens inside the implemented model is known, and no pre-defined physics or parameters in the simulation software will interfere.

8.3 MPC discussion

The MPC has satisfactory results in the tests. The tuning was chosen to balance reference tracking and the longevity of the actuators. The simulated Lone Wolf is able to follow a diverse set of reference trajectories. Figure 8.1 shows Lone Wolf following the Racetrack reference. The MPC is able to communicate with the simulated Lone Wolf with the same ROS2 interface as the real Lone Wolf. Thus, the MPC is integrated into the existing software architecture.

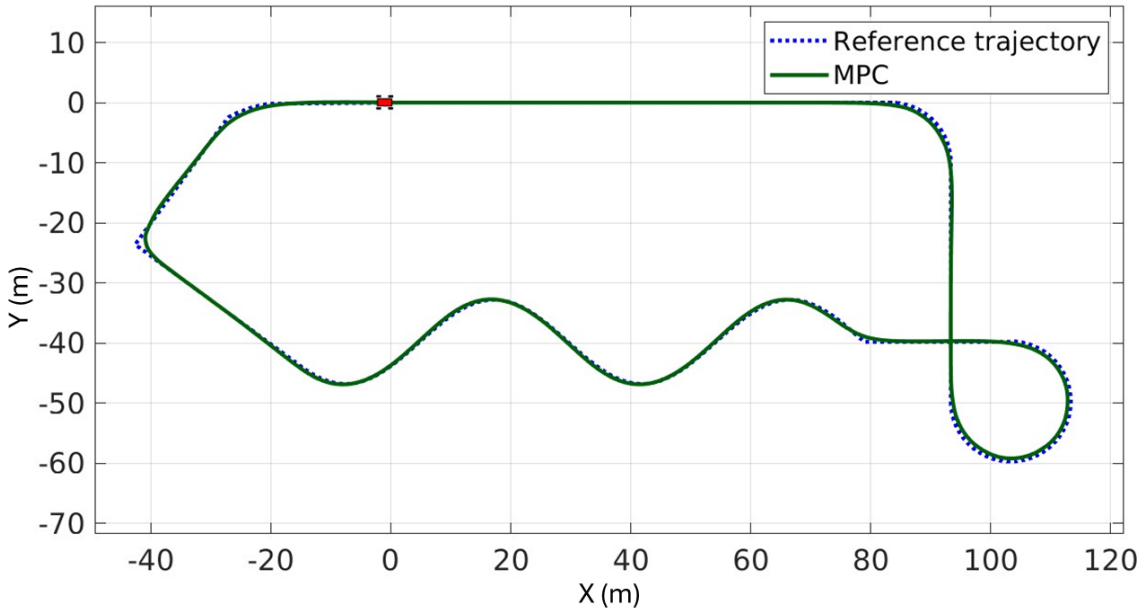


Figure 8.1: Lone Wolf on racetrack reference

The MPC is tuned for the simulated Lone Wolf, which will not perfectly represent the real Lone Wolf. Therefore, it needs to be adjusted before testing in real life. This has been prepared with adjustable parameters, including a user guide to determining these found in Appendix B. This will make it easier to complete the implementation when data is collected from Lone Wolf. Additionally, the brakes have not been included in the MPC model, as these have a difficult set of dynamics in real life and would not be a good representation in the simulation.

Overall, the MPC works well for the tested reference trajectories, and it has been integrated with the current system architecture of the real Lone Wolf. This satisfies thesis objective 3, *Develop a Model Predictive Controller*, and objective 4, *Implement the control algorithm into the Lone Wolf system*, as stated in Section 1.3.

8.4 Relevance for Lone Wolf

Relevance to the physical Lone Wolf has been a big priority throughout the project. This has led to certain choices being made for the ROS2 interface, dynamic model for the MPC, and reference passed to the MPC.

8.4.1 ROS2 interface

In accordance with objective 4 stated in the Project Assignment in Chapter 1, the simulator has been created with the same ROS2 interface as the real Lone Wolf. This has ensured that the MPC has the same ROS2 interface as the physical Lone Wolf. In this way, the MPC can easily be implemented on Lone Wolf with few structural changes to the software, and only requires changes to the model parameters and reference trajectory.

8.4.2 MPC model adjustments

The MPC model was created with adjustable parameters. When data is recorded and analyzed, the MPC model can be fitted to the responses of the physical Lone Wolf via the adjustable parameters. A guide on how to do this can be found in Appendix B.

8.4.3 Reference input

The reference is passed to the MPC via ROS2, and is the same length as the prediction horizon of the MPC. It was set up in anticipation of a working SLAM/ pathfinding algorithm, which can generate a trajectory, when the ATV is in operation. The "Closest point" method makes sure the reference always begins at the closest point to Lone Wolf. This simulates the trajectory expected from the pathfinding algorithm, which could make the implementation of a pathfinding algorithm a more streamlined process.

8.5 Further work

The work presented in this thesis establishes a solid groundwork for further improvements to the Lone Wolf system. The following tasks are suggested extensions of the efforts made in developing the simulator and MPC throughout this thesis:

8.5.1 Data collection and grey-boxing

The original plan for the project was to take a data-driven approach to modelling the dynamics of Lone Wolf. Due to unforeseen challenges, this did not happen, and the approach was changed to an analytical one. To more accurately estimate the dynamics of Lone Wolf, a data-driven approach could be beneficial, and a recommended course of action. Investing time in the dynamic model will improve the simulator to more accurately represent the physical Lone Wolf, which can be used to improve the MPC and its performance, which would reduce implementation time.

8.5.2 Include brakes in the control algorithm

Currently, the brakes are not used in the control algorithm. This is due to its close-to on/off characteristics. The brakes are used primarily to stop completely because the engine brakes often offer sufficiently good brakes, and the MPC uses this in its predictions. Implementing the brakes would make stops a viable option for the MPC.

8.5.3 State estimation - Extended Kalman Filter

One of the assumptions made for the project is that the sensor data is sufficiently accurate. Realistically, this is not going to be the case, especially for the GPS positions. The Vectornav, although a high-quality component, experiences drift which will impact the accuracy of the sensor data. A commonly used method for state estimation is implementing an Extended Kalman Filter (EKF). An EKF is an optimal state estimation algorithm that filters random noise in the sensor data and provides an improved estimate of the current state. Kalman filters are widely used in navigation to estimate positions and bearing angles. [18]

Figure 8.2 shows where the EKF could be implemented in the system. An EKF, compared to a regular Kalman filter, would be a good option for the Lone Wolf system, because it can handle a non-linear system, such as the one identified for Lone Wolf in this thesis.

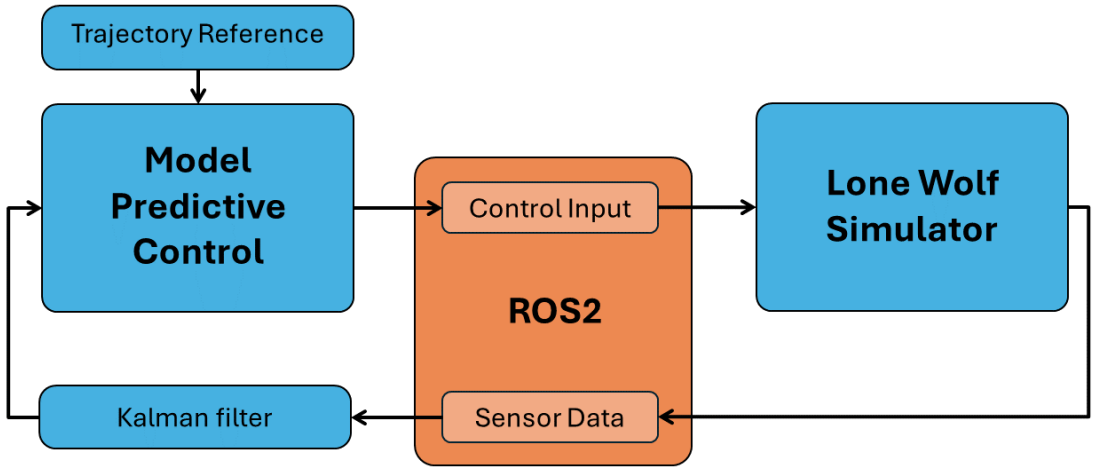


Figure 8.2: Kalman filter implementation

8.6 Sustainable Development Goals

This project aligns with Sustainable Development Goals (SDGs) eight, nine, and eleven, as detailed in Section 1.6. The advancement of autonomous vehicles, such as the Lone Wolf project, supports these goals. While the outcomes of this project are preliminary, as development is ongoing, the research and development conducted during this thesis contribute to further advancements in the Lone Wolf project. Upon completion of the Lone Wolf project, the autonomous ATV will be positioned to address the objectives associated with the previously mentioned SDGs.

This page has been left blank intentionally.

Chapter 9

Conclusion

This thesis has presented the contribution to the ongoing development of Lone Wolf, a student project emphasizing the development of an autonomous all-terrain vehicle. Throughout the course of this study, the objectives laid out in the introduction have been systematically and successfully addressed, culminating in a series of achievements that fulfill the project objectives.

The initial aim to model the dynamics of the Lone Wolf ATV was shifted to a more general goal due to unforeseen circumstances. This revised objective led to the successful identification and modeling of an ATV's dynamics. The dynamic model was implemented in Simulink, a ROS2-compatible simulator, with the same ROS2 interface as the real Lone Wolf ATV. This enabled the development and integration of a Model Predictive Controller into the existing software architecture. Additionally, the flexibility of the developed model and the integration approach ensures that the advancements made here can be leveraged in subsequent projects.

With these accomplishments, all the objectives laid out at the outset of this project have successfully been fulfilled.

Bibliography

[1] Olav Aaen. *Clutch Tuning Handbook*. 2007. URL: https://archive.org/details/aaen-clutch-tuning-handbook-2007.

[2] Robot Academy. *Summary of paths and trajectories*. May 2024. URL: https://robotacademy.net.au/lesson/summary-of-paths-and-trajectories (visited on 10/05/2024).

[3] Brahimi Adel, Youtong Zhang and Shijin Shuai. 'Parallel HEV Hybrid Controller Modeling for Power Management'. In: *World Electric Vehicle Journal* 4 (Mar. 2010), pp. 190–196. DOI: 10.3390/wevj4010190.

[4] *Bike Physics*. Feb. 2024. URL: https://sites.google.com/site/bikephysics/english-version/2-geometry-and-kinematics (visited on 04/02/2024).

[5] Kåre Bjørvik and Per Hveem. *Reguleringsteknikk*. Trondheim, Norway: Kybernetes, 2014. ISBN: 9788292986219.

[6] Encyclopaedia Britannica. 'Centrifugal force: Definition, Examples & Facts'. In: *Encyclopedia Britannica* (Feb. 2024). URL: https://www.britannica.com/science/centrifugal-force (visited on 03/04/2024).

[7] Automobile Catalog. *Detailed specs review of 2011 Audi A4 3.2 FSI Quattro Tiptronic offered up to late-year 2011 for Europe*. May 2024. URL: https://www.automobile-catalog.com/car/2011/1187660/audi_a4_3_2_fsi_quattro_attraction_tiptronic.html#gsc.tab=0 (visited on 03/05/2024).

[8] Maksym Diachuk and Said M. Easa. 'Modeling Combined Operation of Engine and Torque Converter for Improved Vehicle Powertrain's Complex Control'. In: *Vehicles* 4.2 (2022), pp. 501–528. DOI: 10.3390/vehicles4020030. URL: https://www.mdpi.com/2624-8921/4/2/30.

[9] Collins Dictionary. *Final drive: Definition and meaning*. May 2024. URL: https://www.collinsdictionary.com/dictionary/english/final-drive (visited on 07/05/2024).

[10] Yan Ding. 'Simple Understanding of Kinematic Bicycle Model'. In: (Dec. 2021). URL: https://dingyan89.medium.com/simple-understanding-of-kinematic-bicycle-model-81cac6420357 (visited on 03/04/2024).

[11] Bjarne Foss and Tor Aksel N. Heirung. 'Merging Optimization and Control'. In: *NTNU* (2016), pp. 39–50. URL: https://folk.ntnu.no/bjarnean/Publications/OptimalControl.pdf.

[12] *Fox Run Auto Inc.* May 2024. URL: https://www.foxrunauto.com/blog/powertrain-vs-drivetrain-vs-driveline (visited on 07/05/2024).

[13] Eline Marie Håve, Sigrid Mellemseter and Cecilie Nikolaisen. *ROS Simulated World for ATV with SLAM.* 2022. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3002448.

[14] ISO. *8855:2011, Road vehicles - Vehicle dynamics and road-holding ability - Vocabulary.* Mar. 2024. URL: https://www.iso.org/obp/ui/#iso:std:iso:8855:ed-2:v1:en (visited on 20/03/2024).

[15] Siti Norbakyah Jabar et al. 'Modeling, simulation and model optimization of internal combustion engine for PHERB powertrain'. In: *Jurnal Teknologi* 79 (June 2017). DOI: 10.11113/jt.v79.9262.

[16] *Join the pack: Project Lone Wolf.* May 2024. URL: https://www.kongsberg.com/careers/summer-interns/lone-wolf (visited on 02/05/2024).

[17] Andreas Jonsebråten, Vegar Karlsen and Glenn R. Varhaug. *Graph-based Path Planning in a Simulated Rough Terrain.* 2023. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/3080586.

[18] Youngjoo Kim et al. 'Introduction to Kalman Filter and Its Applications'. In: *Introduction and Implementations of the Kalman Filter.* IntechOpen, Nov. 2018. ISBN: 978-1-83880-537-1. DOI: 10.5772/intechopen.80600.

[19] Kiran Kone. *Lateral and longitudinal control of an autonomous racing vehicle.* Oct. 2019. URL: https://webthesis.biblio.polito.it/11982 (visited on 03/04/2024).

[20] Bin Li, Xiaobo Yang and Jingzhou Yang. 'Tire Model Application and Parameter Identification-A Literature Review'. In: *SAE International Journal of Passenger Cars - Mechanical Systems* (May 2014). DOI: 10.4271/2014-01-0872.

[21] Mathworks. *3DOF rigid vehicle body to calculate longitudinal, lateral, and yaw motion.* Mar. 2024. URL: https://se.mathworks.com/help/vdynblks/ref/vehiclebody3dof.html (visited on 20/03/2024).

[22] Mathworks. *Black-Box Modeling.* May 2024. URL: https://se.mathworks.com/help/ident/ug/black-box-modeling.html (visited on 10/05/2024).

[23] Mathworks. *Choose Sample Time and Horizons*. May 2024. URL: https://se.mathworks.com/help/mpc/ug/choosing-sample-time-and-horizons.html (visited on 06/05/2024).

[24] Mathworks. *Simulink Documentation*. May 2024. URL: https://se.mathworks.com/help/simulink (visited on 03/05/2024).

[25] United Nations. *the 17 goals - Sustainable Development*. May 2024. URL: https://sdgs.un.org/goals (visited on 13/05/2024).

[26] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. Springer, 2006. ISBN: 9780387227429.

[27] Hans Pacejka. *Tyre and Vehicle Dynamics*. Amsterdam, Netherlands: Elsevier Science, 2006. ISBN: 980-0-7506-6918-4.

[28] *Pacejka'94 parameters explained*. Feb. 2024. URL: https://www.edy.es/dev/docs/pacejka-94-parameters-explained-a-comprehensive-guide (visited on 20/04/2024).

[29] R Rajamani. *Vehicle Dynamics and Control*. Troy, USA: Springer, 2006. ISBN: 0-387-26396-9.

[30] Oxford Reference. *Frame of reference*. May 2024. URL: https://www.oxfordreference.com/display/10.1093/oi/authority.20110803095832143 (visited on 11/05/2024).

[31] ROS. *Is there an upper limit on the number of running ROS nodes?* May 2024. URL: https://answers.ros.org/question/235851/is-there-an-upper-limit-on-the-number-of-running-ros-nodes (visited on 04/05/2024).

[32] ROS. *msg - ROS Wiki*. May 2024. URL: http://wiki.ros.org/msg (visited on 13/05/2024).

[33] ROS. *Nodes*. May 2024. URL: http://wiki.ros.org/Nodes (visited on 04/05/2024).

[34] ROS. *ROS 2 Documentation: Foxy documentation*. May 2024. URL: https://docs.ros.org/en/foxy/index.html (visited on 03/05/2024).

[35] ROS. *Topics*. May 2024. URL: http://wiki.ros.org/Topics (visited on 04/05/2024).

[36] ROS. *Understanding topics*. May 2024. URL: https://docs.ros.org/en/iron/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (visited on 04/05/2024).

[37] Dieter Schramm, Manfred Hiller and Roberto Bardini. 'Vehicle Dynamics'. In: Berlin, Germany: Springer, July 2014, pp. 143–184. ISBN: 978-3-540-36045-2.

[38] Joga Dharma Setiawan, Mochamad Safarudin and Amrik Singh. 'Modeling, simulation and validation of 14 DOF full vehicle model'. In: Bandung, Indonesia: IEEE, 2009, pp. 1–6. DOI: 10.1109/ICICI-BME.2009.5417285.

[39] Mankaran Singh and Mario Theers. *Kinematic Bicycle Model - Algorithms for Automated Driving*. May 2023. URL: https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/BicycleModel.html (visited on 03/04/2024).

[40] Ingrid Hovda Storaas and Ingrid Bjørndal Farestvedt. 'Lagerbygg har rast sammen i Kongsberg'. In: (Feb. 2024). URL: https://www.vg.no/nyheter/i/2BEpnq/lagerbygg-har-rast-sammen-i-kongsberg (visited on 10/05/2024).

[41] *Why is the time constant 63.2% and not 50% or 70%?* May 2024. URL: https://electronics.stackexchange.com/questions/396653/why-is-the-time-constant-63-2-and-not-50-or-70 (visited on 13/05/2024).

[42] Wikimedia. *Euler method*. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=Euler_method&oldid=1218269416 (visited on 01/05/2024).

[43] Wikipedia. *Grey box model*. Apr. 2021. URL: https://en.wikipedia.org/w/index.php?title=Grey_box_model&oldid=1017274915 (visited on 11/05/2024).

[44] Wikipedia. *Interior-point method*. Mar. 2024. URL: https://en.wikipedia.org/w/index.php?title=Interior-point_method&oldid=1216395542 (visited on 01/05/2024).

[45] Wikipedia. *IPOPT*. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=IPOPT&oldid=1221071743 (visited on 01/05/2024).

[46] Wikipedia. *Nonlinear system*. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=Nonlinear_system&oldid=1217624912 (visited on 11/05/2024).

[47] Wikipedia. *Robot Operating System*. Mar. 2024. URL: https://en.wikipedia.org/w/index.php?title=Robot_Operating_System&oldid=1214363848 (visited on 04/05/2024).

[48] Wikipedia. *System identification*. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=System_identification&oldid=1219694103 (visited on 11/05/2024).

[49] Wikipedia. *Tire model*. Apr. 2024. URL: https://en.wikipedia.org/w/index.php?title=Tire_model&oldid=1220238481 (visited on 11/05/2024).

[50] Wikipedia. *White box*. Jan. 2023. URL: https://en.wikipedia.org/w/index.php?title=White_box_(software_engineering)&oldid=1134333943 (visited on 11/05/2024).

[51] WKM. *Can Am Large Gear Box*. May 2024. URL: https://www.warrantykillerperformance.com/products/can-am-large-gear-box#includes (visited on 06/05/2024).

[52] x-engineer.org. *How to calculate wheel torque from engine torque*. Apr. 2024. URL: https://x-engineer.org/calculate-wheel-torque-engine (visited on 30/04/2024).

This page has been left blank intentionally.

Appendix A

Bachelor Assignment

NTNU

Institutt for elektroniske systemer
Institutt for elkraftteknikk
Institutt for teknisk kybernetikk

Oppgaveforslag bacheloroppgave elektroingeniør (BIELEKTRO) i Trondheim, vårsemester 2024

Navn bedrift: Kongsberg Defence & Aerospace	**Kontaktperson:** Roger Werner Laug **Epost:** roger.werner.laug@kongsberg.com **Telefon/mobil:** 98220530	
Tittel på oppgave: ***Utvikle dynamisk modell av Lone Wolf ATV med tilhørende kontrollalgoritmer***		
Hvilke studieretninger passer oppgaven for? (kryss av for alle aktuelle retninger; flervalg er mulig):	**Automatisering og robotikk**	☒
	Elektronikk og sensorsystemer	☐
	Elkraft og bærekraftig energi	☐
Er oppgaven reservert for noen bestemte studenter? (skriv navnene på studentene inn til høyre)	Eirik Bjørnevik, Eskil Bakken, Jørgen Vesterheim Knutsen, Vegard Ohren	
Har dere arbeidsplass for studentene	☐ ja ☒ nei ☐ usikker?	
Er dette en lukket oppgave? Dvs. at sluttrapporten ikke kan publiseres fordi den inneholder sensitiv informasjon.	☐ ja ☒ nei ☐ ikke enda bestemt	

Kort beskrivelse av oppgaven med problemstilling.

Lone Wolf er sommerprosjektet for studenter i Kongsberg Defence & Aerospace (KDA), Division Land Systems (DLS) der vi utforsker ulike teknologier. Denne Bacheloroppgaven passer godt inn i utviklingen til neste års sommerprosjekt.

Lone Wolf ATV'en er konstruert til å kunne svinge, akselerere og bremse slik at den skal kunne kontrolleres fra en datamaskin. Det er tidligere implementert path planning for å finne beste vei fra start til mål. Det er laget og montert en egenutviklet, miljøsikker kasse til en prosesseringsenhet med egen strømforsyning. Prosesseringsenheten er ment til å kjøre den autonome pipeline, slik at kjøretøyet kan kjøre av seg selv. Autonomi er hovedsakelig testet i simulator til nå, og kun delvis på ATV. ATV'en har en fjernstyrt nødstopp som skal brukes under kjøring. Kommunikasjon mellom ATV og en operatør posisjon er laget med en to-kanals kommunikasjonsløsning. Fra operatørposisjon kan ATV styres med en play-station kontroller, og tilbake fra kjøretøyet mottas GPS-punkter, fart og retning som vises på skjerm.

Det er flere oppgaver på Lone Wolf som gjenstår for å kunne kjøre godt og trygt autonomt. Denne oppgaven vil bygge på tidligere utviklet software og valgte komponenter, og utvide med en fysikk-modell av ATV der kontroll av styring, gasspådrag og brems implementeres. Dette er oppgavens deler:

- Kartlegge dynamikken til ATVen. Basere seg på ATVens spesifikasjoner samt ta målinger på responsen av de ulike aktuatorene (i hovedsak gasspådrag, brems og rattutslag).
- Implementere fysisk modell i simulator basert på funnene i punktet over. I Gazebo eller annen ROS2 kompatibel simulator. Ta i bruk aktuelle sensorer.
- Implementere kontrollalgoritme som baserer seg på fysisk modell av systemet, samt tilgjengelig sensordata fra GPS/INU, eksempelvis MPC
- Kontrollalgoritmen skal integreres inn i den nåværende system- og softwarearkitekturen, slik den kan tas lett i bruk i kommende prosjekter.

Alternative utvidede oppgaver:

- Se på nødvendighet og mulighet av closed-loop regulering av vinkel på rattutslag. Tilføres i simulator.
- Sammenligne kontrollalgoritmene. Dagens regulator er en PID på rattutslag og hastighet. Se på respons og ytelse på disse.
- Dagens system baserer seg i hovedsak på GPS for posisjonsestimering og orientering. Se på muligheten til å fusjonere de ulike sensorene inn med modellen for å gi et bedre estimat av de ulike tilstandene, eks LiDAR.

Dersom det er behov for utstyr skal KDA være behjelpelig med det (begrenset av tilgjengelighet og kostnad). KDA har hatt et møte med studentene og de virker interessert i oppgaven.
https://www.kongsberg.com/careers/summer-interns/lone-wolf/

Appendix B

Tuning of MPC internal model

Tuning of the Internal MPC Model

When tuning the internal MPC model, it is recommended to choose parameter values based on a step response for simplicity, and later validate them with different size steps and nonlinear inputs. For selecting parameters for the throttle, the following procedure is suggested and examples are provided in Figures B.1, B.2, B.3, and B.4:

- Create an input step for the throttle which is a square pulse with a width that allows sufficient time at the peak to reach a steady state. Log the velocity over time.
- Simulate the same throttle step in Python or other software using the state space model found in Equation 6.21. For this, the software `MPC_internal_model_guide.py` has been created. This can be found on Azure or the Predator Orion computer belonging to KDA.
- Start with a $\tau = 0.01$ and initially focus on T_{gain} and F_r values.
- Find a combination of T_{gain} and F_r that results in the same steady-state velocity as the physical test. Multiply both parameters by a constant to approximate the initial shape of the curve, with a larger constant leading to a sharper rise and fall rate.
- Fine-tune the shape of the curve with τ if necessary.

For selecting the values for the steering dynamics, ζ and ω, follow this procedure for constructing a second-order filter:

1. Start with real poles as shown in Equation 6.6 placing the time constants τ strategically.
2. Convert these time constants to the $\zeta - \omega$ format.
3. Manually tune the ζ and ω values to match the real steering response. This tuning involves adjusting the damping ratio, ζ, and the natural frequency, ω, until the simulated response closely follows the observed response.

An example of the tuning result can be seen in Figure B.5, which compares the step response of the internal MPC model to the simulator's response.

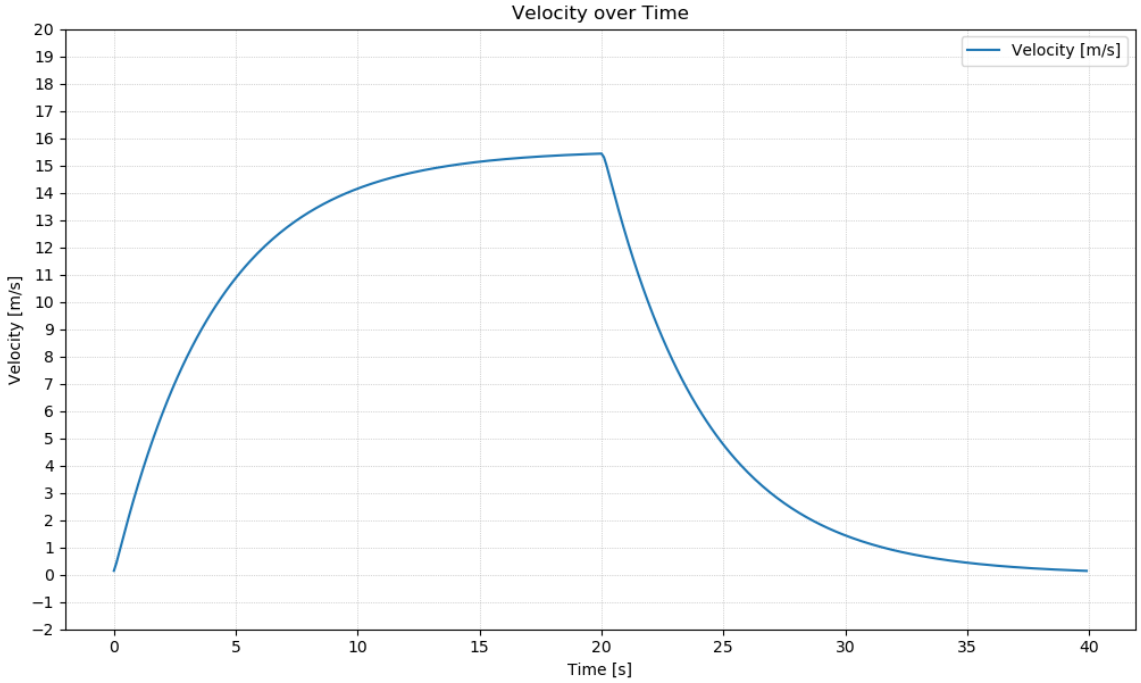


Figure B.1: Throttle: Low T_{gain} and F_r

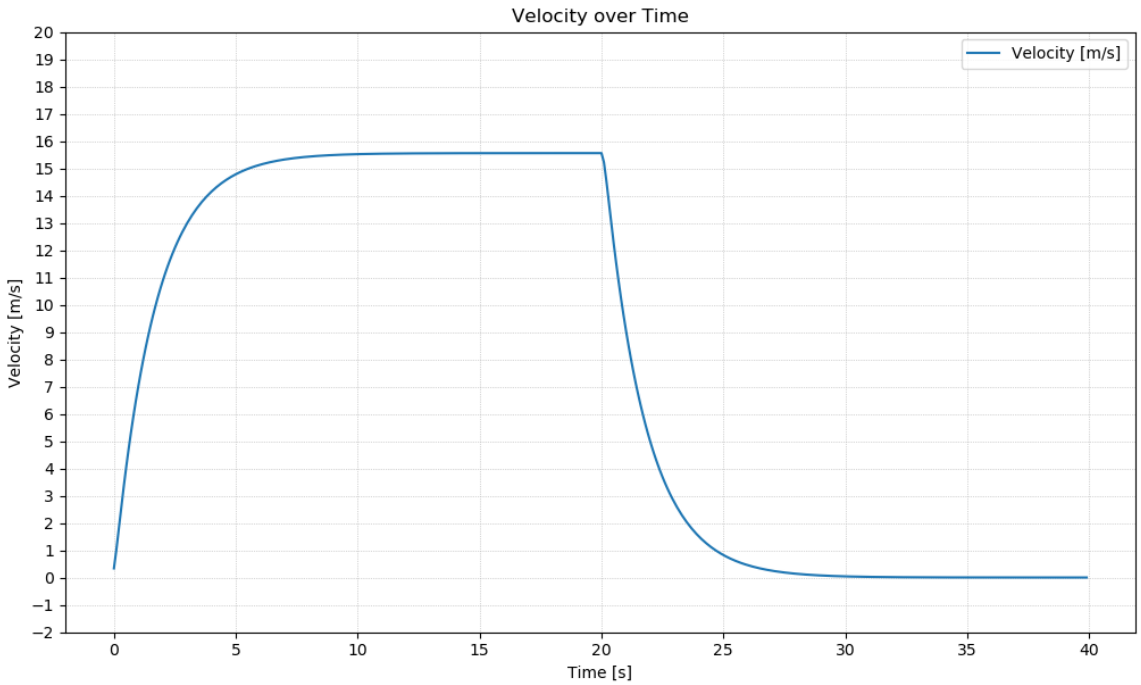


Figure B.2: Throttle: High T_{gain} and F_r

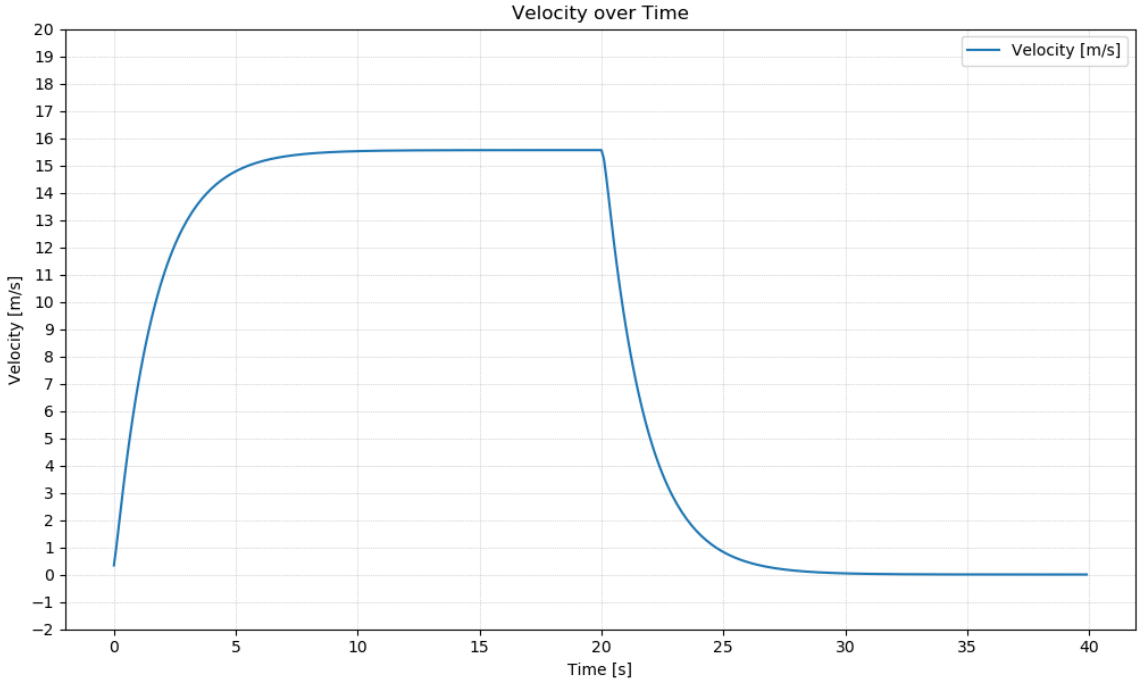


Figure B.3: Throttle: $\tau \approx 0$

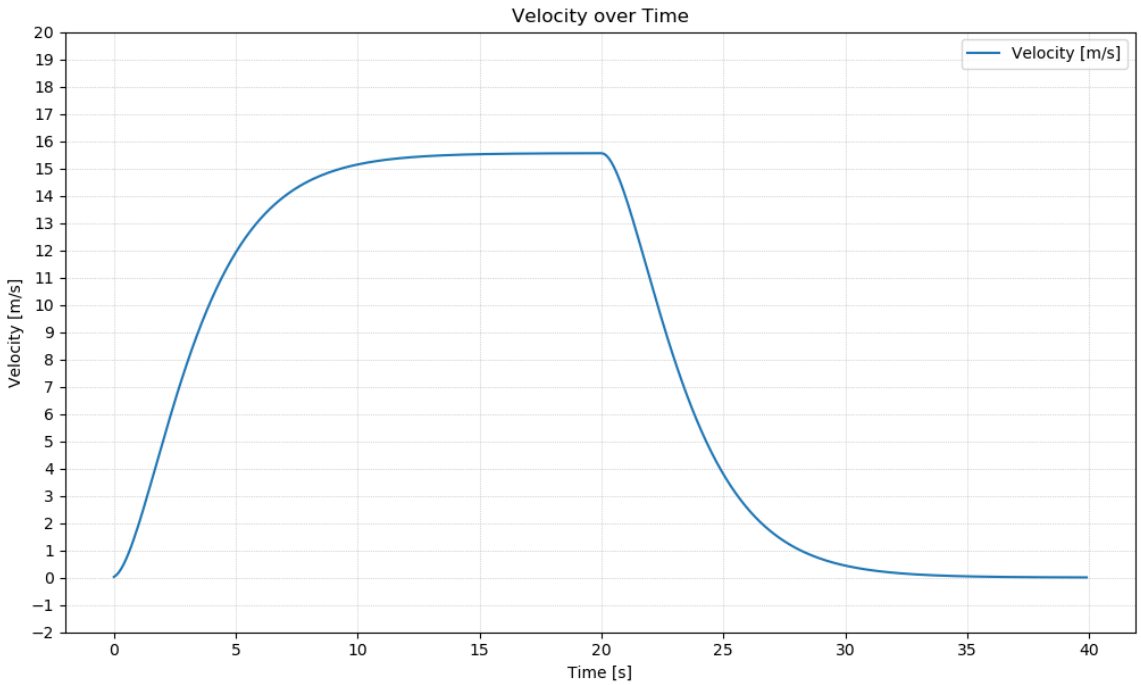


Figure B.4: Throttle: $\tau = 2$

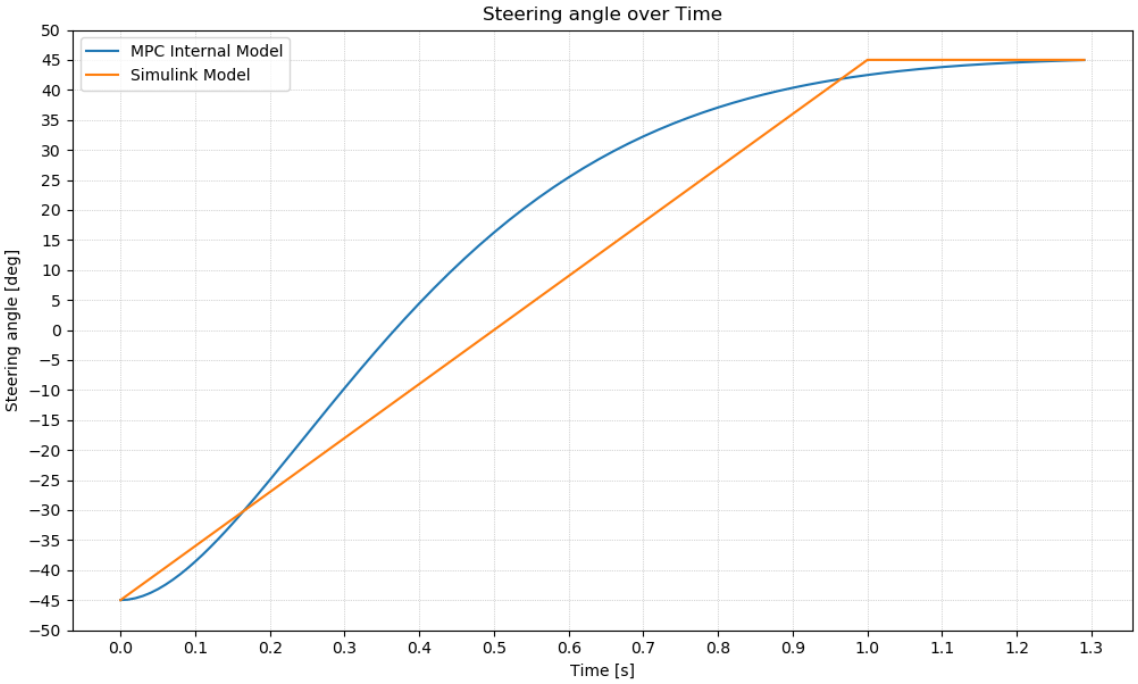


Figure B.5: Steering: Step response of internal model vs simulator

Appendix C

Discretizing the system for real-time calculations

Discretizing the system for real time calculations

Jørgen Vesterheim Knutsen

April 2024

Preface

This note has been created to explore different ways of discretizing dynamic systems. Model predictive control (MPC), is based on discrete time dynamic systems, which motivates the creation of this note. The methods discussed will also have applications for simulations, both in real time and pre-calculated.

There are two approaches that will be compared in this note: The analytical, exact solution, and a numerical approach.

1 Analytical solution

For a linear autoregressive moving average (ARMA) model, such as

$$\dot{x} = ax + bu$$

the solution to the differential equation is the sum of the autonomous response, i.e. the impulse response $h(t)$, and the convolution between the impulse response and the input-signal.

$$x(t) = h(t) + b \int_0^t h(t-\tau) \cdot u(\tau) d\tau$$

Here, $h(t)$ is the impulse-response, i.e. the autonomous part of the system. For this system the impulse response is given as:

$$h(t) = x_0 \cdot e^{at}$$

$$x(t) = x_0 \cdot e^{at} + b \int_0^t e^{a(t-\tau)} \cdot u(\tau) d\tau$$

Moving on to a multi-dimensional system to make it relevant for our bachelors thesis, we assume the general non-linear system has been linearized, and is expressed in the form:

$$\dot{x} = Ax + Bu$$

We assume that the system uses a zero-order-hold method for its input, and thus the input is continous, but piecewise constant.

$$u(t) = u(k), \quad kT \leq t < (k+1)T$$

The solution to this system in continous time is:

$$\mathbf{x}(t_f) = e^{A(t_f - t_0)} \cdot \mathbf{x}(t_0) + \int_{t_0}^{t_f} e^{A(t_f - \tau)} \cdot B\mathbf{u}(\tau) d\tau$$

We can say that initial time $t_0 = kT$, and the final time $t_f = (k+1)T$, resulting in the following:

$$\mathbf{x}(k+1) = e^{AT} \cdot \mathbf{x}(k) + \int_{kT}^{(k+1)T} e^{A((k+1)T - \tau)} \cdot B\mathbf{u}(\tau) d\tau$$

where:

$$\lambda = (k+1)T - \tau$$

$$d\lambda = -d\tau$$

This simplifies the equations a fair bit [6]:

$$\mathbf{x}(k+1) = e^{AT} \cdot \mathbf{x}(k) + \int_0^T e^{A\lambda} \cdot Bd\lambda \cdot \mathbf{u}(k)$$

$u(k)$ has been moved outside, because in the interval we are talking about, the input is held constant due to it's zero order hold nature.

Note: When substituting λ back to τ, we get the following: $T(k+1) - \tau$.

$$\lambda\big|_{\tau=kT}^{\tau=T(k+1)} = [T(k+1) - \tau]\big|_{\tau=kT}^{\tau=T(k+1)} = [T(k+1) - T(k+1)] - [T(k+1) - Tk] = T$$

The result is that the integral $\int_0^T e^{A\lambda} \cdot Bd\lambda$ remains a constant. If A is invertible, the integral may be calculate as [5]:

$$\int_0^T e^{A\lambda} \cdot Bd\lambda = A^{-1}[e^{At} - I]B$$

This holds for when the system is linear, meaning that in a non-linear system, the system would have to be linearized for every step.

If the system doesn not have an invertible A-matrix, we have to integrate. When it comes to the integration of the matrix, the following holds [7]:

$$\int_a^b A\,dx = \begin{bmatrix} \int_a^b a_{11}(x)\,dx & \cdots & \int_a^b a_{1n}(x)\,dx \\ \vdots & \ddots & \vdots \\ \int_a^b a_{m1}(x)\,dx & \cdots & \int_a^b a_{mn}(x)\,dx \end{bmatrix}$$

Matrix Exponentials

Any *nxn* system can be rewritten to a normal jordan form. One may obatin the Jordan Normal form by using the transofmation matrices, which are a vector of the

generalized eigenvectors [9]. Remember to transform back to the original coordinate system after running the calculations. This is done by multiplying with T^{-1} from the left.

$$J = TAT^{-1} = \begin{bmatrix} \lambda & 1 & & & \\ & \lambda & 1 & & \\ & & \lambda & & \\ & & & \ddots & \\ & & & & \lambda \end{bmatrix}$$

$$J_\lambda^{(n)} = \lambda I + N \ \text{, where } N = A - \lambda I$$

This implies [1]:

$$e^{J_\lambda^{(n)} t} = e^{(\lambda I + N)t} = e^{\lambda I t} e^{Nt}$$

$$e^{\lambda I t} = e^{\lambda t} I$$

$$e^{Nt} = I + Nt + N^2 \frac{t^2}{2!} + N^3 \frac{t^3}{3!} + \ldots = \begin{bmatrix} 1 & t & \frac{t^2}{2!} & \frac{t^3}{3!} & \ldots \\ & 1 & t & \frac{t^2}{2!} & \ldots \\ & & & \ddots & \\ 0 & & & & 1 \end{bmatrix}$$

$$J_\lambda^{(n)} = \lambda I + N \longrightarrow e^{J_\lambda^{(n)} t} = e^{(\lambda I + N)t} = I e^{\lambda t} e^{Nt}$$

$$= \mathrm{I} \begin{bmatrix} e^{\lambda t} & & & \\ & e^{\lambda t} & & \\ & & \ddots & \\ & & & e^{\lambda t} \end{bmatrix} \begin{bmatrix} 1 & t & \frac{t^2}{2!} & \frac{t^3}{3!} & \ldots \\ & 1 & t & \frac{t^2}{2!} & \ldots \\ & & \ddots & & \\ 0 & & & & 1 \end{bmatrix} = \begin{bmatrix} e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \frac{t^3}{3!}e^{\lambda t} & \ldots \\ & e^{\lambda t} & te^{\lambda t} & \frac{t^2}{2!}e^{\lambda t} & \ldots \\ & & \ddots & \ddots & \\ & & & \ddots & \vdots \\ 0 & & & & e^{\lambda t} \end{bmatrix}$$

1.1 Full procedure - analytical solution

Given the system of equations, linearized and written in the state-space representation:

$$\dot{\mathbf{x}} = \mathbf{f}(x_1, x_2, \ldots) \approx \dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

Transforming the system with the generalized eigenvectors:

$$V\dot{\mathbf{x}} = JV\mathbf{x} + VB\mathbf{u}$$

Substituting: $\widetilde{\dot{x}} = V\dot{x}$, $\widetilde{x} = Vx$, $\widetilde{B} = VB$

$$\dot{\widetilde{\mathbf{x}}} = J\widetilde{\mathbf{x}} + \widetilde{B}\mathbf{u}$$

This leads to the modified equation:

$$\widetilde{\mathbf{x}}(k+1) = e^{JT} \cdot \widetilde{\mathbf{x}}(k) + \int_0^T e^{J\lambda} \cdot \widetilde{B} d\lambda \cdot \mathbf{u}(k)$$

If the A matrix is invertible:

$$\int_0^T e^A B d\lambda = A^{-1}[e^{AT} - I_n]B$$

Otherwise, do the matrix exponentials and integration detailed earlier in this note.

Finally,

$$\mathbf{x}(k+1) = V^{-1}\widetilde{\mathbf{x}}(k+1)$$

This solution is the exact, analytical solution in discrete time. If this procedure is done on a linear system, no information will be lost, because it is still a *true* representation on the system. This procedure is, however fairly complicated, and may require a lot of computation.

2 Numerical solution

A widely used, and simpler approach, is the numerical approach to the calculating the system. For the purpose of this, the Runge-Kutta methods will be used. "The Runge-Kutta methods are a familiy of implicit and explicit iteratice methods" [2] used to solve systems of differential equations.

2.1 First order Runge-Kutta

Runge-Kutta of order 1, also known as the Euler Methods [2], uses the previous value, a timestep, and the derivative to calculate the next step. The Euler Method can be done as an explicit or implicit verison. Here we will focus only on the explicit method, because it is easier to calculate. Further, the system will be less stable than for the implicit method. For the model used in the MPC, it will likely be more beneficial to have a less stable model, than one which is more stable than in reality.

For the following differential equation, using forward Euler (explicit method):

$$\dot{x} = -3x$$

$$x[k+1] = x[k] + h \cdot \dot{x} = x[k] - h \cdot 3x$$

The stepsize for numerical methods is important for the stability of the solution, as demonstrated in figure 1. Forward Euler may move the poles in marginally stable systems to unstable discrete poles when discretizing. This may lead to numerical instability. The effect of this is based on the step-size and the prediction horizon.

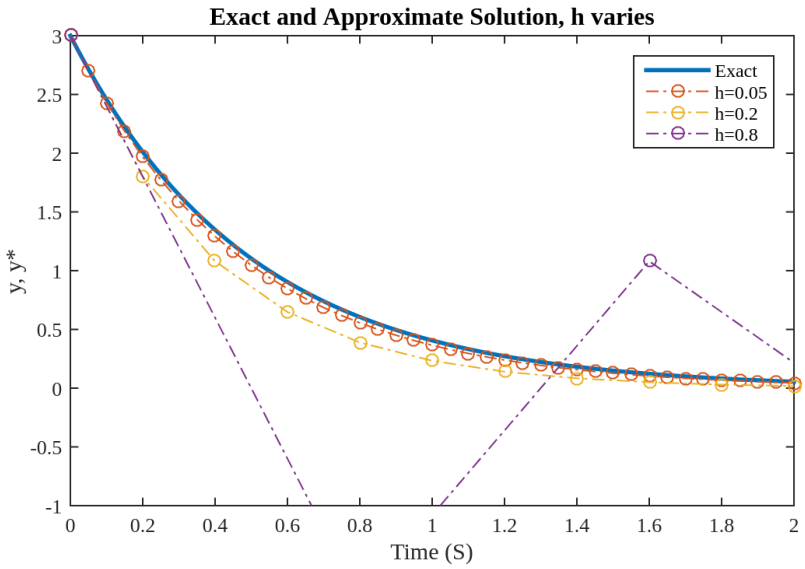


Figure 1: [3]

For a general, linearized set of differential equations,

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

the system may be written in its descrete form as

$$\mathbf{x}[k+1] = \mathbf{x}[k] + h \cdot \dot{\mathbf{x}} = \mathbf{x}[k] + h \cdot (A\mathbf{x}[k] + B\mathbf{u}[k]),$$

2.2 2nd order Runge-Kutta

The midpoint method is a specific type of RK2. The method calculates two steps ahead, and uses the midpoint of these to predict the next step. It is often seen to have better convergence than RK1. [4][8].

It is done in the following way:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$

$$\begin{aligned}
\mathbf{k}_1 &= A\mathbf{x}[k] + B\mathbf{u}[k] \\
\mathbf{x}_{\text{mid}} &= \mathbf{x}[k] + \frac{h}{2}\mathbf{k}_1 \\
\mathbf{k}_2 &= A\mathbf{x}_{\text{mid}} + B\mathbf{u}[k] \\
\mathbf{x}[k+1] &= x[k] + h \cdot \mathbf{k}_2
\end{aligned}$$

Discretizing using the midpoint method leads to a discrete system that is more stable than the original continuous system. This is because it averages out the change, and in that way dampens changes. This is demonstrated in Figure 3, where convergence compare to the exact solution is shown.

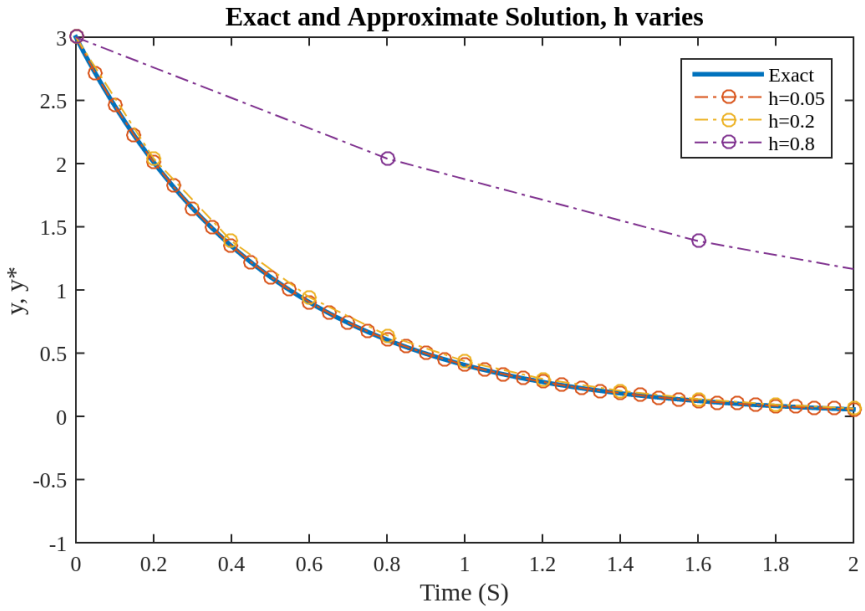


Figure 2: Convergence RK2[4]

After testing, the decision has been made to use a numerical approach, and use RK1 to compute the trajectory of the system. This was chosen because of the faster calculation, less implementation-time, and good results for the system at hand. Additionally, the increased change of instability from Forward Euler could make the MPC more averse to making large changes, which may be another positive. The simulated system, which will not be explained further in this note, is the following:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \\ \dot{F}_e \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\psi) \\ v \cdot \sin(\psi) \\ \frac{v}{L} \cdot \tan(\delta) \\ \frac{F}{m} \\ -\frac{1}{\tau_1} \cdot F_e + \frac{1}{\tau_1} \cdot u_1 \\ \dot{\delta} \\ -2\zeta\omega\dot{\delta} - \omega^2\delta + u_2 \end{bmatrix}$$

For a ten second period, with right turn given as input from 0-5 sec, and left turn from 5-10 sec, with a throttle input at 1-3 sec:

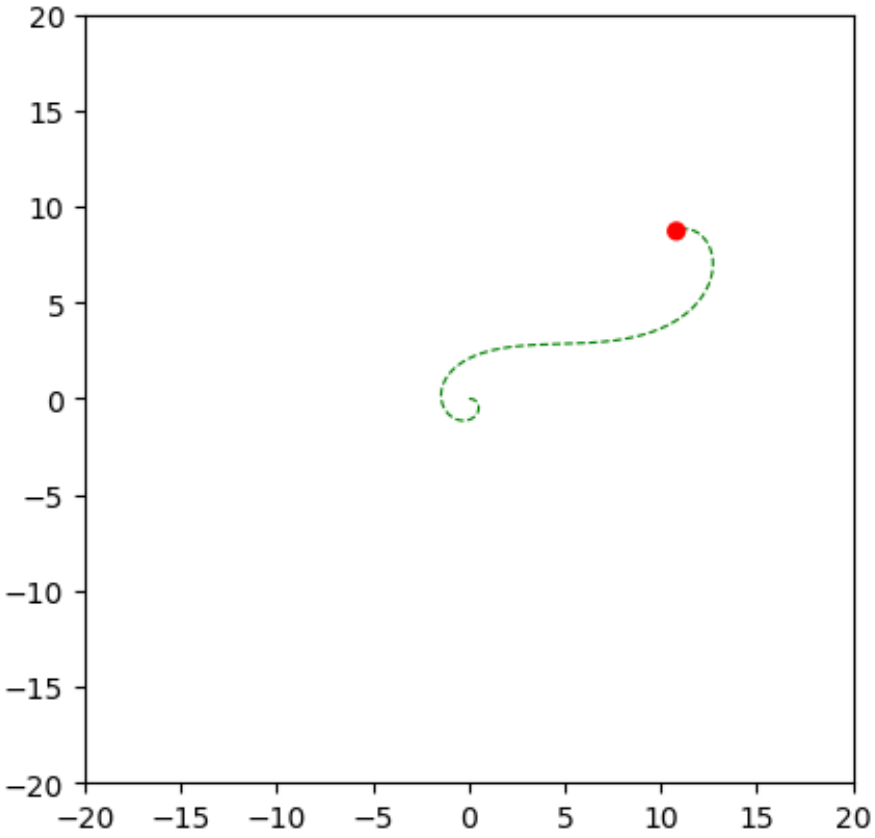


Figure 3: Simulation results

Bibliography

[1] Contributors to Wikimedia projects. *Matrix exponential - Wikipedia*. [Online; accessed 4. Apr. 2024]. Nov. 2023. URL: https://en.wikipedia.org/w/index.php?title=Matrix_exponential&oldid=1184902135.

[2] Contributors to Wikimedia projects. *Runge–Kutta methods - Wikipedia*. [Online; accessed 4. Apr. 2024]. Mar. 2024. URL: https://en.wikipedia.org/w/index.php?title=Runge-Kutta_methods&oldid=1214270720.

[3] Swarthmore College Erik Cheever. *Euler's Method (First Order Runge-Kutta)*. [Online; accessed 4. Apr. 2024]. Mar. 2019. URL: https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html.

[4] Swarthmore College Erik Cheever. *Second Order Runge-Kutta*. [Online; accessed 4. Apr. 2024]. Mar. 2019. URL: https://lpsa.swarthmore.edu/NumInt/NumIntSecond.html.

[5] M. Sami Fadali and Antonio Visioli. *Digital Control Engineering: Analysis and Design*. English. 3rd ed. Academic Press, 2019. ISBN: 9780128144343.

[6] Lino Guzzella. *Discrete Time Control Systems*. Online. Lecture slides for Digital Control Systems course at ETH Zurich. 2013. URL: https://idsc.ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Digital-Control-Systems/Slides_DigReg_2013.pdf.

[7] *Integrating a matrix*. [Online; accessed 4. Apr. 2024]. Apr. 2024. URL: https://math.stackexchange.com/questions/450560/integrating-a-matrix.

[8] Libretexts. '8.03: Runge-Kutta 2nd-Order Method for Solving Ordinary Differential Equations'. In: *Mathematics LibreTexts* (Oct. 2023). URL: https://math.libretexts.org/Workbench/Numerical_Methods_with_Applications_(Kaw)/8%3A_Ordinary_Differential_Equations/8.03%3A_Runge-Kutta_2nd-Order_Method_for_Solving_Ordinary_Differential_Equations.

[9] Northwestern University. *Notes on Jordan Form*. Lecture notes. Summer term. 2015.

Appendix D

Thesis Poster

Lone Wolf ATV: Dynamic Model and Model Predictive Control

Eskil Bakken *Eirik N. Bjørnevik* *Jørgen V. Knutsen* *Vegard Ohren*

Department of Engineering Cybernetics

NTNU

KONGSBERG

Background and Thesis Statement

Project Lone Wolf is a multi-disciplinary project within KONGSBERG's business area, Kongsberg Defence & Aerospace. The project is dedicated to designing and constructing an autonomous all-terrain vehicle (ATV) capable of obstacle avoidance. [1] The thesis statement for the project is:

"Development of a dynamic model of the Lone Wolf ATV and integration of a complementary control algorithm"

Dynamic Model

The dynamic model of the ATV is derived using first principles, advanced tire models and approximations for the engine and drivetrain mechanics. The model, simplified as a bicyce model, describes the ATV's dynamics within the XY-plane, capturing movements along the x-axis, y-axis, and yaw angle, Ψ.

The model features adjustable parameters for later refinement using real-world data, where employing grey box methods can enhance its fidelity to the actual system dynamics.

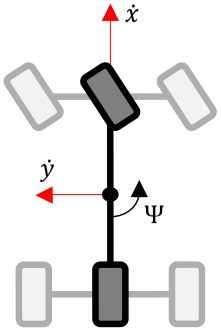


Figure 1: Dynamic Model, simplified

Model Predictive Control

A Model Predictive Controller (MPC) was developed to enable a vehicle to follow a predefined trajectory by predicting future states and optimizing throttle and steering adjustments. The MPC uses a simplified model of vehicle dynamics and an internal state estimator to account for unmeasured system variables.

The controller's objective is to minimize trajectory tracking errors and control effort, quantified by a cost function. To address the nonlinear optimization involved, the controller utilizes the IPOPT (Interior Point OPTimizer) algorithm, known for its effectiveness in nonlinear optimization.

Project Overview

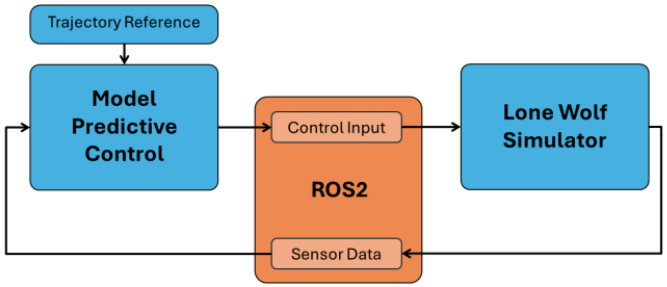


Figure 2: Project Overview

Figure 2 illustrates how the simulator and the MPC developed in the thesis communicate using the ROS2 framework. This is done to mimic the software architecture of the actual Lone Wolf and facilitate easy integration into the real system at a later occasion.

Simulator in Simulink

The dynamic model of the ATV was implemented in the ROS2-compatible software Simulink. A graphical user interface (GUI) simulates the vehicle in real-time, shown in Figure 3. It plots the vehicle's movement in the XY-plane, the reference trajectory, and the velocity, steering angle, and deviation from the reference over time.

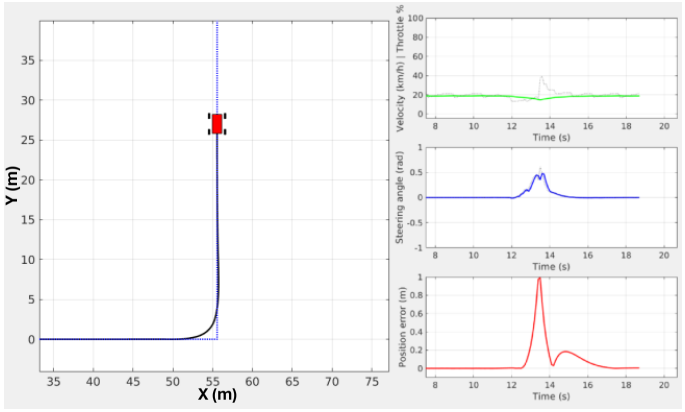


Figure 3: Screenshot from the Lone Wolf simulator in use

Results and Conclusion

The MPC has been tested in the simulator with various reference trajectories, as shown in Figure 4. The MPC proved to be effective in dynamically adjusting the throttle and steering angle to closely match the predetermined trajectories, demonstrating robust performance. This confirms the system's potential for real-world applications.

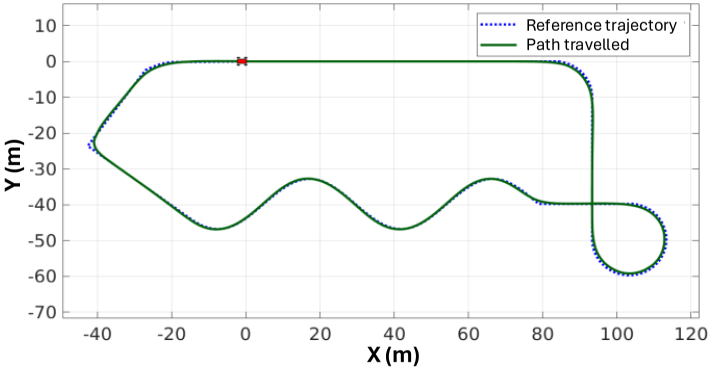


Figure 4: Racetrack reference

Further Work

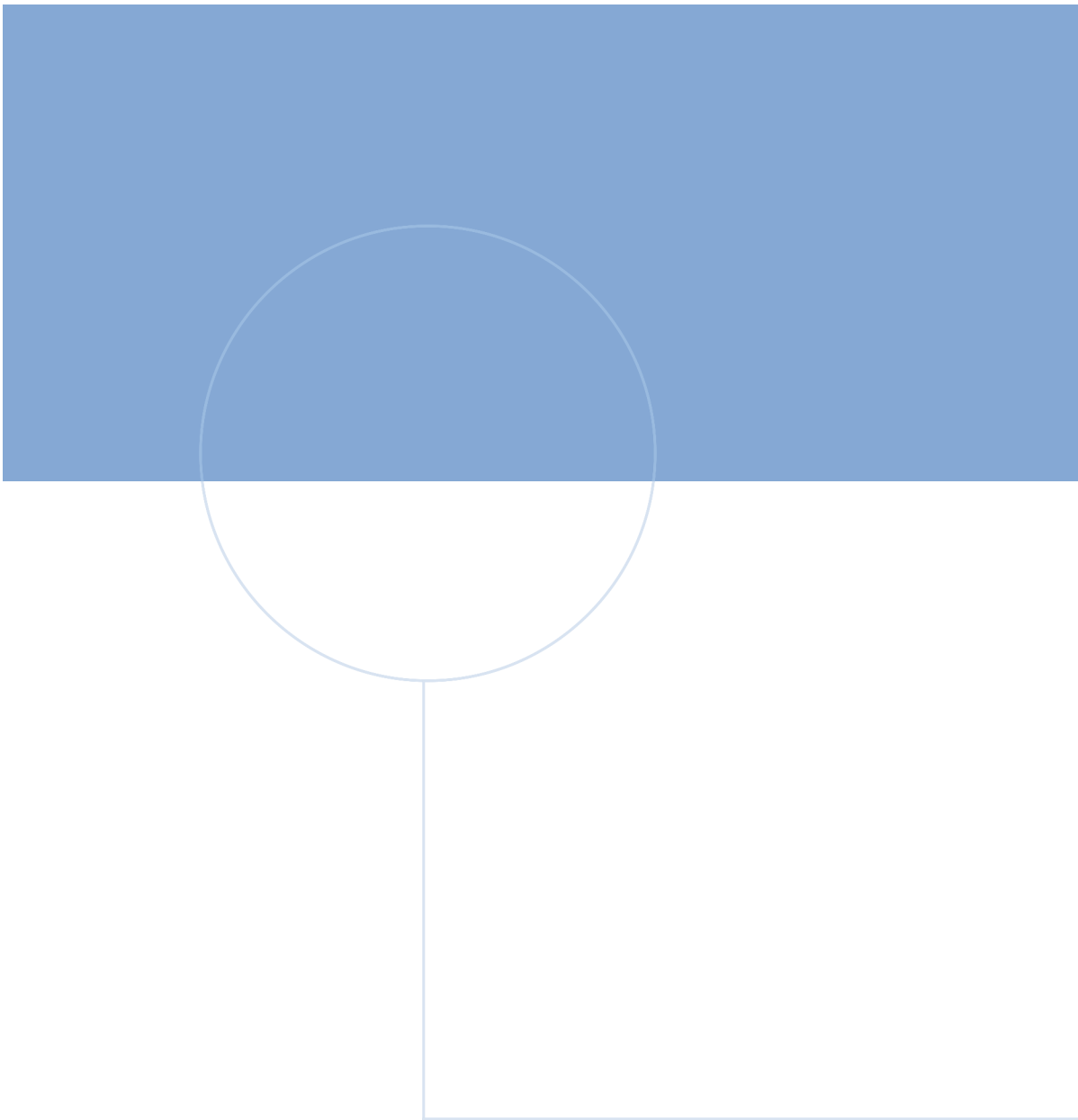
The work underlaid throughout this thesis has made a foundation for further development on Project Lone Wolf. Recommended tasks for the future, building on the work done in this thesis, include:

- Data collection for model improvements and validation
- Path-/trajectory planning algorithms
- Implementation of an Extended Kalman Filter for signal processing

References

[1] *Join the pack,* KONGSBERG, 2024, www.kongsberg.com/careers/summer-interns/lone-wolf/

[2] E. Bakken, E.N. Bjørnevik, J.V. Knutsen, V. Ohren, *«Lone Wolf ATV: Dynamic Model and Model Predictive Control»*, 2024



NTNU
Norwegian University of
Science and Technology

KONGSBERG