ÓBUDAI EGYETEM

Bánki Donát Gépész és Biztonságtechnikai Mérnöki Kar Gépészeti és Biztonságtudományi Intézet

Development and Evaluation of an Emergency Collision Avoidance System for Autonomous Vehicles

Dr. Szakács Tamás

OE-BGK

2033/12/15

Abd-Alfatah Alodainy

UA3AGW

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Bánki Donát Gépész és Biztonságtechnikai
Mérnöki Kar Gépészeti és
Biztonságtudományi Intézet

STUDENT DECLARATION

I, the undersigned student, declare that this thesis/thesis is the result of my own work, and that I have identified the literature and tools used. The results contained in the thesis/thesis may be used by the university and the institution issuing the assignment for its own purposes without charge, subject to any confidentiality restrictions.

Budapest, 20_____

_____

Student signature

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbuda University
Bánki Donát Faculty of Mechanical and Safety Engineering
Institute of Mechatronics and Vehicle Engineering

THESIS ASSIGNMENT

Student's surname, forename (s): Abd-Alfatah Sadeq Ahmed Alodainy
Thesis number: SZD2309220915262242UA3AGW
Registration number: T010365/FI12904/B **Neptun code:** UA3AGW

Branch of study: Mechatronical Engineering (BSc) (in English)
Specialization: Mechatronical Engineering (BSc) (in English) - Industrial Robot Systems

Thesis title: Development and Evaluation of an Emergency Collision Avoidance System for Autonomous Vehicles
Task description:
1. Design of basic and emergency operating modes for autonomous vehicles and surface robots.
2. Development of navigation methods used in basic and emergency modes using ArduPilot-based solutions.
3. Solving safety issues such as collision avoidance, design and construction of redundant on-board systems, and integration of ADAS sensors.
4. Simulation-based experiments using CarSim and MATLAB/Simulink to test the effectiveness and feasibility of the emergency collision avoidance system.
5. Comparison and analysis of the simulation results to assess the collision avoidance capabilities, trajectory tracking accuracy, and stability of the simulated system.
6. Discussion of the implications and potential applications of the developed system in real autonomous vehicles and surface robots.
7. Recommendations for further improvements and future research directions based on the simulation findings.

Institutional consultant's name: Dr. Tamás Szakács

The limitation period of the theme issued: 30.10.2025.
Deadline for submission of Thesis: 15.12.2023.

The thesis is: Not secreted.

Issued: Budapest, 30.10.2023.

Bánki Donát Gépész és Biztonságtechnikai Mérnöki Kar
ÓBUDAI EGYETEM
Mechatronikai és Járműtechnikai Intézet
P.H.

.....
Director of Institute

The Thesis is suitable for submission:

Szakács Tamás
.....
Institutional consultant

.....
External consultant

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Bánki Donát Gépész és Biztonságtechnikai
Mérnöki Kar Gépészeti és
Biztonságtudományi Intézet

STUDENT DECLARATION

I, the undersigned student, declare that this thesis/thesis is the result of my own work, and that I have identified the literature and tools used. The results contained in the thesis/thesis may be used by the university and the institution issuing the assignment for its own purposes without charge, subject to any confidentiality restrictions.

Budapest, 20 23/12/27

Student signature

CONTENTS

1 Introduction **4**

1.1 Motivation 4

1.2 Objective 4

2 Formulation of the Problem to Solve **6**

2.1 Systems for preventing collisions in a single vehicle 6

2.1.1 Lateral control 8

2.1.2 Longitudinal control 10

2.1.3 Combined lateral and longitudinal controller 11

2.1.4 Thesis Contribution 13

3 Analysis of the Problem, Development of the Specification **14**

3.1 Lateral vehicle dynamics 14

3.1.1 Two Degree of freedom (DOF) 14

3.1.2 Three DOF 18

3.1.3 14-DOF vehicle dyanmic 22

3.2 Cornering stiffness estimation 32

3.3 General Longitudinal vehicle dynamic 33

4 Analysis of Possible Solutions **37**

4.1 Selection Steering control algorithm 37

4.1.1 Proportional, integral, derivative controller (PID) steering controller 37

4.1.2 Stanley controller 39

4.1.3 Model predictive controller (MPC) controller 41

4.2 Selection of longitudinal controllers 45

4.2.1 PID speed and brake controller 45

4.2.2 MPC-based Speed and Brake Control Algorithm 49

4.3 Selection of path planning algorithm 50

4.3.1 Simple Dynamic Path Adjustment 51

4.3.2 Spline Algorithm 52

4.3.3 Bezier Planning Algorithm 53
4.3.4 Dijkstra's Algorithm 54
4.3.5 A* path planning 57

5 Description of Detailed Specification 60
5.1 Vehicle parameters setting 60
5.2 Tune-able parameters and gains setting for steering and speed control . 67
5.2.1 Steering controllers 67
5.2.2 Speed controllers 68
5.3 Tunable Parameters for Path Planning and Controllers 68
5.3.1 Cubic Spline Interpolation 68
5.3.2 Bezier Planning 68

6 Description of Work Phases during Design and Lessons Learned 70
6.1 Challenges and Potential Pitfalls 70
6.1.1 Tuning control parameters 70
6.1.2 Tuning Path Planning Parameters 71
6.1.3 Controller-Path Planning Coordination 71
6.2 MATLAB Useful Tools 71
6.2.1 Virtual Vehicle Composer 71
6.2.2 Driving Scenario Designer 74
6.2.3 Control System Tool box 75
6.2.4 Parameter Estimation 76
6.3 Opportunities for Self-Checking Methods and Testing 77

7 Description of Implementation 82
7.1 Models Build in Simulink environment 82
7.1.1 Longitudinal controller implementation 83
7.1.2 Lateral controllers implementation 85
7.1.3 Vehicle Visualization in Simulink 91
7.2 Integrating Simulink models with CarMaker 92
7.2.1 CarMaker environment setup 93
7.2.2 Connecting Simulink model to CarMaker 95

8 Presentation of Results and Testing **97**
8.1 MATLAB/Simulink Model Results 97
8.1.1 Path Tracking Results 97
8.1.2 Longitudinal results 100
8.2 CarMaker testing and results 101
8.2.1 Lateral controllers evaluation(testing results) 104
8.2.2 Longitudinal Controller 119

9 Analysis of the Implementation, Summary of Application, and Possibilities for Further Development **122**
9.1 Opportunities for Further Development 123

Bibliography **125**

Acronyms **130**

List of Figures **131**

Appendices **135**

Appendix A Code Snippets **136**
A.1 MATLAB codes 136
A.1.1 Vehicle Visualization Simulink 136
A.1.2 PID speed regulator 138
A.1.3 MPC speed regulators 140
A.1.4 cross-track error (lateral error) 141
A.1.5 Stanley steering 142
A.1.6 MPC state space model 142
A.1.7 instantaneous path generation 144
A.1.8 Spline based path planning 145
A.1.9 Bezeir path planning 147
A.1.10 Desired speed adjusting 152
A.2 PID speed and brake regulators 154

Appendix B Models pictures **156**

1. INTRODUCTION

1.1 Motivation

In the constantly changing field of autonomous vehicle technology, safety is still the top priority. Ensuring autonomous vehicles can navigate complex environments and react to unforeseen obstacles becomes increasingly important as they become a vital part of our transportation ecosystem. The requirement for sophisticated emergency collision avoidance systems is at the heart of this problem.

This thesis is driven by the urgent need to give autonomous cars the capacity to respond quickly and efficiently to unforeseen obstacles and possible collision situations. Even though these cars have the potential to completely transform transportation by lowering accident rates and increasing traffic flow, they still need to have strong collision avoidance systems in place to protect drivers and other road users as well as passengers. The potential to save lives and drastically lower the societal and financial costs related to traffic accidents emphasises how urgent this endeavour is.

1.2 Objective

This thesis's main goal is to create, develop, and assess a sophisticated emergency collision avoidance system specifically for self-driving cars. This system will be painstakingly designed to allow self-driving cars to react appropriately to unforeseen obstacles or circumstances, reducing the likelihood of collisions and improving overall safety.

In order to accomplish this main goal, we will start a thorough investigation that entails:

- The development of basic and emergency operating modes for autonomous vehicles.
- Addressing safety challenges through the design of collision avoidance strategies and integration of redundant on-board systems.
- Discussing different controlling techniques and algorithms
- Seamless integration of Advanced Driver Assistance Systems (ADAS).

- Extensive simulation experiments conducted primarily within the MATLAB/Simulink environment to validate the feasibility and effectiveness of the emergency collision avoidance system.
- Use the system designed in MATLAB and validate in a Real Time simulation (RTL) environment

Eventually, the goal of this thesis is to advance the field of autonomous vehicle safety protocols and open the door to safer and more efficient transportation in a society that is becoming more and more automated.

2. FORMULATION OF THE PROBLEM TO SOLVE

In our daily life, we have seen a lot of accidents happening in the street these accidents might be because of the driver's attention, low rate reaction, or wrong behavior, in this thesis I will discuss the possibilities to solve this problem in autonomous driving. The effort of maintaining vehicle safety on the road never ends! The issue of vehicle safety has grown increasingly important in contemporary transport systems due to the swift rise in traffic density. The World Health Organisation states that traffic for youth and young adults between the ages of 5 and 29, injuries rank as the primary cause of mortality and the eighth the main cause of death across all age categories. An estimated 1.35 million people are injured in automobile accidents, which account for 3% of most nations' annual gross domestic output[35]. In figure 2.1 we can see the number of fatalities which show the decrease due to more advanced technology

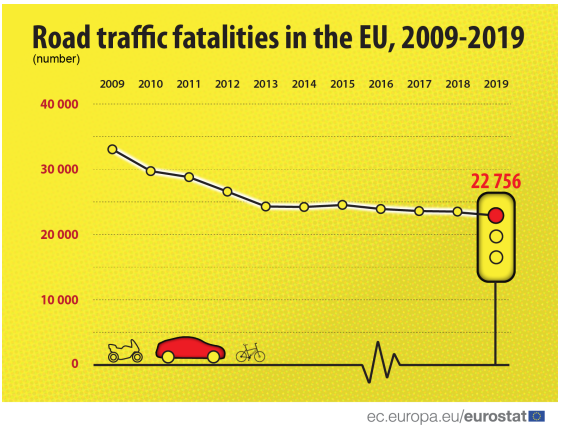


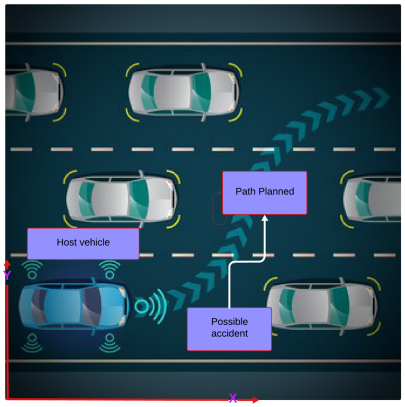
Figure 2.1: Road accidents: number of fatalities in the EU, 2009 - 2019 [10]

Many experts think automated and connected cars will offer a viable solution to the safety issues in transportation in the future. Several studies also revealed that adding Advanced Driver Assistance Systems (ADAS) to cars could potentially reduce the roughly 40–50 % of traffic accidents that result from driver error[32].

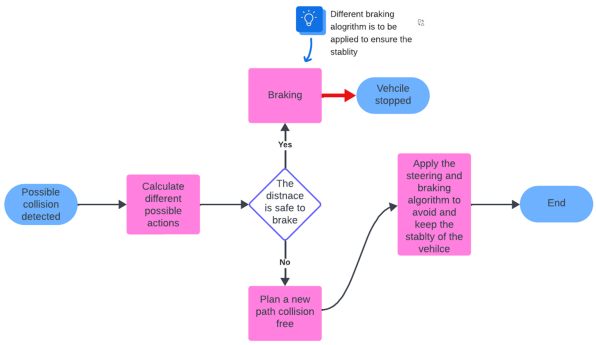
2.1 Systems for preventing collisions in a single vehicle

The most obvious way to increase the safety performance of intelligent transportation systems is to prevent collisions between moving vehicles and other obstacles. As such,

related technologies have long been the focus of intensive research in the domains of autonomous robotics and intelligent vehicles. Since autonomous robots have been the subject of research for almost thirty years, research on intelligent vehicles should profit somewhat from the lessons gained in this area.



(a) A possible scenario where a collision avoidance system is necessary



(b) Decision-making for obstacle avoidance

Figure 2.2: Collision avoidance scenario and decision making tree for the proposed method

On the other hand, autonomous robots typically use the collision avoidance method, which computes a collision-free path in any direction by treating the robot as a simplified point-mass motion model[30]. However, it is not hard to understand that manoeuvrability plays a crucial role in lateral collision avoidance systems for an intelligent vehicle.

The chance of a collision between a moving vehicle and an obstruction increases as speed does, and the resulting damage is also increased. The collision avoidance controller for intelligent vehicles must take into account more kinematics and dynamics factors in comparison to the collision avoidance system for robots because it must navigate more complicated road conditions.

The goal of a collision avoidance system is to create a vehicle control algorithm that will prevent an impending collision. For collision avoidance manoeuvres, two actuation configuration options are available: lateral control (only active steering) and longitudinal control (emergency braking only) [18]. The two manoeuvres mentioned above are depicted in Figure2.2.

Most of the time, braking is the driver's attempt to prevent a collision. In actuality, at low speeds, the brake-based collision avoidance system is a superior option. The efficacy of autonomous emergency braking (AEB) and forward collision warning (FCW) in enhancing driving safety, as well as the drivers' subjective assessment, has been demonstrated by experimental research and market feedback[3]. However, longitudinal collision avoidance controllers are not very helpful in preventing head-on or rear-end collisions when there is not enough space between moving cars at higher speeds (such as over 60 km/h). In a rear-end collision, over 80% of drivers failed to steer, according to a thorough study of 635 incidents in the GIDAS database. Subsequent investigation revealed that the space requirements for an evasion manoeuvre had been met in 24% of these incidents, indicating that steering would have been the wiser course of action [29]. Under these conditions, evasion-based CAS manoeuvres are more appropriate if there is room in a nearby lane; the manoeuvres can be executed in a shorter distance than what is needed to bring the vehicle to a stop.

2.1.1 Lateral control

The integration of suitable path-planning and tracking modules can enable autonomous driving systems to integrate collision avoidance capability for vehicles, as demonstrated in the lateral motion control studies. By utilising data from onboard sensors regarding the road ahead and other vehicles, a path-planning module that accounts for the vehicle dynamics in the given traffic conditions can be built to prevent crashes. A path-tracking controller, which can ensure the vehicle's tracking capability to ensure that the vehicle is followed safely, should also implement the collision avoidance

objectives. Path-planning and tracking for lateral collision avoidance are difficult in high-speed driving conditions for a number of reasons. Actuator saturation and maximum friction forces, as well as physical limitations resulting from vehicle dynamics, pose the first challenge to trajectory feasibility. The existence of unstable equilibrium points in the vehicle yaw dynamics, which depend on the tyre properties, steering angle, and vehicle speed, presents another difficulty. These dynamically unstable areas must be avoided in the path-tracking design. Finally, in order to respond quickly to changes in the environment, algorithm computation times must be brief and fast.

A range of cutting-edge technologies, including sensing, control, decision-making, and other critical technologies, are needed for the development and implementation of lateral collision avoidance systems for intelligent vehicles. A hierarchical structure of collision avoidance systems can be divided into four layers based on different stages of collision avoidance: environmental perception, decision-making, path-planning, and path-tracking. Of these, path-planning and path-tracking are the most fundamental and important factors influencing the safety and comfort performance of the collision avoidance system, as shown in Fig2.3.

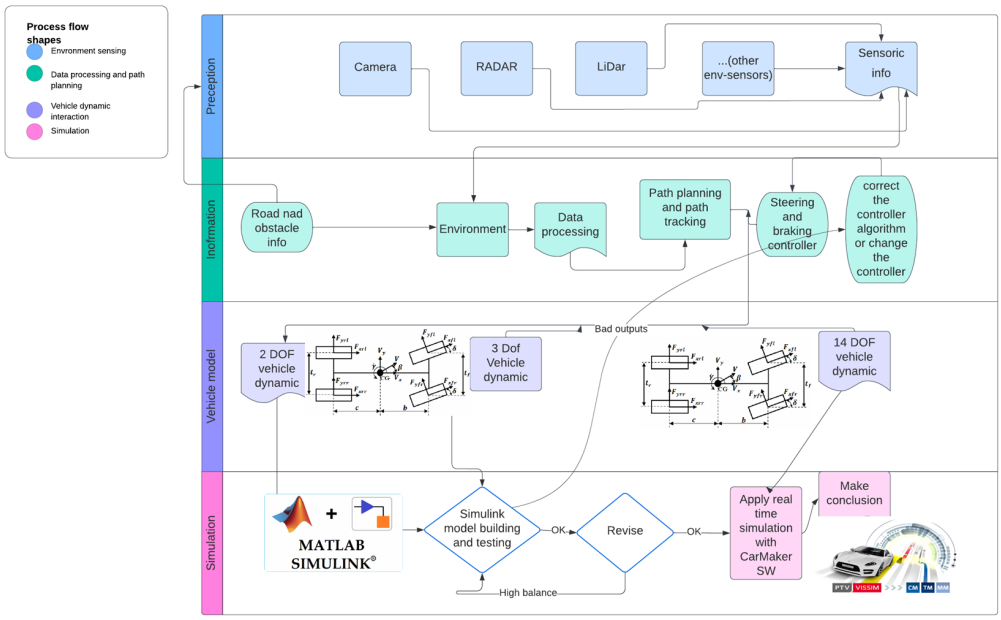


Figure 2.3: The architecture of an Emergency Collision Avoidance System work

As can be seen from the above, designing collision avoidance algorithms for high-speed intelligent cars in the direction of lateral motion while adhering to kinematics and vehicle dynamics constraints remains a difficult task.

2.1.2 Longitudinal control

The most common kind of collision in the longitudinal motion control studies, according to accident analysis, is a rear-end collision. As illustrated in Fig.2.4, rear-end collisions actually account for 29% of all traffic accidents in the United States, with an estimated 1.7 million occurring annually [2]. These crashes account for a sizable percentage of highway accidents and cause a considerable number of injuries and fatalities annually, with an estimated 1,700 people dying and another 500,000 being injured. The rear-end collision has two primary causes:

- a tardy or nonexistent response from the motorist (for example, approaching a traffic bottleneck while paying inattentive attention);
- inadequate headway, which occurs when a driver fails to maintain enough distance from the car in front, leaving them with little time to react if it stops abruptly.

Many automotive active safety systems currently on the market are made to lessen the chance of a collision in the direction of longitudinal motion. For instance, present and upcoming passenger cars are equipped with features like Forward Collision Warning (FCW), Precrash Brake Assist (PBA), Autonomous Emergency Braking (AEB), and Anti lock Braking System (ABS). The FCW system uses visual, auditory, and/or tactile cues at the very last second before a collision is likely to occur to alert the driver to an approaching collision and give them enough time to make corrections[9]. When a car applies its brakes forcefully, it can cause wheel lock and cause an accident on dry or wet roads. By preventing the wheels from locking up, an ABS helps cars maintain their ability to steer while applying maximum braking force to shorten stopping distances [34]. About half of all drivers do not apply the brakes quickly enough or forcefully enough to fully utilise their vehicle's braking power when making emergency stops[17].

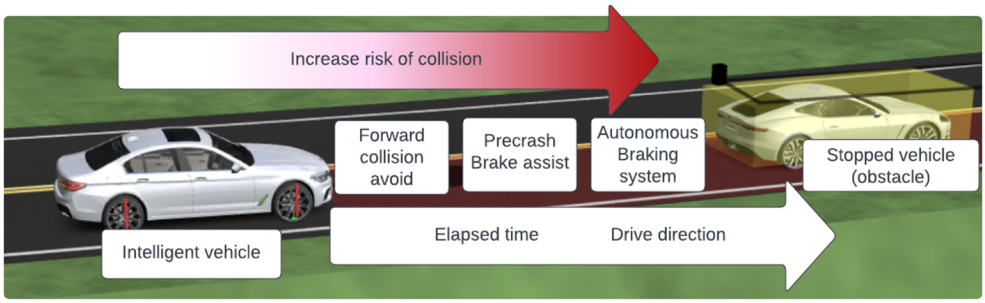


Figure 2.4: Longitudinal controller for Emergency collision system

When the driver applies the brake, PBA is activated, which occurs when the car detects an emergency braking situation and increases the amount of braking input. Lastly, AEB warns a driver of an impending collision and assists him in applying the vehicle's maximum braking force [9]. Even if a critical situation arises and no human intervention is made, the most basic AEB systems function at low speeds to prevent or lessen the severity of collisions by automatically adding to the vehicle's braking deceleration. As illustrated in figure 2.4, more advanced systems operate over a larger speed range and are activated last, closest to the collision.

2.1.3 Combined lateral and longitudinal controller

Using the vehicle's available actuators sensibly and effectively to avoid collisions is one of the most difficult tasks for a collision avoidance system. ABS is the primary actuation used in the majority of collision avoidance research projects to regulate the longitudinal dynamics of a car. Even though the car has applied all of its brakes and utilised all of the road friction, it can still be challenging to avoid colliding with an obstacle—like a deer—when it suddenly materialises in the middle of the road during emergency situations [11]. Thus, adding more actuators to the lateral control system could enhance the collision avoidance system's functionality.

Research on preventing collisions has unequivocally shown that steering is more effective at higher speeds and braking is more effective at lower speeds [28]. This does not imply, however, that the most effective method of preventing collisions is always provided by pure braking or pure steering. The emergency brake may cause a vehicle to lose stability due to complicated traffic conditions and different road frictions, but lateral

motion control can help with this issue when designing a longitudinal collision avoidance control strategy. Because of this, combined steering and braking may be a more effective strategy to prevent collisions over a wider speed range, particularly in situations where the dynamics are nonlinear and the vehicle speed is high [8]. Using some of the active safety systems currently on the market, accidents can be prevented or at least somewhat mitigated by automatic steering or braking. However, mass-produced vehicles still lack a collision avoidance system that combines lateral and longitudinal control technologies. This is because these systems are typically designed to handle particular accident types, for which steering or braking may be an option. The following difficulties still affect the lateral and longitudinal motion control technologies used in collision avoidance.

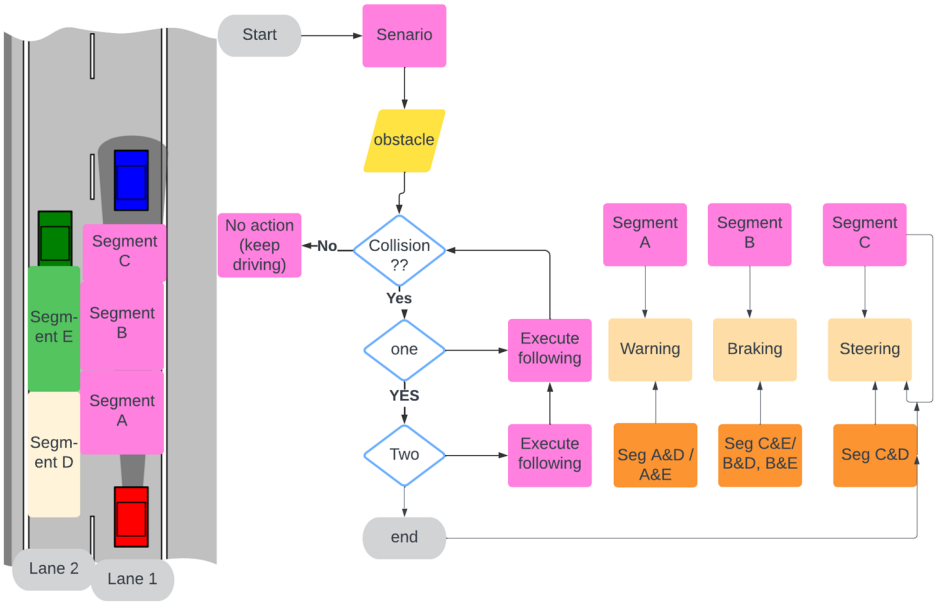


Figure 2.5: Combined controller logic

As seen in Fig. 2.5, the first challenge this technology faces is determining when to intervene with automatic steering and when to intervene with automatic braking. The ”threat assessment of traffic scenarios” is the basis for decisions about when and how to help the driver avoid collisions [6]. It primarily depends on the collision avoidance system’s perception, comprehension, and control over the surrounding traffic condi-

tions. Finding the last instance prior to a potential conflict scenario is another difficulty, although this can be done by examining the driving patterns of seasoned drivers in high-stress situations. The decision is much more nuanced because it is influenced when a collision cannot be prevented but can only be mitigated.

The design of controllers at the vehicle level, a crucial part of the collision avoidance system, presents another difficulty in carrying out the collision avoidance objectives in an efficient manner. According to the literature currently in publication, the majority of studies on vehicle control have historically focused on two entirely different motion directions, giving vehicle characteristics like dynamic coupling and tyre force coupling little thought.

2.1.4 Thesis Contribution

In this thesis, three different types of vehicle dynamics are discussed: 2-DOF, 3-DOF, and 14-DOF. The study involves summarizing existing techniques in emergency collision avoidance systems, providing a comparative analysis of these techniques.

A new approach to collision avoidance is implemented, starting with lateral control. Various controllers are discussed and analyzed to highlight the pros and cons of each. Efforts are made to enhance the controllability of these lateral controllers. Subsequently, a longitudinal controller is implemented, and its performance is analyzed, showcasing potential improvements along with the associated advantages and disadvantages.

After constructing all the necessary vehicle dynamic models in MATLAB Simulink, testing is initially conducted in the MATLAB/Simulink environment. The models are then adapted for real-time simulation, and the results are validated using the real-time simulation software CarMaker.

Additionally, the thesis explores path planning strategies for collision avoidance. Various path planning algorithms are investigated, and their effectiveness in avoiding collisions is analyzed. The integration of path planning with lateral and longitudinal control strategies is explored to enhance the overall collision avoidance system.

3. ANALYSIS OF THE PROBLEM, DEVELOPMENT OF THE SPECIFICATION

3.1 Lateral vehicle dynamics

As stated, In our problem solving I will explore three models for vehicle dynamics starting with two DOF, then three DOF, and lastly I will use a 14 DOF vehicle dynamic to have a robust results from the simulation that will be conducted in MATLAB Simulink along with the real time simulation CarMaker.

3.1.1 Two DOF

In this model the vehicle is considered as a bicycle that has one front and one rear wheel, the equations that I considered when thinking of a 2 DOF assuming that the vehicle runs in ideal conditions is the following

Kinematic model

The Kinematic model for 2-DOF can be simplified [27] as shown in the following table3.1

Table 3.1: Summary of Kinematic Model Equations

Symbol	Description	Equation
X	Global axis coordinate	$\dot{X} = V\cos(\gamma + \alpha)$
Y	Global axis coordinate	$Y_G = V\sin(\gamma + \alpha)$
γ	Yaw angle; orientation angle of vehicle with respect to global axis	$\dot{\gamma} = V\frac{\cos(\alpha)}{l_f + l_r} \cdot (\tan(\delta_f) - \tan(\delta_r))$
α	Vehicle slip angle	$\alpha = \tan^{-1}\left(\frac{L_r \tan(\delta_r) + L_f \tan(\delta_f)}{L_f + l_r}\right)$
V	Vehicle velocity	-

Road bank angle consideration

If the influence of road bank angles is included, then the global position at x direction can be rewritten as

$$\dot{X} = V\cos(\gamma + \alpha)$$

where γ is the road bank angle with a sign convention. The yaw dynamics of the vehicle are not affected by the road bank angle. Hence, Following Equation remains the same even in the presence of a bank angle.

$$\dot{\gamma} = V\frac{\cos(\alpha)}{l_f + l_r} \cdot (\tan(\delta_f) - \tan(\delta_r))$$

Regarding lateral tyre forces at greater slip angles, remark Large slip angles will break the assumption that the lateral tyre force is proportionate to the slip angle. In these situations, the lateral tyre force will be influenced by the longitudinal tyre force that is concurrently generated, the slip angle, the normal tyre load N, and the tire-road friction coefficient. The tyre model will no longer be linear at large slip angles.

Lateral and Yaw motion

$$m(\dot{v}_y + v_x\gamma) = F_{yf} \cdot cos(\delta_f) + F_{yr} \cdot cos(\delta_r) \tag{3.1}$$

$$I_z\dot{\gamma} = l_f \cdot F_{yf} \cdot cos(\delta_f) + l_r \cdot F_{yr} \cdot cos(\delta_r) \tag{3.2}$$

where v_x and v_y are the longitudinal and lateral velocities, and m is the vehicle's mass; The lengths of l_f and l_r, respectively, between the front and rear axles and the CG; The lateral forces of the vehicle are F_{yf} and F_{yr}. The steering angles of the front and rear wheels are denoted by δ_f and δ_r. I_z is the inertia's yaw moment. The front and rear tyres' slip angles are calculated as

$$\alpha_f = \delta_f - \frac{v_y + l_f \cdot \gamma}{v_x} \tag{3.3}$$

$$\alpha_r = \delta_r - \frac{v_y + l_r \cdot \gamma}{v_x} \tag{3.4}$$

therefore the linear tire model can be expresses as follows,

$$F_{yf} = k_f \cdot \alpha_f \tag{3.5}$$

$$F_{yr} = k_r \cdot \alpha_r \tag{3.6}$$

where, K_f and k_r are the vehicle cornering stiffness. Then the vehicle dynamics equations can be written as follows,

$$\begin{aligned} m\dot{v}_y + \frac{(k_f + k_r)\cdot v_y}{v_x} + (m\cdot v_x + \frac{l_f k_f - l_r k_r}{v_x})\cdot\gamma = k_f\delta_f + k_r\delta_r \\ I_z\dot{\gamma} + \frac{({l_f}^2 k_f + {l_r}^2 k_r)\cdot\gamma}{v_x} + \frac{(l_f k_f - l_r k_r)\cdot\gamma}{v_x} = l_f k_f\delta_f - l_r k_r\delta_r \end{aligned} \tag{3.7}$$

So the state space equation for two DOF vehicle dynamic can be written as follows,

$$\dot{x} = Ax + Bu \tag{3.8}$$

where $x = [v_y, \gamma]$, lateral motion of the vehicle and $u = [\delta_r, \delta_f]$ moreover,

$$A = -\frac{1}{v_x}\cdot\begin{bmatrix} \frac{1}{m}(k_f + k_r) & \frac{1}{m}(l_f k_f - l_r k_r) + {v_x}^2 \\ \frac{(l_f k_f - l_r k_r)}{I_z} & \frac{({l_f}^2 k_f + {l_r}^2 k_r)}{I_z} \end{bmatrix} \tag{3.9}$$

$$B = \begin{bmatrix} \frac{1}{m}k_f & \frac{1}{m}k_r \\ \frac{l_f k_f}{I_z} & -\frac{l_r k_r}{I_z} \end{bmatrix} \tag{3.10}$$

To find the characteristic roots, we need to compute the determinant of matrix A and solve for the eigenvalues. The characteristic equation is given by:

$$\det(A - \lambda I) = 0 \tag{3.11}$$

Where λ is the eigenvalue and I is the identity matrix.

Let's calculate the characteristic roots:

$$\det\left(-\frac{1}{v_x}\begin{bmatrix} \frac{1}{m}(k_f + k_r) - \lambda & \frac{1}{m}(l_f k_f - l_r k_r) + v_x^2 \\ \frac{(l_f k_f - l_r k_r)}{I_z} & \frac{(l_f^2 k_f + l_r^2 k_r)}{I_z} - \lambda \end{bmatrix}\right) = 0 \tag{3.12}$$

Solve for λ:

$$\det\left(\begin{bmatrix} \lambda - \frac{1}{v_x}\left(\frac{1}{m}(k_f + k_r) - \lambda\right) & -\frac{1}{v_x}\left(\frac{1}{m}(l_f k_f - l_r k_r) + v_x^2\right) \\ -\frac{1}{v_x}\left(\frac{(l_f k_f - l_r k_r)}{I_z}\right) & \lambda - \frac{1}{v_x}\left(\frac{(l_f^2 k_f + l_r^2 k_r)}{I_z} - \lambda\right) \end{bmatrix}\right) = 0 \tag{3.13}$$

Expanding the determinant, we get:

$$\left(\lambda - \frac{1}{v_x}\left(\frac{1}{m}(k_f + k_r) - \lambda\right)\right)\left(\lambda - \frac{1}{v_x}\left(\frac{(l_f^2 k_f + l_r^2 k_r)}{I_z} - \lambda\right)\right) \tag{3.14}$$

$$-\left(-\frac{1}{v_x}\left(\frac{1}{m}(l_f k_f - l_r k_r) + v_x^2\right)\right)\left(-\frac{1}{v_x}\left(\frac{(l_f k_f - l_r k_r)}{I_z}\right)\right) = 0 \tag{3.15}$$

Simplifying the equation, we obtain a quadratic equation in terms of λ:

$$\lambda^2 - \lambda\left(\frac{1}{v_x}\left(\frac{1}{m}(k_f + k_r) + \frac{(l_f^2 k_f + l_r^2 k_r)}{I_z}\right)\right) \tag{3.16}$$

$$+\frac{1}{v_x^2}\left(\frac{1}{m}(k_f + k_r)\frac{(l_f^2 k_f + l_r^2 k_r)}{I_z} - \left(\frac{1}{m}(l_f k_f - l_r k_r) + v_x^2\right)\left(\frac{(l_f k_f - l_r k_r)}{I_z}\right)\right) = 0 \tag{3.17}$$

Substituting $\frac{m(l_r k_r - l_f k_f)}{I^2 k_f k_r}$ with K, we have:

$$\lambda^2 - \lambda\left(\frac{1}{v_x}\left(\frac{1}{m}(k_f + k_r) + \frac{(l_f^2 k_f + l_r^2 k_r)}{I_z}\right)\right) \tag{3.18}$$

$$+\frac{l^2 k_f k_r}{m \cdot I_z \cdot v_x^2}(1 + K v_x^2) = 0 \tag{3.19}$$

The solutions to this quadratic equation will give us the characteristic roots, which are the eigenvalues of matrix A. OR another way to simplify this is the following, we obtain a quadratic equation in terms of λ:

$$\lambda^2 - \lambda\left(\frac{M}{v_x} + N\right) + \frac{K}{v_x^2} = 0 \tag{3.20}$$

where

$$M = \frac{1}{m}(k_f + k_r) + \frac{l_f^2 k_f + l_r^2 k_r}{I_z}$$

$$N = \frac{1}{m}(l_f k_f - l_r k_r) + v_x^2$$

$$K = \frac{1}{m}(k_f + k_r)\frac{l_f^2 k_f + l_r^2 k_r}{I_z} - \left(\frac{1}{m}(l_f k_f - l_r k_r) + v_x^2\right)\left(\frac{l_f k_f - l_r k_r}{I_z}\right)$$

Therefore it can be conculuded from Eq3.19 and Eq 3.20 that the 4 Wheel steering system will by stable for $K < -\frac{1}{v_x^2}$

In stationary situations, $\dot{v}_y = 0$, $\dot{\gamma} = 0$. Equation 3.7 can be expressed as:

$$mv_x^2(l_f k_f - l_r k_r) - l^2 k_f k_r \gamma = -k_f k_r l v_x(\delta_f - \delta_r) \tag{3.21}$$

$$\left(1 + Kv_x^2\right)\gamma = \frac{v_x}{l}(\delta_f - \delta_r) \tag{3.22}$$

$$\delta_f - \delta_r = \frac{\gamma l}{v_x\left(1 + Kv_x^2\right)} \tag{3.23}$$

It has been proven that the lateral stability evaluation index of γ is adjustable by controlling δ_r before a vehicle reaches the limit operating condition. According to Eq. 3.23, the rear steering can be designed as:

$$\delta_r = \delta_f - \frac{\gamma l}{v_x\left(1 + Kv_x^2\right)} \tag{3.24}$$

3.1.2 Three DOF

This section describes a 3-DOF system as the handling model. In addition to yaw motion[31] and 1-DOF resulting from the rotational motion of each tyre, it takes into account the lateral and longitudinal motions of the vehicle body. The vehicle moves in three directions: longitudinally along the x-axis, laterally along the y-axis, and angularly in the yaw direction about the vertical z-axis. Accelerations along the longitudinal and lateral axes, represented by the letters axe and ay, respectively, can be used to describe motion in the horizontal plane. The symbolsV_x and V_y, respectively, represent the velocities in the longitudinal and lateral directions.

The annotations The tyre forces in the longitudinal and lateral directions are represented by the symbols F_{xpq} and F_{ypq}, respectively. The index p indicates the front f or rear r tyres, while the index q indicates the left l or right r tyres. δ represents the steering angle, $\dot{r}$ the yaw rate, and m the mass of the vehicle. The definition of acceleration in the longitudinal x-axis is $\dot{v}_x = a_x + \dot{r} \cdot v_y$ The definition of longitudinal acceleration,

obtained by adding up all the forces in the x-axis, is:

$$a_x = \frac{F_{xrr} + F_{xrl} + (F_{xfl} + F_{xfr}) \cdot \cos(\delta) - (F_{yfl} + F_{yfr}) \cdot \sin(\delta)}{m} \tag{3.25}$$

$$\dot{V}_y = a_y - \dot{r} V_x \cdot \tag{3.26}$$

$$a_y = \frac{F_{yrr} + F_{yrl} + (F_{yfl} + F_{yfr}) \cdot \cos(\delta) + (F_{xfl} + F_{xfr}) \cdot \sin(\delta)}{m} \tag{3.27}$$

The vehicle body side slip angle, represented by β, can be obtained by integrating the longitudinal and lateral vehicle velocities, V_x and V_y:

$$\alpha = \arctan\left(\frac{V_y}{V_x}\right) \tag{3.28}$$

The tyre forces F_{xpq} and F_{ypq} operating on each tyre affect the yaw acceleration ($\ddot{r}$) as well:

$$\begin{aligned}\ddot{r} = \frac{1}{I_z}\Bigg[&\left(\frac{t_r}{2}F_{xrr} - F_{yrr} \cdot c - \frac{t_r}{2}F_{xrl} - F_{yrl} \cdot c\right) + \left(-\frac{t_f}{2}F_{xfl}\cos(\delta) + F_{yfl}\sin(\delta) \cdot b\right) \\ &+ \left(\frac{t_f}{2}F_{yfl}\sin(\delta) + F_{yfl}\cos(\delta) \cdot b\right) + \left(\frac{t_f}{2}F_{xfr}\cos(\delta) + F_{xfr}\sin(\delta) \cdot b\right) \\ &+ \left(-\frac{t_f}{2}F_{yfr}\sin(\delta) + F_{yfr}\cos(\delta) \cdot b\right)\Bigg](6)\end{aligned} \tag{3.29}$$

where b and c stand for the distances, respectively, between the front and rear axles and the centre of gravity. Iz is the moment of inertia around the z-axis, and t_f and t_r, respectively, stand for the distances between the centres of the right and left tyres at the front and rear, respectively. Tyre models can be used to determine the forces acting on the tyres, which are necessary in equations 3.25, 3.27, and 3.29. The Calspan tyre model ([20], [16], [19]) is utilised out of all the tyre models that are available in the literature. Like the majority of tyre models, this one also determines the longitudinal and lateral tyre forces by using the vertical forces F_{zpq}, the tyre slip angles α_{pq}, and the tyre slip rates s_{pq} which are provided by:

$$F_{zfl/r} = \frac{mg \pm ma_y}{2t_f}\frac{b}{l} - \frac{ma_x \cdot h}{2l} \tag{3.30}$$

$$F_{zrl/r} = \frac{mg \pm ma_y}{2t_r}\frac{c}{l} + \frac{ma_x \cdot h}{2l} \tag{3.31}$$

$$\alpha_{fl/r} = \text{arctan}(\frac{v_y + b \cdot \dot{r}}{v_x + \frac{t_f \dot{r}}{2}}) - \delta \tag{3.32}$$

$$\alpha_{fl/r} = \text{arctan}(\frac{v_y - c \cdot \dot{r}}{v_x + \frac{t_r \dot{r}}{2}}) \tag{3.33}$$

$$s_{pq} = \frac{\omega_{pq} \cdot R_\omega}{v_{\omega pq}} - 1 \tag{3.34}$$

The above used to derive the wheel radius: where R_ω is the wheel radius; h is the height of the centre of gravity; l is the wheelbase (distance between the centre of the front axle and the centre of the rear axle); ω_{pq} is the wheel angular speed; and $v_{\omega pq}$ indicates the wheel forward velocities.

The following provides the components of the longitudinal velocities in the front and rear:

$$v_{\omega xfl/r} = V_{tf} \cos(\alpha_{fl/r}) \tag{3.35}$$

where the front tire's speed is,

$$V_{tf} = \sqrt{(V_y + b\dot{r})^2 + V_x^2} \tag{3.36}$$

The component of rear longitudinal velocity is

$$v_{\omega xrl/r} = V_{tr} \cos(\alpha_{rl/r}) \tag{3.37}$$

where the rear tire's speed is,

$$V_{tr} = \sqrt{(V_y - c\dot{\gamma})^2 + V_x^2} \tag{3.38}$$

The lateral and longitudinal accelerations determine the roll and pitch motions. The vehicle body experiences roll and pitch, so when assessing how handling affects pitch

and roll motions, the sprung mass, represented by m_s, must be taken into account [33]:

$$J_{sx}\ddot{\phi} = -m_s h_s a_y + (m_s g h_s - \kappa_\phi)\phi + \dot{\phi} \cdot (-\beta_\phi) \tag{3.39}$$

$$J_{sy}\ddot{\theta} = -m_s h_s a_x + (m_s g h_s - \kappa_\theta)\theta + \dot{\theta} \cdot (-\beta_\theta) \tag{3.40}$$

where g is the gravitational acceleration, h_s is the height of the sprung mass centre of gravity, and κ_ϕ, β_ϕ, κ_θ, β_θ are the stiffness and damping constants for roll and pitch, respectively. The sprung mass's moment of inertia about the x- and y-axes is represented by the symbols J_{sx} and J_{sy}, respectively. The essential symbols, definitions, and corresponding equations for the lateral dynamics of a three-degree-of-freedom vehicle model are summarized in table 3.2. The equations are predicated on the idea of linear tyre behaviour and small angles. The difference between the steering angle $(\delta)$and the ratio of lateral to longitudinal velocity approximates the slip angles at the front and rear tyres, which determines the lateral forces (F_{yf}) and (F_{yr}). The net moment about the vehicle's vertical axis (I_z), which is impacted by the lateral forces at the front and rear axles as well as the distances (b) and (c) from the vehicle's centre of gravity to the corresponding axles, determines the yaw rate (r).

Table 3.2: Summary of Lateral Dynamics for 3 DOF Vehicle Model

Symbol	**Description**	**Equation**
y	Lateral displacement	$\dot{y} = V\sin(\psi)$
ψ	Yaw angle	$\dot{\psi} = r$
r	Yaw rate	$\dot{r} = \frac{1}{I_z}\left(bF_{yf} - cF_{yr}\right)$
F_{yf}	Lateral force at front axle	$F_{yf} = C_{\alpha f}\left(\delta - \frac{V\sin(\psi)}{V\cos(\psi)}\right)$
F_{yr}	Lateral force at rear axle	$F_{yr} = C_{\alpha r}\left(-\frac{V\sin(\psi)}{V\cos(\psi)}\right)$
V	Vehicle velocity	-
I_z	Yaw moment of inertia	-
b, c	Distance from CG to front and rear axle	-
$C_{\alpha f}, C_{\alpha r}$	Cornering stiffness of front and rear tires	-
δ	Steering angle	-
$\dot{v}_x$	Longitudinal acceleration	$\dot{v}_x = a_x + \dot{r}\cdot v_y$
a_x	Definition of longitudinal acceleration	Look equation 3.25
$\dot{V}_y$	Y Velocity	$\dot{V}_y = a_y - \dot{r}V_x$
a_y	Lateral acceleration	Look equation 3.27
α	Side slip angle	$\alpha = \arctan\left(\frac{V_y}{V_x}\right)$
$\ddot{r}$	Yaw acceleration	Look equation 3.29
$v_{\omega xfl/r}$	Front/rear longitudinal velocity component	$v_{\omega xfl/r} = V_{tf/tr}\cos(\alpha_{fl/r})$
V_{tf}	Front tire's speed	$V_{tf} = \sqrt{(V_y + b\dot{r})^2 + V_x^2}$
$v_{\omega xrl/r}$	Rear longitudinal velocity component	$v_{\omega xrl/r} = V_{tr}\cos(\alpha_{rl/r})$
V_{tr}	Rear tire's speed	$V_{tr} = \sqrt{(V_y - c\dot{\gamma})^2 + V_x^2}$

3.1.3 14-DOF vehicle dyanmic

In this section I am summarizing all 14-DOF vehicle dynamic equations, the tables 3.3 and 3.4 show the description of each symbol used in the model as follows,

Table 3.3: List of Symbols and Descriptions - Part 1

Symbol	Description
ig	represents front right, front left, rear right and rear left parts of the vehicle
a_{cg}	Acceleration of the vehicle's center of gravity (m/s^2)
a_{uij}^{F2}	Acceleration of the center of the unsprung mass of ij in F2 (m/s^2)
c	Suspension damping coefficient (N.s/m)
$F_{txij}/F_{tyij}/F_{tzij}$	Force exerted by the ij tire in the Longitudinal/lateral/vertical direction (N)
$F_{cgxij}^{F1}, F_{cgyij}^{F1}$	Force exerted by the ij tire in the Longitudinal/lateral direction at the contact point in F1 (N)
$F_{cgxij}^{F2}, F_{cgyij}^{F2}$	Force exerted by the ij tire in the forward/lateral direction at the contact point in F2 (N)
F_{cgz}^{F1}	Force exerted upward by the ij tire in F1 (N)
F_{sz}^{F2}	Force exerted upward by the suspension in F2 (N)
F_x, F_y, F_z	Total force exerted by the vehicle in the forward/lateral/vertical direction (N)
g	Gravitational acceleration (m/s^2)
h	Height of the vehicle's center of gravity (m)
I_{roll}	Moment of inertia resisting vehicle roll (kg.m^2)
I_{pitch}	Moment of inertia resisting vehicle pitch (kg.m^2)
I_{yaw}	Moment of inertia resisting vehicle yaw (kg.m^2)
I_{wheel}	Moment of inertia resisting wheel spin (kg.m^2)
K_s	Stiffness of the suspension spring (N/m)
K_t	Stiffness of the tire (N/m)
L	Wheelbase (m)
l_f	Distance from the front axle to the center of gravity (m)
l_r	Distance from the rear axle to the center of gravity (m)
l_s	Instantaneous length of the strut (m)
$M_{xfr}/M_{yfr}/M_{zfr}$	Moment causing rotation around the x/y/z axis due to forces on the front right corner (N.m)
$M_{xfl}, M_{yfl}, M_{zfl}$	Moment causing rotation around the x/y/z axis due to forces on the front left corner (N.m)
$M_{xrr}, M_{yrr}, M_{zrr}$	Moment causing rotation around the x/y/z axis due to forces on the rear right corner (N.m)
$M_{xrl}, M_{yrl}, M_{zrl}$	Moment causing rotation around the x/y/z axis due to forces on the rear left corner (N.m)
m	Mass of the vehicle body (kg)
m_t	Mass of the tire (kg)

Table 3.4: List of Symbols and Descriptions - Part 2

Symbol	Description
r_{ij}	Radius of the ij tire when it touches the ground (m)
r_0	Radius of the tire when it is not loaded (m)
$\vec{r}_{uij/c}$	Distance from the center of the ij unsprung mass to the center of gravity (m)
$\vec{r}_{cgij/c}$	Distance from the contact point of the ij tire to the center of gravity (m)
$\ddot{x}$	Rotation rate of the vehicle body around the x-axis (rad/s)
$\ddot{y}$	Rotation rate of the vehicle body around the y-axis (rad/s)
$\ddot{z}$	Rotation rate of the vehicle body around the z-axis (rad/s)
s	Steering-ratio Sensitivity of tire angle to steering wheel angle
T	Width of the Track (m)
$T_{\text{drive}}/T_{\text{brake}}$	Braking and driving torque (N.m)
V_{cg}	Speed of the vehicle's center of gravity (m/s)
$V_x/V_y/V_z$	Speed of the vehicle's center of gravity in the forward/sideways/upward direction (m/s)
V_{uij}^{F2}	Speed of the center of the ij unsprung mass in F2 (m/s)
V_{ij}^{F2}	Speed of the center of the ij unsprung mass in the forward/sideways/upward direction (m/s)
V_{cgij}^{F1}	Speed of the contact point of the ij tire in F1 (m/s)
V_{cgij}^{F2}	Speed of the contact point of the ij tire in F2 (m/s)
$V_{cgxij}/V_{cgyij}/V_{cgzij}$	Speed of the contact point of the ij tire in the longitudinal/lateral/vertical direction (m/s)
$\vec{\omega}$	Rotational rate (Angular velocity) of the vehicle body (rad/s)
$\omega_x/\omega_y/\omega_z$	Angular velocity of the vehicle body in longitudinal/lateral/vertical direction (rad/s)
ω_{ij}	Rotation rate of the ij tire (rad/s)
Δz_{sij}	Compression or stretching of the suspension spring for ij (m)
Δz_{so}	Compression or stretching of the suspension spring when the vehicle is at rest (m)
Δz_{tij}	Compression or stretching of the tire for ij (m)
Δz_{to}	Compression or stretching of the tire when the vehicle is at rest (m)
$\phi/\theta/\psi$	Tilt angles of the vehicle in the Roll/Pitch/Yaw axis (rad)
δ	Steering wheel angle (rad)
$\delta_{\text{FR}}/\delta_{\text{FL}}$	Steering angles of the front right/left tire (rad)
λ_{ij}	Longitudinal slip of the ij tire
α_{ij}	Lateral slip of the ij tire

Kinematics

Systems coordination

As illustrated in the figure, two moving coordinate systems are used. The first coordinate system (F1) is fixed to the tire-ground contact point, at coordinate axes (x', y', z') and the vector units $(\hat{i}', \hat{j}', \hat{k}')$ obtained through rotating the inertial frame X, Y, and Z along with the yaw angle ψ about the Z-axis, yielding a rotation matrix:

$$R_\psi = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.41}$$

Two consecutive rotations of F1 yielded the second coordinate system (F2), which is attached to the vehicle's c.g. and has coordinate axes (x, y, z) and unit vectors $(\hat{i}, \hat{j}, \hat{k})$. The first rotation of F1 was performed with a pitch angle θ about the y-axis, yielding a rotation matrix:

$$R_\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.42}$$

Next, a rotation matrix is obtained with the roll angle ϕ about the x-axis:

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.43}$$

Velocities

The vehicle's top view schematic is displayed in the following figure []. Let $\vec{\omega}$ be the angular velocity of the vehicle body and $\vec{v}_{cg}$ be the linear and angular velocities of c.g.

$$\vec{v}_{cg} = v_x\hat{i} + v_y\hat{j} + v_z\hat{k} \tag{3.44}$$

$$\vec{\omega} = \omega_x\hat{i} + \omega_y\hat{j} + \omega_z\hat{k} \tag{3.45}$$

From the velocity of c.g., the following values can be obtained: the velocity of the ij centre of tyre contact patch ($\vec{v}_{2ij}$) and the velocity of the centre of the ij unsprung mass ($\vec{v}_{0ij}$).

$$\vec{v}_{uij} = \vec{v}_{cg} + \vec{\omega} \times \vec{r}_{uij} \tag{3.46}$$

$$\vec{v}_{cgij} = \vec{v}_{cg} + \vec{\omega} \times \vec{r}_{cgij} \tag{3.47}$$

where ij stands for rear left (rl), rear right (rr), front right (fr), and front left(fl).

Accelerations

By differentiating c.g.'s velocity with respect to time in the following way, the acceleration of c.g. ($\vec{a}_{cg}$) can be found:

$$\vec{a}_{cg} = \frac{d}{dt}\vec{v}_{cg} = \frac{d}{dt}(v_x) + \vec{\omega} \times \vec{v}_{cg} = a_x\hat{i} + (\dot{v}_y - \omega_x v_z + \omega_z v_x)\hat{j} + (\dot{v}_z + \omega_x v_y - \omega_y v_x)\hat{k} \tag{3.48}$$

&

$$a_x = \dot{v}_x + \omega_y v_z - \omega_z v_y \tag{3.49}$$

Similarly, in F2 ($\vec{a}_{uij}$), the acceleration of the centre of ij unsprung mass is provided by:

$$\vec{a}_{uij} = \frac{d}{dt}(v_{uij}) + \vec{\omega} \times \vec{v}_{uij} = a_x\hat{i} + (\dot{v}_{uyij} - \omega_x v_{uzij} + \omega_z v_{uxij})\hat{j} + (\dot{v}_{uzij} + \omega_x v_{uyij} - \omega_y v_{uxij})\hat{k} \tag{3.50}$$

&

$$a_x = \dot{v}_{uxij} + \omega_y v_{uzij} - \omega_z v_{uyij} \tag{3.51}$$

As we will see, the terms v_{uxij} and v_{uyij} in the last equation are derived from 3.46, whereas v_{uzij} is derived from 3.94.

2-Kinetics

Steering mechanism

Using "Ackerman steering" the steering mechanism involves two steering angles: the front left (δ_{fl}) and the front right (δ_{fr}).

$$\delta_{fr} = \arctan\left(\frac{L}{\frac{L}{\tan(\frac{\delta}{S})} + \frac{T}{2}}\right) \tag{3.52}$$

$$\delta_{fl} = \arctan\left(\frac{L}{\frac{L}{\tan(\frac{\delta}{S})} + \frac{T}{2}}\right) \tag{3.53}$$

Tire model

The tyre model utilised in this instance is Pacejka's model [23], which has longitudinal along with lateral slips (λ_{ij}, α_{ij}) denoted by the following definitions:

$$\lambda_{ij} = \left(\frac{\omega_{ij} \times r_{il} - (v_{gxij} \times \cos(\delta_{ij})) + v_{gyij} \times \sin(\delta_{ij})}{v_{gxij} \times \cos(\delta_{ij}) + v_{gyij} \times \sin(\delta_{ij})} \right) \tag{3.54}$$

$$\alpha_{ij} = \arctan\left(\frac{v_{gyij}}{v_{gxij}}\right) - \delta_{ij} \tag{3.55}$$

Where:

$$\vec{v}_{2ij} = R_\phi R_\theta \vec{v}'_{2ij} = v_{cgxij}\hat{i}' + v_{cgyij}\hat{j}' + v_{cgzij}\hat{k}' \tag{3.56}$$

We can calculate the longitudinal and lateral tyre forces (F_{txij}, F_{tyij}) using the tyre model with the slip ratios defined above. As a result, we can obtain the longitudinal and lateral forces at the ground contact patch in F1 (F_{cgxij}, F_{cgyij}) as follows:

$$F_{cgxij} = F_{txij}\cos(\delta_{ij}) - F_{tyij}\sin(\delta_{ij}) \tag{3.57}$$

$$F_{cgyij} = F_{txij}\sin(\delta_{ij}) + F_{tyij}\cos(\delta_{ij}) \tag{3.58}$$

Since each corner's tyre and suspension are represented as in Figure 3.1, the vertical tyre force (F'_{zij}) can be computed as follows:

$$F_{cgzij} = K_{ti} \times \Delta z_{tij} \tag{3.59}$$

where:

$$\dot{\Delta z_{ij}} = -v_{cgzij} \tag{3.60}$$

Using the following initial values that were derived from static conditions:

$$\Delta z_{fr} = \Delta z_{fl} = \frac{m \cdot g \cdot \frac{l_r}{2L} + m_t \cdot g}{K_t f} \tag{3.61}$$

$$\Delta z_{rr} = \Delta z_{rf} = \frac{m \cdot g \cdot \frac{l_f}{2L} + m_t \cdot g}{K_t r} \tag{3.62}$$

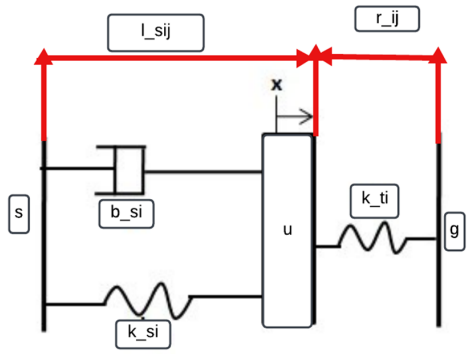


Figure 3.1: Tire suspension model in ij corners

We employ matrix transformations in the manner described below to obtain these forces in F2 ($\vec{F}_{2ij}$):

$$\vec{F}_{2ij} = R_\theta^T R_\phi^T \vec{F}_{cgij} \tag{3.63}$$

Then, the total forces (inertia and external forces) acting in the directions x, y, z (F_x, F_y, F_z) are:

$$F_x = \sum (F'_{xij} - m_t \dot{v}_{uxij}) + 4m_t \ddot{y} \sin(\theta) + m\ddot{z} \sin(\theta) \tag{3.64}$$

$$F_y = \sum (F'_{yij} - m_t \dot{v}_{uyij}) - 4m_t g \sin(\phi) \cos(\theta) - m_t \cdot g \sin(\phi) \cos(\theta) \tag{3.65}$$

$$F_z = \sum \left(F'_{szij}\right) - m_t g \cos(\phi) \cos(\theta) \tag{3.66}$$

where the vertical force at the suspension is denoted by F'_{zij} and the longitudinal/lateral inertia force in each tyre is represented by $m\dot{a}_x$ / $m\dot{a}_y$ (Fig.3.1). and equal:

$$F'_{zij} = K_{si} \cdot \Delta z_{sij} + c_{si} \cdot \dot{\Delta} z_{sij} \tag{3.67}$$

where:

$$\Delta \dot{z}_{sij} = v_{uzij} - v_{szij} \tag{3.68}$$

An initial values of:

$$\Delta z_{fr} = \Delta z_{fl} = \frac{m \cdot g \cdot l_r}{2L \cdot K_s f} \tag{3.69}$$

$$\Delta z_{rr} = \Delta z_{rl} = \frac{m \cdot g \cdot l_f}{2L \cdot K_s r} \tag{3.70}$$

where the following relations can be used to obtain v_{szij} and where v_{uzij} is calculated from 3.94:

$$v_{szfr} = v_z - l_f \cdot \omega_y - h \cdot \omega_x \tag{3.71}$$

$$v_{szfl} = v_z - l_f \cdot \omega_y + h \cdot \omega_x \tag{3.72}$$

$$v_{szrr} = v_z + l_r \cdot \omega_y - h \cdot \omega_x \tag{3.73}$$

$$v_{szrl} = v_z + l_r \cdot \omega_y + h \cdot \omega_x \tag{3.74}$$

where $h = T/2$ as shown in the fig 3.2

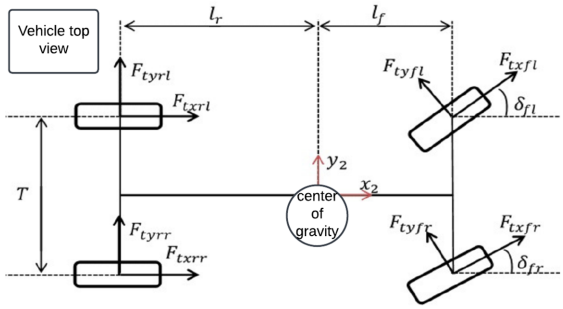


Figure 3.2: Vehicle top view

Vehicle Moments

Furthermore, the moments resulting from forces at the front right corner acting, for example, on x, y, z directions (M_{xfr}, M_{yfr}, M_{zfr}) can be computed as follows:

$$\Delta M_{xfr} = F_{cgyfr} \cdot (l_{sfr} + r_{fr}) - (m_t a_{uyfr} + m_t g \sin(\phi)\cos(\theta)) \cdot l_{sfr} - F_{szfr} \cdot T/2 \tag{3.75}$$

$$\Delta M_{yfr} = -F_{cgxfr} \cdot (l_{sfr} + r_{fr}) - (m_t a_{uxfr} - m_t g \sin(\theta)) \cdot l_{sfr} + F_{szfr} \cdot l_f \tag{3.76}$$

$$\begin{aligned} M_{zfr} = {} & (F_{cgxfr} - m_t a_{uxfr} + m_t g \sin(\theta)) \cdot T/2 \\ & + (F_{gyrr} - m_t a_{uyfr} - m_t g \sin(\phi)\cos(\theta)) \cdot l_f \end{aligned} \tag{3.77}$$

Where

$$r_{fr} = \frac{F_{r_0 - x_{tfr}}}{\cos(\phi)} \frac{1}{\cos(\theta)} \tag{3.78}$$

$$l_{sfr} = h - r_0 + \Delta z_{t0fr} - \Delta z_{xsfr} + \Delta z_{sofr} \tag{3.79}$$

Likewise, moments at the remaining corners resulting from forces are:

Vehicle front-left:

$$\Delta M_{xfr} = F_{cgyfl} \cdot (l_{sfl} + r_{fl}) - (m_t a_{uyfl} + m_t g \sin(\phi) \cos(\theta)) \cdot l_{sfl} - F_{szfl} \cdot T/2 \tag{3.80}$$

$$\Delta M_{yfl} = -F_{cgxfl} \cdot (l_{sfl} + r_{fl}) - (m_t a_{uxfl} - m_t g \sin(\theta)) \cdot l_{sfl} - F_{szfl} \cdot l_f \tag{3.81}$$

$$\begin{aligned} M_{zfl} = &(-F_{cgxfl} + m_t a_{uxfl} - m_t g \sin(\theta)) \cdot T/2 \\ &+ (F_{gyfl} - m_t a_{uyfl} - m_t g \sin(\phi) \cos(\theta)) \cdot l_f \end{aligned} \tag{3.82}$$

Vehicle Rear-right:

$$\Delta M_{xrr} = F_{cgyrr} \cdot (l_{srr} + r_{rr}) - (m_t a_{uyrr} + m_t g \sin(\phi) \cos(\theta)) \cdot l_{srr} - F_{szrr} \cdot T/2 \tag{3.83}$$

$$\Delta M_{yrr} = -F_{cgxrr} \cdot (l_{srr} + r_{rr}) + (m_t a_{uxrr} - m_t g \sin(\theta)) \cdot l_{srr} + F_{szrr} \cdot l_r \tag{3.84}$$

$$\begin{aligned} M_{zrr} = &(F_{cgxrr} - m_t a_{uxrr} + m_t g \sin(\theta)) \cdot T/2 \\ &+ (-F_{gyrr} + m_t a_{uyrr} + m_t g \sin(\phi) \cos(\theta)) \cdot l_r \end{aligned} \tag{3.85}$$

Vehicle Rear-left:

$$\Delta M_{xrl} = F_{cgyrl} \cdot (l_{srl} + r_{rl}) - (m_t a_{uyrl} + m_t g \sin(\phi) \cos(\theta)) \cdot l_{srl} - F_{szrl} \cdot T/2 \tag{3.86}$$

$$\Delta M_{yrl} = -F_{cgxrl} \cdot (l_{srl} + r_{rl}) - (m_t a_{uxrl} - m_t g \sin(\theta)) \cdot l_{srl} + F_{szrl} \cdot l_r \tag{3.87}$$

$$\begin{aligned} M_{zrl} = &(-F_{cgxrl} + m_t a_{uxrl} - m_t g \sin(\theta)) \cdot T/2 \\ &+ (-F_{gyrl} + m_t a_{uyfr} + m_t g \sin(\phi) \cos(\theta)) \cdot l_r \end{aligned} \tag{3.88}$$

All forces and moments acting on the acceleration of the centre of gravity (c.g.). Newton's laws can be directly applied to obtain the following equations, which represent the body's six degrees of freedom:

$$\ddot{x} = \frac{1}{m} F_x - \omega_y v_z + \omega_z v_y \tag{3.89}$$

$$\ddot{y} = \frac{1}{m} F_y - \omega_z v_x + \omega_x v_z \tag{3.90}$$

$$\ddot{z} = \frac{1}{m} F_z - \omega_x v_y + \omega_y v_z \tag{3.91}$$

$$\dot{\omega}_x = \frac{1}{I_x} \sum \Delta M_{xij} \tag{3.92}$$

$$\dot{\omega}_y = \frac{1}{I_y} \sum \Delta M_{yij} \tag{3.93}$$

$$\dot{\omega}_z = \frac{1}{I_z} \sum \Delta M_{zij} \tag{3.94}$$

And by reapplying Newton's law to the z direction of the unsprung mass centre, we can obtain the four equations that represent the four degrees of freedom for the vertical velocity of the unsprung mass, which are as follows:

$$\dot{v}_{uzij} = \frac{1}{m_t} \left(F_{cgzij} - m_t g \cos(\phi) \cos(\theta) - K_{si} x_{sij} - b_{si} \dot{x}_{sij} - m_t \left(v_{uyij} \omega_x - v_{uxij} \omega_y \right) \right) \tag{3.95}$$

The four DOF of wheel spin represented by the remaining equations (Fig 3.3) can be computed as follows:

$$\dot{\omega}_{ij} = \frac{T_{dij} - T_{bij} - F_{txij} \cdot r_{ij}}{I_w} \tag{3.96}$$

The vehicle model with 14 degrees of freedom equations of motion are represented by equations 3.89 through 3.96.

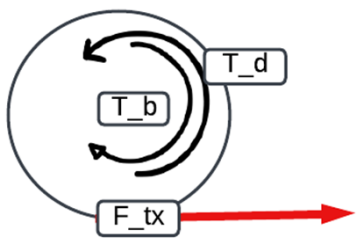


Figure 3.3: Wheel rotational model for braking and driving

3.2 Cornering stiffness estimation

Road surface condition classifications in real time are essential for preserving appropriate vehicle control within its handling parameters, and cornering stiffness value is a critical component of lateral dynamics. Road surface conditions have been estimated numerous times, but since most estimation techniques have some robustness issues in real-world applications, there is always room for improvement. I will take into account the following when estimating in this work. There are multiple ways to calculate a tire's cornering stiffness. Using a model-based estimator based on hub accelerometer measurements is one popular method. The wheel hub's acceleration is measured by the hub accelerometer and is correlated with the lateral force and tyre slip angle. This equation provides a basic mathematical model for calculating a tire's cornering stiffness:

$$C = \alpha F_y \tag{3.97}$$

where:

C : Cornering stiffness

α : Tire slip angle

F_y : Lateral force acting on the tire

According to this model, the lateral force and tyre slip angle are proportionate because tyres are assumed to behave linearly. It's crucial to remember that big slip angles and nonlinear tyre properties might make this assumption invalid.

A critical component of vehicle dynamics, cornering stiffness gauges the amount of force needed to shift a tire's direction. Numerous factors, such as tyre pressure, temperature, wear, load, and road surface conditions, affect this parameter. Precise assessment of cornering rigidity is necessary for the efficient operation of car control systems, including stability, braking, and steering.

More complex models are used to capture the nonlinearities and uncertainties related to tyre and road surface characteristics in order to overcome the shortcomings of the simpler model. The Pacejka Magic Formula is one such model that is frequently applied in the field of vehicle dynamics. The lateral force is expressed as a function of the slip angle, longitudinal slip, and vertical load using the Pacejka Magic Formula: In

actuality, estimating cornering stiffness requires compiling information from a number of sensors, including accelerometers, steering angle sensors, and wheel speed sensors. The cornering stiffness is then estimated in real-time by processing these measurements using sophisticated algorithms, which frequently include Kalman filters and recursive least squares techniques.

It is essential to estimate cornering stiffness accurately in order to improve control systems and vehicle dynamics. While basic models offer a fundamental understanding, more precise models that capture the intricacies of tyre behaviour include the Pacejka Magic Formula. To achieve real-time cornering stiffness estimation in practice, complex algorithms and sensor fusion techniques are needed.

$$F_y = D \sin\left(C \arctan\left(B\alpha - E(B\alpha - \arctan(B\alpha))\right)\right) \tag{3.98}$$

where:

F_y : Lateral force acting on the tire

D : Peak value of lateral force

C : Shape factor

B : Stiffness factor

E : Horizontal shift factor

α : Tire slip angle

This formula works well under a variety of operating conditions and gives a more accurate depiction of the tire's lateral force characteristics.

3.3 General Longitudinal vehicle dynamic

Engineering's field of vehicle dynamics studies how a vehicle moves in relation to its intended use. This topic pertains to a specific category of products, mainly automobiles. Vehicle dynamics makes use of theories and methods from machine design, mechanical engineering, and other fields. Vehicle dynamics can be divided into three main categories: longitudinal, vertical, and lateral. In these papers, I will focus on the longitudinal dynamics. According to some theories, the three main goals of designing a

car are improved driver comfort, safety, and driving dynamics.

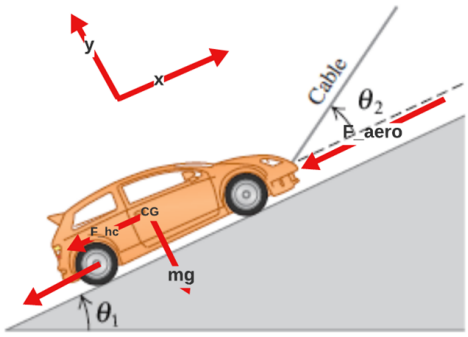


Figure 3.4: Free body diagram for vehicle showing the elementary dynamical forces on vehicle body

Air drag force

The drag or air resistance the vehicle encounters while moving is known as the ”air drag force,” and it can be represented as can be seen in 3.99,

$$F_d = \frac{1}{2} \times \rho A C_d \cdot v^2 \tag{3.99}$$

Where A is the frontal area, C_d is the drag force coefficient, v is the vehicle velocity, and ρ is the air density, which is 1.225 $\frac{kg}{m^2}$.

Acceleration

From Newton second law,

$$F_{total} = m \cdot a \tag{3.100}$$

The drag force caused by the air, as seen in section3.3, is equal to the subtraction of the forces exerted by the wheels, where F_{total} represents the total reaction forces acting on the vehicle body (it acts opposite to the direction of movement of the vehicle)[24]

$$F_{total} = F_{wheels} - F_{airdrag} \tag{3.101}$$

Thus,

$$a = F_{total} \cdot \frac{1}{m} \tag{3.102}$$

velocity

We can obtain the velocity vector or the scalar term (speed) in the following ways using the velocity definition, which states that acceleration is the velocity vector's first-time derivative: $a \cdot dt = dv$ After completing the integration, I obtain the following:

$$v = v_0 + \int_{t_0}^{t_1} a \cdot dt \tag{3.103}$$

Position

Since the velocity is the initial time derivative of the position by definition, we can obtain the position formula using the same method as for the velocity, as shown below.

$$x(t) = x_0 + v \cdot \Delta t \tag{3.104}$$

front wheel forces/reactions

Since a wheel is a rotating object, its torque can be computed using the following formula.

$$T_{frwheel} = F_{frwheel} \times R \tag{3.105}$$

As we know a wheel is a rotating object which has a torque that can be calculated through the following $F_{frwheel}$=$T_{frwheel} \cdot \frac{1}{R}$.
The reactions forces on the front wheels can be formulated as below, the details can be found in this work I did before, cited here [1]

$$F_{frwheels} = \frac{1}{l}(mg \cdot l_1 + h(F_b + f_r \cdot mg)) \tag{3.106}$$

You can view the various parameters by consulting figure 3.4, where l is the distance between the canters of the front and rear wheels, l_1 is the distance from the car centre to the front wheel, h is the height of the car, Fb is the braking force, and fr is the rolling

resistance.

$$F_b = \frac{w}{g} \cdot a \tag{3.107}$$

Where w is the car weight force (mg), g is the gravitational acceleration and a is the vehicle acceleration

$$F_{bmax} = \mu \cdot W \tag{3.108}$$

With the aid of (3.3), I can write the equation for brake forces in the front wheels where μ represents static friction and w is the weight force. We also know that the brake force is simply the amount needed to stop the vehicle's wheels from moving.

$$F_{bf} = K_{bf} \cdot F_{frwheel} \tag{3.109}$$

Where K_{bf} is the front brake force distribution factor which is < 1.

Back wheel forces

The same thing done for the front wheel can be done for the rear wheel you only need to change the index, the way to model the rear wheel forces is similar to the way used in the front wheels. All the equations used are the following

$$F_{rwheels} = \frac{T_{rwheels}}{R} \tag{3.110}$$

$$F_{rwheels} = \frac{1}{l}(mg \cdot l_2 + h(F_b + F_r \cdot mg)) \tag{3.111}$$

where l_2 is the distance from the car centre to the rear wheels, h is the car's height from the centre, and l is the distance between the front and rear wheel canters.

$$F_{br} = K_{br} \cdot F_{rwheels} \tag{3.112}$$

$$K_{br} + K_{bf} = 1 \tag{3.113}$$

where the rear brake force distribution factor is denoted by K_{br}. The Matlab model is available here [21].

4. ANALYSIS OF POSSIBLE SOLUTIONS

4.1 Selection Steering control algorithm

In this section I will discuss three different steering control algorithms, specifically PID, Stanley and MPC (model predictive controller) will be discussed here.

4.1.1 PID steering controller

The difference between a process variable that is measured and a setpoint that is desired is how a PID controller determines an error value. The controller modifies the process control inputs in an effort to reduce the error. The P-term, which stands for proportional, the I-term for integral, and the D-term for derivative make up the PID control signal.

Proportional Term (P)

An output value proportionate to the current error value is generated by the proportional term. The proportional gain constant, K_p, can be used to modify the proportional response by multiplying the error by it.

$$P_{\text{out}} = K_p \cdot e(t) \tag{4.1}$$

Integral Term (I)

The accumulation of previous mistakes is the subject of the integral term. The integral term rises when the error persists for a longer amount of time, which makes the controller work harder to return the process variable to the setpoint. The strength of the integral term is determined by the integral gain constant K_i.

$$I_{\text{out}} = K_i \int_0^t e(\tau) d\tau \tag{4.2}$$

Derivative Term (D)

The derivative term is an estimation of the error's future based on its rate of change right now. It enhances system stability and produces a damping effect. The derivative

term's effect is influenced by the derivative gain constant K_d.

$$D_{\text{out}} = K_d \frac{d}{dt} e(t) \tag{4.3}$$

Control Signal

The final control signal $u(t)$ is the sum of the proportional, integral, and derivative terms.

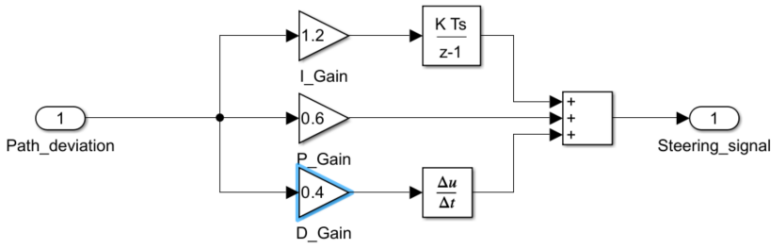
$$u(t) = P_{\text{out}} + I_{\text{out}} + D_{\text{out}} \tag{4.4}$$

Effects of PID Terms

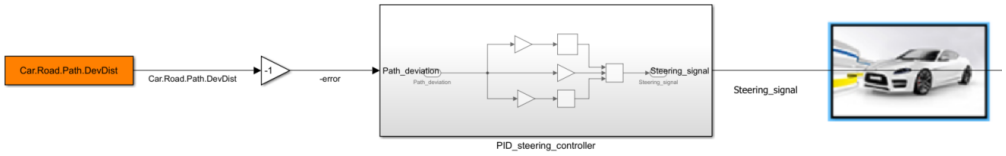
- The **P-term** provides a control signal to reduce the current error. However, it may lead to a steady-state error and oscillations if used alone.
- The **I-term** eliminates the steady-state error by integrating the error over time. It can cause the system to become too responsive and may lead to overshoot if not properly tuned.
- The **D-term** predicts future errors and smoothens the control action, reducing overshoot and improving stability. However, it can amplify noise in the error signal if the gain is too high.

The first controller to try is the PID controller,there can be seen the equation4.5 and the implementaion in Simulink in fig4.1

$$u(t) = K_p \cdot e(t) + K_d \cdot \frac{de(t)}{dt} + K_i \cdot \int_{t_0}^{t} (e(t) \cdot dt) \tag{4.5}$$



(a) PID controller



(b) Input and output of the PID controller connected to the vehicle steering input

Figure 4.1: Implementation of a PID controller for steering

4.1.2 Stanley controller

The Stanford Racing Team at Stanford University created the Stanley controller, a geometric trajectory tracking controller, for their autonomous vehicle known as ”Stanley” that competed in the 2005 DARPA Grand Challenge. The Stanley controller uses both heading and cross-track errors for the same purpose as the pure pursuit controller that was previously discussed, which only uses the cross-track error to determine steering actions. Instead of using the lookahead point that the pure pursuit controller uses, the cross-track error, or e_{Δ}, is defined with respect to the nearest point on the reference trajectory. In the meantime, the vehicle's heading in relation to the reference trajectory determines the heading error, which is represented by the symbol e_{ψ}. The ego vehicle's wheelbase is represented as L, its forward velocity as v_f, its steering angle as δ, and the frame of reference is the centre of the front axle. In Figure 4.2, the geometric relationships are shown.

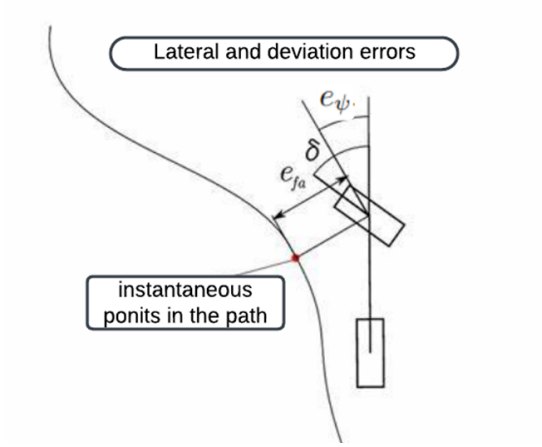


Figure 4.2: Stanley geometry (terms description)

The steering command is determined by the Stanley controller by accounting for both heading and cross-track errors. Remember that the steering actuation limits must be followed by the steering angle produced by the Stanley controller and other techniques. As such, the primary goals of the Stanley control law are to satisfy the subsequent three specifications:

1. **Heading Error Correction:** To make the vehicle's heading align with the intended heading, a steering control action δ proportional to the heading error e_ψ is produced.

$$e_\psi = \delta$$

2. **Cross-Track Error Correction:** The goal of cross-track error correction, or e_Δ correction, is to achieve coupled control by producing a steering control action δ that is directly proportional to the cross-track error and inversely proportional to the vehicle's velocity v_f. In addition, the use of an inverse tangent function can limit the effect for large cross-track errors.

$$e_\Delta = y - y_r$$

The steering control action δ is given by:

$$\delta = \arctan\left(\frac{k \cdot e_{\Delta}}{v_f}\right)$$

where k is a tuning parameter that determines the controller's sensitivity.

3. **Combined Heading and Cross-Track Error Correction:** The final steering command is a combination of the heading error correction and the modified cross-track error correction. The steering control action δ is obtained by summing the heading error e_{ψ} and the modified cross-track error correction:

$$\delta = e_{\psi} + \arctan\left(\frac{k \cdot e_{\Delta}}{v_f}\right)$$

This combined control action helps the vehicle align its heading with the intended direction while minimizing lateral deviations from the reference trajectory.

4.1.3 MPC controller

STATE space model

A useful control technique that can methodically take into account system operating constraints and future predictions of plant response during the design phase is called model predictive control, or MPC. This makes it a good option for path-tracking in situations where the system must meet both physical and critical safety constraints, such as actuator saturation and collision avoidance, in a dynamic traffic environment [4]. The control signal for vehicle actuators in the path-tracking application of the model predictive control algorithm is typically obtained by solving the optimisation problem, which is typically made up of the constraints of the control actions as well as the plant outputs, along with an optimisation objective of safety and tracking performance [7].

To forecast the vehicle's future motion state and absolute position given the current control input, a vehicle model is employed. This prediction is used to optimise a performance index under operating constraints at each time step t in relation to a series of upcoming input moves. The control action given to the plant at time t is the first of these ideal actions. As illustrated in Algorithm1, a new optimisation is solved at time t_{C1} over a shifted prediction horizon. The MPC-based path-tracking method is special because,

as was previously mentioned, it continuously optimises performance indexes by gathering data on the position of the vehicle, heading angle, and obstacles while it is moving. Thus, the secret to ensuring the controller's accuracy and real-time performance is to figure out how to choose a suitable optimisation solver and the right performance index based on the path-tracking task. The block diagram for model predictive control for path-tracking is shown in Figure 4.3. With this diagram, the controller can determine the front steering angle's future input sequence for avoiding collisions within a given horizon.

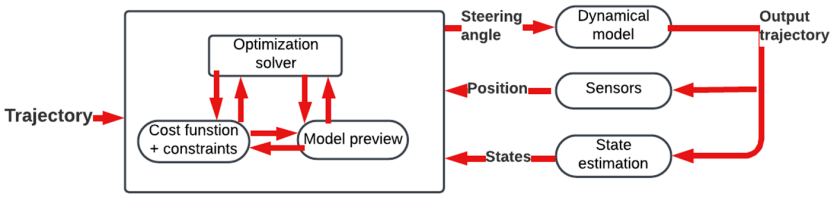


Figure 4.3: MPC Block diagram

The discrete error dynamics models are typically used in most MPC applications to predict the state of our vehicle at time t_{C1} from the last state at time t, as MPC is typically implemented with digit computers. The errors in the model are primarily lateral distance error and heading angle error. From the last state, we can easily determine the location, heading angle, and speed using the kinematic bicycle model. The vehicle model with discrete time can be embodied by

$$x_{k+1} = f(x_k, u_k) \& y_k = g(x_k) \tag{4.6}$$

Vectors include the states xk, controls uk, and outputs yk. MPC is an optimal control technique that minimises an objective function to determine control inputs. A crucial stage in the process is designing the objective function, which is definable in terms of the current and anticipated system variables. The following scalar performance measure

is used by an MPC controller for output reference tracking.

$$
\begin{aligned}
\min_{U_t} \quad & J_N(x_t; U_t) = \sum_{k=t}^{t+N-1} \text{cost}(x_{k;t}, u_{k;t}) \\
\text{subject to} \quad & x_{k+1;t} = f(x_{k;t}, u_{k;t}) \quad k = t, \ldots, t+N-1 \\
& x_{k;t} \in X \quad k = t, \ldots, t+N-1 \\
& u_{k;t} \in U \quad k = t, \ldots, t+N-1;
\end{aligned}
\tag{4.7}
$$

where N is the prediction horizon and the symbol $x_k; t$ denotes the variable x at time k predicted at time t. is the system's condition and, when path-tracking, it also includes the yaw angle and lateral position. The steering angle, braking torque, and throttle are all included in u, the control input to the system. Based on the current states and control inputs, the function $fx_t; u$ makes it possible to predict the future states of the system. The state and input constraints that the controller must adhere to are represented by the second and third terms in Equation 4.7(part b). Equation 4.7(part a) allows for the cost to be any performance index, which can include penalties for inputs, penalties for state tracking errors, penalties for input change rates, etc.

The co-simulation block diagram of MPC application in path-tracking is established in accordance with the previously mentioned basic principles of MPC, as section 7.1.2 illustrates. In the case of collision avoidance, where the actuators' intervention is essential in the path-tracking actions, MPC can be applied to systems that are either overly or under actuated.

Algorithm 1 Adaptive Model Predictive Control Pseudo Code

Define the plant model structure as $x(k+1) = Ax(k) + Bu(k) + Bd(k) + Bv(k)$
$y(k) = Cx(k) + Dd(k) + Dv(k)$, where A, B, C, D are matrices that can vary with time
Define the state, input, output, disturbance, and reference vectors
Define the MPC controller parameters such as prediction horizon, control horizon, weights, constraints, etc.
Define the adaptive MPC algorithm parameters such as forgetting factor, covariance matrix, etc.
Initialize the plant model parameters with some nominal values
Initialize the state estimate with some initial value
Initialize the adaptive MPC controller with the plant model and the controller parameters
for $k = 1 : N$ **do**
    $y(k) = ...$
    $d(k) = ...$
    $r(k) = ...$
    STATE $x_{\hat{k}} = ...$
    $A(k)$, $B(k)$, $C(k)$, $D(k) = ...$
    $mpc_controller.Model = ...$
    $u(k) =$ mpcmove$(mpc_controller, x_{\hat{k}}, y(k), r(k))$
    $x(k+1) = ...$
end forreturn $encontrado$ =0

Additionally, an optimisation goal is provided by the cost function with system constraints, which must be resolved during each sampling period. This gives the MPC an advantage over other control technologies and guarantees dependable control actions for collision avoidance. Its simple tuning options are another benefit of MPC over other controller types. This covers control horizons, the constraints concept, and the future prediction of the manipulative input. However, the fundamental MPC control strategy needs to be enhanced for a complicated collision avoidance scenario. Multiple objective functions, reference trajectories, and the requirement for Nonlinear Model Predictive Control (NMPC) can all be used to achieve this. Better computing devices have made it feasible to implement more complex MPC formulations that were previously impractical.

4.2 Selection of longitudinal controllers

In this section I am discussing various techniques to solve the speed control issue along with braking, two algorithms are implemented the first one is the most commonly used, the PID controller and the second one which I implemented here in this thesis is a simple linear model predictive controller with a cost function which will be described later.

4.2.1 PID speed and brake controller

As described in section 4.1.1, PID controller is the most commonly used in longitudinal controllers, as it is simple and does not require many inputs it works with single input single output, where the input is the speed error which can be seen in equation 4.8, and the outputs is the control signal (u), the general formula was described before in section 4.1.1 and equation 4.5, the effect of each term in the PID controller (K_p, K_i, K_d) was also described here 4.1.1

$$\Delta e_{speed} = V_{current} - V_{desired} \tag{4.8}$$

MATLAB implementation For the implementation I used a MATLAB function that supports code generation and to have more flexibility in editing the source code of the algorithm based on the path curvature and minimum safe speed during cornering, the algorithm is explained in 2 and in 3, in figure 4.4 you can see the blocks in simulink where path curvature is calculated in another block and being fed into the controller to adjust the speed based on the curvature as well as the safest speed during cornering that is also calculated in another block as well.

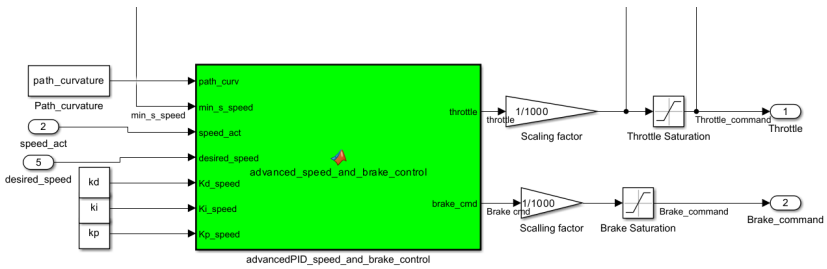


Figure 4.4: PID speed controller implementation in MATLAB/Simulink

Algorithm 2 PID Speed and Brake Control

```
procedure PIDSpeedAndBrakeControl(path_curv, min_s_speed, speed_act, desired_speed, Kd_speed, Ki_speed, Kp_speed)
Initialize control variables to zero
Calculate speed error and update integrator
Compute throttle using PID control
Calculate brake error and update integrator
Compute brake command using PID control
Adjust brake command based on path curvature
Update previous errors
return throttle and brake command
end procedure=0
```

Path curvature

Path curvature is a measure of how much a path deviates from a straight line. It is defined as the inverse of the radius of the circle that best approximates the path at a given point. The formula for path curvature is:

$$\kappa = \frac{1}{R} \tag{4.9}$$

where κ is the path curvature and R is the radius of curvature.

To calculate the path curvature from a given set of coordinates (x, y) that describe the path, we can use the following formula:

$$\kappa = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}} \tag{4.10}$$

where x', y', x'', and y'' are the first and second derivatives of x and y with respect to a parameter t that represents the distance along the path.

Minimum safe speed is the maximum speed that a vehicle can travel on a curved path without skidding or overturning. It depends on the path curvature, the gravitational acceleration, and the friction and banking forces that act on the vehicle. It can be estimated by the following formula:

$$v_{min} = \sqrt{g \cdot \frac{1}{|\kappa|}} \tag{4.11}$$

where v_{min} is the minimum safe speed, g is the gravitational acceleration, and κ is the path curvature.

In this formula, we assume that the path is horizontal and that the friction and banking forces are negligible. If the path is banked at an angle θ, we can modify the formula as follows:

$$v_{min} = \sqrt{g \cdot \frac{1}{|\kappa| \cdot \cos\theta}} \tag{4.12}$$

If the friction and banking forces are not negligible, we can use a more general formula that involves the maximum superelevation e_{max} and the maximum side friction factor f_{max}:

$$v_{min} = \sqrt{g \cdot \frac{R}{15(e_{max} + f_{max})}} \tag{4.13}$$

where R is the radius of curvature.

Term	Meaning
x and y	the coordinates of the path in a Cartesian plane
t	a parameter that represents the distance along the path
x', y', x'', and y''	the first and second derivatives of x and y with respect to t
κ	the path curvature, which is the reciprocal of the radius of curvature R
v_{min}	the minimum safe speed, which is the maximum speed that a vehicle can travel on the path without skidding or overturning
g	the gravitational acceleration, which is approximately 9.81 m/s^2 on Earth
θ	the angle of banking of the path, which is the angle between the horizontal plane and the plane of the path
e_{max}	the maximum superelevation, which is the ratio of the height difference between the inner and outer edges of the path to the width of the path
f_{max}	the maximum side friction factor, which is the ratio of the friction force to the normal force acting on the vehicle

Table 4.1: Terms and meanings related to path curvature and minimum safe speed

Table 4.1 gives the meaning for the terms used in path curvature calculation. Furthermore, the Simulink implementation is shown in figure 4.5.

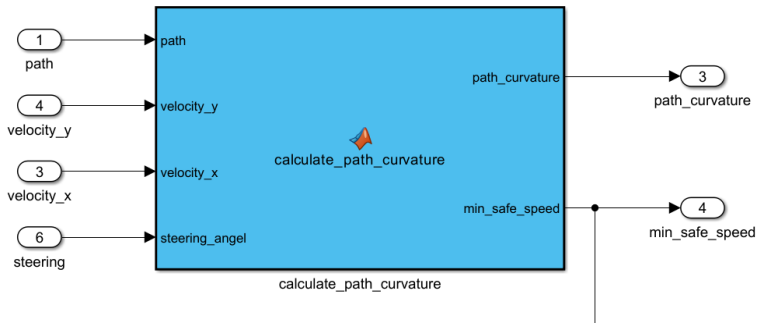


Figure 4.5: Path curvature calculation Simulink block

The final algorithm for PID speed controller is shown in algorithm 3.

Algorithm 3 PID Speed and Brake Control considering path curvature

0: **procedure** PIDControl(path_curv, min_s_speed, speed_act, desired_speed, Kd, Ki, Kp)

1: **Init:** init_persistent()
2: speed_error ← desired_speed - speed_act
3: integrator_speed ← update_integrator(integrator_speed, speed_error)
4: throttle ← compute_throttle(speed_error, integrator_speed, Kd, Ki, Kp)
5: brake_error ← -speed_error
6: integrator_brake ← update_integrator(integrator_brake, brake_error)
7: brake_cmd ← compute_brake(brake_error, integrator_brake, Kd, Ki, Kp)
8: path_curvature ← max(path_curv)
9: min_safe_speed ← max(min_s_speed)
10: predictive_brake_factor ← speed_act / 100
10: **if** path_curvature > 0 **and** speed_act < min_safe_speed **then**
11: brake_cmd ← brake_cmd + predictive_brake_factor ·(min_safe_speed - speed_act)
11: **end if**
12: update_prev_errors(speed_error, brake_error)
13: **Output:** throttle, brake_cmd
13: **end procedure**=0

4.2.2 MPC-based Speed and Brake Control Algorithm

The speed control algorithm is designed to regulate the vehicle speed towards a desired reference speed using a model predictive control. A cost function that minimises deviations from the reference speed and control input constraints is a part of the MPC formulation.

Algorithm 4 MPC-based Speed Control Algorithm

0: **procedure** AdvancedSpeedControlMPC(speed_ref, current_speed)
1: $Ts \leftarrow 0.1$ {Sampling time}
2: $N \leftarrow 20$ {Prediction horizon}
3: $Q \leftarrow \text{diag}([1, 0.1])$ {STATE weights (tune accordingly)}
4: $R \leftarrow 0.1$ {Input weight}
5: $S \leftarrow 0.1$ {Terminal state weight (integral term)}
6: $A \leftarrow \begin{bmatrix} 1 & Ts \\ 0 & 1 \end{bmatrix}$ {STATE matrix}
7: $B \leftarrow \begin{bmatrix} Ts \\ 0 \end{bmatrix}$ {Input matrix}
8: $lb \leftarrow 0$ {Lower bound on throttle}
9: $ub \leftarrow 1$ {Upper bound on throttle}
10: costFunction ← Function defining the cost function
11: Aineq ← [repmat$(B, N, 1)$] {Repeating B for N times}
12: Bineq ← [repmat$(speed_ref - current_speed, N, 1)$] {Repeating (speed_ref - current_speed) for N times}
13: $u0 \leftarrow$ zeros$(N, 1)$ {Initial guess}
14: $u_{\text{opt}} \leftarrow$ fmincon(costFunction, $u0$, Aineq, Bineq, ...) {Solve optimization problem}
15: throttle ← $u_{\text{opt}}[0]$ {Extract control input (throttle)}
16: **return** throttle
16: **end procedure**=0

The state-space equation for a one-dimensional system with vehicle speed v and throttle input u is given by:

$$\dot{v} = au \tag{4.14}$$

In this case, the matrix A is a scalar a, representing the system dynamics.

For a more complex two-dimensional system with position x_1 and velocity x_2, the

state-space equation is:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -a & -b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ c \end{bmatrix} u \tag{4.15}$$

Here, the matrix A is the 2×2 matrix $\begin{bmatrix} 0 & 1 \\ -a & -b \end{bmatrix}$, for simplicity the parameters, b, and a can be estimated using MATLAB parameter estimator app, which will be shown later in 6.2.4 section. The cost function is a function that measures the performance of a control input u for a given speed error. It is defined as:

$$\text{costFunction}(u) = (e + Bu)^T Q(e + Bu) + u^T Ru + S \sum u \tag{4.16}$$

where e is the speed error, B is a matrix that relates the control input to the speed error, Q and R are positive definite matrices that weight the speed error and the control input, and S is a scalar that penalizes the sum of the control input.

The original cost function in MATLAB code is:

```
costFunction = @(u) (speed_error + B * u)' * Q * (speed_error + B * u)
            + u' * R * u + S * sum(u);
```

The goal is to find the optimal control input u^* that minimizes the cost function, subjected to some constraints.

4.3 Selection of path planning algorithm

Path-planning is one of the key components required to achieve collision avoidance for self-driving cars, and it is a significant topic in the field of unmanned robots. Using data from highly automated driving maps [22] and onboard sensors for obstacle position and road shape, a path planner computes a workable, collision-free route to the destination in a complex traffic environment while meeting predetermined criteria. The following set of specifications can be used to characterise the path planning problem for collision avoidance.

The final outcome of path planning usually consists of a path or a trajectory, $\mathcal{W}[t_0; t_f] \rightarrow \mathcal{X}$, such that $\mathcal{W}(t_0) = x_0$ and $\mathcal{W}(t_f) = x_f \in \mathcal{X}_g$, where $\mathcal{X}$ is the vehicle's state space, and $\mathcal{X}_g$ is the goal region, x_0, x_f are the initial and final state in the

path, for some $t_0 \in R$ and $t_f \in R$ defined by the planner such that $t_0 \leq t_f$. When every time a planning algorithm links a time to each $\mathcal{W}(t)$ for all $t \in [t_0; t_f]$, $\mathcal{W}$ is subsequently called a trajectory [12].

Generally speaking, there are two types of path-planning techniques: global planners and local planners. The former uses the road map's previous information to generate the best possible path to the destination; it performs well when creating global paths that are either time- or distance-optimal, but it is less effective when responding to unforeseen obstacles. Local planners, on the other hand, recalculate a dynamic path to prevent collisions with other vehicles or obstacles. While they are effective in dynamic and initially unknown environments, they are particularly inefficient in complex environments (see [26]). I will provide here an overview of the primary methods used to address these two categories of path-planning issues.

4.3.1 Simple Dynamic Path Adjustment

The basic Dynamic Path Adjustment for the Obstacle Avoidance algorithm is implemented at first, for collision avoidance. The algorithm discussed in this thesis considers the vehicle's speed, dimensions, and the presence of obstacles to dynamically adjust the vehicle's path.

Algorithm Description

If there is an obstacle in the path of the vehicle, the algorithm detects it and computes a route to avoid it completely. The vehicle's speed, the obstacle's dimensions, and its relative position are used to calculate the trajectory. After that, the resulting path is modified to adhere to the revised trajectory, guaranteeing a safe separation from the obstruction. 5 contains the pseudo-code for the Basic Dynamic Path Adjustment algorithm.

Algorithm 5 Basic Dynamic Path Adjustment

```
function ObstacleAvoidance(Car_x, Car_y, dx, dy, w, l, path_x, path_y, v, v_ceil)
d ← max(v, 14), r ← √(dx² + dy²)
    if (r − l < d) and (r > 0) and (dx > 0) then
θ ← arctan 2(dy, dx), φ ← arctan 2(l, w)
return [−1 × (path_x + (v_ceil + l) × sin(θ + φ));
            −1 × (path_y + (v_ceil + w/2) × cos(θ + φ))]
    else
return [path_x; path_y]
    end if
end function=0
```

4.3.2 Spline Algorithm

A spline is a piecewise-defined polynomial function that is used for interpolating or approximating a set of data points. One commonly used spline type is the cubic spline, which uses cubic polynomials for interpolation see [5].

Cubic Spline Interpolation

Consider a set of n data points $\{(x_i, y_i)\}$, where $i = 1, 2, \ldots, n$. The goal is to find a smooth cubic polynomial between each pair of consecutive data points.

Let $S_i(x)$ be the cubic polynomial defined on the interval $[x_{i-1}, x_i]$. The general form of $S_i(x)$ is given by:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3 \tag{4.17}$$

The coefficients a_i, b_i, c_i, and d_i are determined by imposing certain conditions on the spline. These conditions typically include continuity of function values, first derivatives, and second derivatives at the data points.

The cubic spline is then expressed as the piecewise sum of the individual cubic polynomials:

$$S(x) = S_1(x) + S_2(x) + \ldots + S_{n-1}(x) \tag{4.18}$$

Equations for Coefficients

The coefficients of the cubic polynomials are determined by solving a tridiagonal system of linear equations. The system is constructed based on the conditions imposed on the spline.

Let $h_i = x_i - x_{i-1}$ and $f_i = \frac{y_i - y_{i-1}}{h_i}$, then the system of equations is:

$$2(c_{i-1} + c_i) = 3(f_i - f_{i-1}) \quad \text{(for each interval)} \tag{4.19}$$

$$c_0 = c_n = 0 \quad \text{(boundary conditions)} \tag{4.20}$$

The solution of this system provides the coefficients c_i for each interval, and other coefficients can be computed accordingly.

Advantages of Cubic Splines

Cubic splines offer smooth interpolation between data points and are widely used in computer graphics, computer-aided design, and numerical analysis.

Limitations

Although cubic splines offer an excellent trade-off between simplicity and smoothness, they have the potential to cause oscillations in the interpolating function, particularly when noisy data is present. Such situations may call for the use of higher-order splines or other interpolation techniques. [5]

4.3.3 Bezier Planning Algorithm

Bezier curves are widely used in computer graphics and path planning due to their simplicity and smoothness. The Bezier planning algorithm involves generating a path based on control points using Bezier interpolation.

Bezier Curve

A Bezier curve of degree n is defined by a set of $n+1$ control points $\{(P_0, P_1, \ldots, P_n)\}$. The curve is expressed as a weighted sum of Bernstein basis functions:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i \tag{4.21}$$

Here, $\binom{n}{i}$ is the binomial coefficient, and t ranges from 0 to 1.

For a cubic Bezier curve ($n = 3$), the equation becomes:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3 \tag{4.22}$$

Bezier Path Planning

In path planning, Bezier curves can be used to generate smooth paths based on a set of control points. The control points can be strategically chosen to achieve desired curvature and continuity properties.

Let P_0 be the starting point and P_n be the goal point. Intermediate control points $P_1, P_2, \ldots, P_{n-1}$ influence the shape of the path. The Bezier curve is generated using these control points, and the resulting path is a smooth trajectory from the start to the goal.

Advantages of Bezier Planning

- Bezier curves provide smooth and visually appealing paths.
- Control points allow for flexible shaping of the path.
- Easy to implement and understand.

Limitations

- Bezier curves may not provide the shortest path between two points.
- The influence of control points on the path may need to be adjusted for specific requirements.

4.3.4 Dijkstra's Algorithm

The Dijkstra algorithm's robust experimental performance and ease of debugging make it one of the simplest and most effective methods for solving global path planning.

The algorithm finds the shortest path between a target node in the graph and a given source node in a weighted graph, which may or may not be directed.

To apply the algorithm, each edge of the graph needs to have a non-negative weight [25], [36]. The Dijkstra algorithm's initial step involves using a graph to represent the open space in the surroundings and a graph search algorithm to find a path. The Dijkstra algorithm's path-planning phase starts with a set of possible nodes, or empty spaces, where the car can go, and it assigns a cost value to each one. The number of nodes that must pass through in order to reach each node is added to this value starting at the starting point, or home.

The algorithm can be summarized as follows:

1. Assign a current distance of 0 to the source node and infinity to the remaining nodes.

2. Assign the current node to the non-visited node that is currently the closest.

3. Determine the distance from the source node through each neighbouring node in the current node and compare it to the current distance. Update the neighbor's previous node and distance if it is smaller.

4. Remove the current node from the non-visited set and mark it as visited.

5. Continue steps 2 through 4 until either the non-visited set is empty or the destination node is visited.

The algorithm has the following mathematical expression:

Given a graph $G = (V, E)$, which has n nodes and m edges, let V represent the node set and E represent the edge set. Let $d(u, v)$ be the shortest path between nodes u and v, and let $w(u, v)$ be the weight of the edge $(u, v) \in E$. Let t be the destination node and s the source node. Define S as the visited nodes set and T as the non-visited nodes set. The previous node of node v in the shortest path from s to v is denoted by $p(v)$.

The algorithm can be initialized as follows:

$$
\begin{aligned}
&S = \emptyset \\
&T = V \\
&d(s,v) = \begin{cases} 0 & \text{if } v = s \\ \infty & \text{otherwise} \end{cases} \\
&p(v) = \text{undefined for all } v \in V
\end{aligned}
$$

The algorithm can be iterated as follows:

$$
\begin{aligned}
&\text{while } t \notin S \text{ and } T \neq \emptyset \\
&\quad u = \text{argmin}_{v \in T} d(s,v) \\
&\quad \text{for each } (u,v) \in E \\
&\qquad \text{if } d(s,u) + w(u,v) < d(s,v) \\
&\qquad\quad d(s,v) = d(s,u) + w(u,v) \\
&\qquad\quad p(v) = u \\
&\quad S = S \cup \{u\} \\
&\quad T = T \setminus \{u\}
\end{aligned}
$$

The algorithm can be terminated as follows:

$$
\begin{aligned}
&\text{if } t \in S \\
&\quad \text{return } d(s,t) \text{ and the path } s \to p(t) \to \cdots \to p(p(t)) \to t \\
&\text{else} \\
&\quad \text{return no path exists}
\end{aligned}
$$

Algorithm 6 show the whole algorithm in pseudo-code format, which can be implemented in MATLAB following it.

Algorithm 6 Dijkstra's Algorithm

```
function Dijkstra(start, goal, obstacles)
distances ← ∞(mapSize)
parent ← zeros(mapSize, 2)
current ← start
distances(sub2ind(size(distances), start(2), start(1))) ← 0
   while ¬isequal(current, goal) do
neighbors ← generateNeighbors(current, obstacles)
      for neighbor in neighbors do
         if ¬inObstacles(neighbor, obstacles) then
tentativeDistance ← distances(sub2ind(size(distances), current(2), current(1))) +
cost(neighbor, current)
            if tentativeDistance < distances(sub2ind(size(distances), neighbor(2), neighbor(1)))
then
distances(sub2ind(size(distances), neighbor(2), neighbor(1))) ←
tentativeDistance
parent(sub2ind(size(parent), neighbor(2), neighbor(1)), :) ← current
            end if
         end if
      end for
distances(sub2ind(size(distances), current(2), current(1))) ← ∞
[∼, idx] ← min(distances(:))
[row, col] ← ind2sub(size(distances), idx)
current ← [col, row]
   end while
return reconstructPath(parent, goal)
end function=0
```

4.3.5 A* path planning

In a graph, the shortest path between two nodes can be found using the A* algorithm. It functions by monitoring the distance that each node currently has from the source node and updating the distance if a shorter path is discovered. When the destination node is reached, or when every reachable node has been visited, the algorithm comes to an end.

The algorithm estimates the total cost of a path through node n from the source node to the destination node using a function $f(n)$. The sum of two functions, $g(n)$ and $h(n)$, is $f(n)$. The true path cost from the source node to node n is represented by the function $g(n)$. A heuristic function called $h(n)$ calculates the price of the least expensive route

from node n to the destination node. As long as the heuristic function never overestimates the true cost, it should be considered admissible. The entire algorithm, which can be used in Simulink as a MATLAB function, is shown in 7. Moreover, the following is a summary of the algorithm:

1. Make a set of nodes that have been visited but not expanded (open nodes) and a set of nodes that have been visited and expanded (closed nodes).

2. The source node should be added to the open set and given the value $f(n) = 0$.

3. Perform the following actions as long as the destination node is not in the closed set and the open set is not empty:

   (a) Assign the current node to the node in the open set with the lowest value of $f(n)$.

   (b) Add the current node to the closed set and remove it from the open set.

   (c) Perform the following actions for each current node's neighbour:

      i. Ignore the neighbour if it is part of the closed set.

      ii. If not, multiply the value of $g(n)$ for the current node by the cost of the edge connecting it to the neighbour to find the value of $g(n)$ for the neighbour.

      iii. Add the neighbour to the open set and give it the value $f(n) = g(n) + h(n)$ if it is not already there.

      iv. Compare the new and old values of $g(n)$ if the neighbour is in the open set. Update the value of $f(n)$ and the neighbor's parent if the new value is less.

4. Use the parent pointers to retrace the path from the destination node to the source node if it is part of the closed set.

5. There is no path from the source node to the destination node if the open set is empty and the destination node is not in the closed set.

Algorithm 7 A* Algorithm

```
function AStar(start, goal, obstacles) {Similar to Dijkstra's algorithm with modified heuristic}
distances ← ∞(mapSize)
parent ← zeros(mapSize, 2)
current ← start
distances(sub2ind(size(distances), start(2), start(1))) ← 0
  while ¬isequal(current, goal) do
neighbors ← generateNeighbors(current, obstacles)
    for neighbor in neighbors do
      if ¬inObstacles(neighbor, obstacles) then
tentativeDistance ← distances(sub2ind(size(distances), current(2), current(1))) + cost(neighbor, current)
        if tentativeDistance < distances(sub2ind(size(distances), neighbor(2), neighbor(1))) then
distances(sub2ind(size(distances), neighbor(2), neighbor(1))) ← tentativeDistance
parent(sub2ind(size(parent), neighbor(2), neighbor(1)), :) ← current
        end if
      end if
    end for
distances(sub2ind(size(distances), current(2), current(1))) ← ∞
[∼, idx] ← min(distances(:) + heuristic(goal, current))
[row, col] ← ind2sub(size(distances), idx)
current ← [col, row]
  end while
return reconstructPath(parent, goal)
end function=0
```

5. DESCRIPTION OF DETAILED SPECIFICATION

In this chapter the parameters of the vehicle are listed and discussed to have full view of the vehicle and its characteristics

5.1 Vehicle parameters setting

2-DOF vehicle

In Section 3.1.1, I discussed the mathematical model. In this section, I am going to specify the vehicle parameters used during the simulation and the state space equation filled with the values in Chapter 7. The Simulink model implementation will be shown.

As stated earlier, the two most important equations for a 2-DOF lateral vehicle dynamic model are the yaw rate and the vehicle velocity. Other vehicle dynamics parameters can be extracted from those two, using equations 3.10 and 3.9 along with the parameters shown in Table 5.1.

Parameter	Symbol	Value
Carbody mass	m	2100 kg
Carbody inertia	I	3904 kgm^2
Distance from CG to front axle	l_f	1.2 m
Distance from CG to rear axle	l_r	1.6 m
Slip stiffness rear axle	c_r	160000 N/rad
Slip stiffness front axle	c_f	100000 N/rad

Table 5.1: Vehicle parameters used in the simulation

The MATLAB code for the functions that calculate the lateral velocity and the yaw rate of the vehicle can be summarized in equations below where v_x is set to a thresh hold to avoid dividing by zeros, the threshold set is $0.15m/s = 0.54km/h$:

$$Vy_{\text{dot}} = \begin{cases} A + A2 + B3 & \text{if } |Vx| > 0.15 \\ 0 & \text{otherwise} \end{cases}$$

$$r_{\text{dot}} = \begin{cases} B1 + B2 + B4 & \text{if } |Vx| > 0.15 \\ 0 & \text{otherwise} \end{cases}$$

where

$$
\begin{aligned}
A &= -\frac{c_f + c_r}{mVx} Vy \\
A2 &= \left(\frac{l_r c_r - l_f c_f}{mVx} - Vx \right) r \\
B3 &= \frac{c_f}{m} \delta \\
B1 &= \frac{l_r c_r - l_f c_f}{IVx} Vy \\
B2 &= -\frac{l_f^2 c_f + l_r^2 c_r}{IVx} r \\
B4 &= \frac{l_f c_f}{I} \delta
\end{aligned}
$$

and the inputs and outputs are:

- Vx: velocity of the car x position
- Vy: velocity of the car y position
- r: yaw rate
- δ: steering angle
- Vy_{dot}: lateral velocity
- r_{dot}: yaw rate

For 3-DOF implementation the same parameters listed in table 5.1 are used as well, the 3-DOF equations are listed in section 3.1.2.

14-DOF CarMaker model

The model parameters can be set through vehicle data-set from the CarMaker Graphical user interface (GUI), no need to create the whole data-set you can just import an example and make your editing on the vehicle, I imported one of the vehicle examples and made my modifications on the vehicle, after setting the data-set you can do model check through CarMaker as shown in figure 5.1. In tables, 5.2, and 5.3 basic vehicle

parameters are listed. The whole list of vehicle parameters will be converted to excel file and I will upload it along with the model structure.

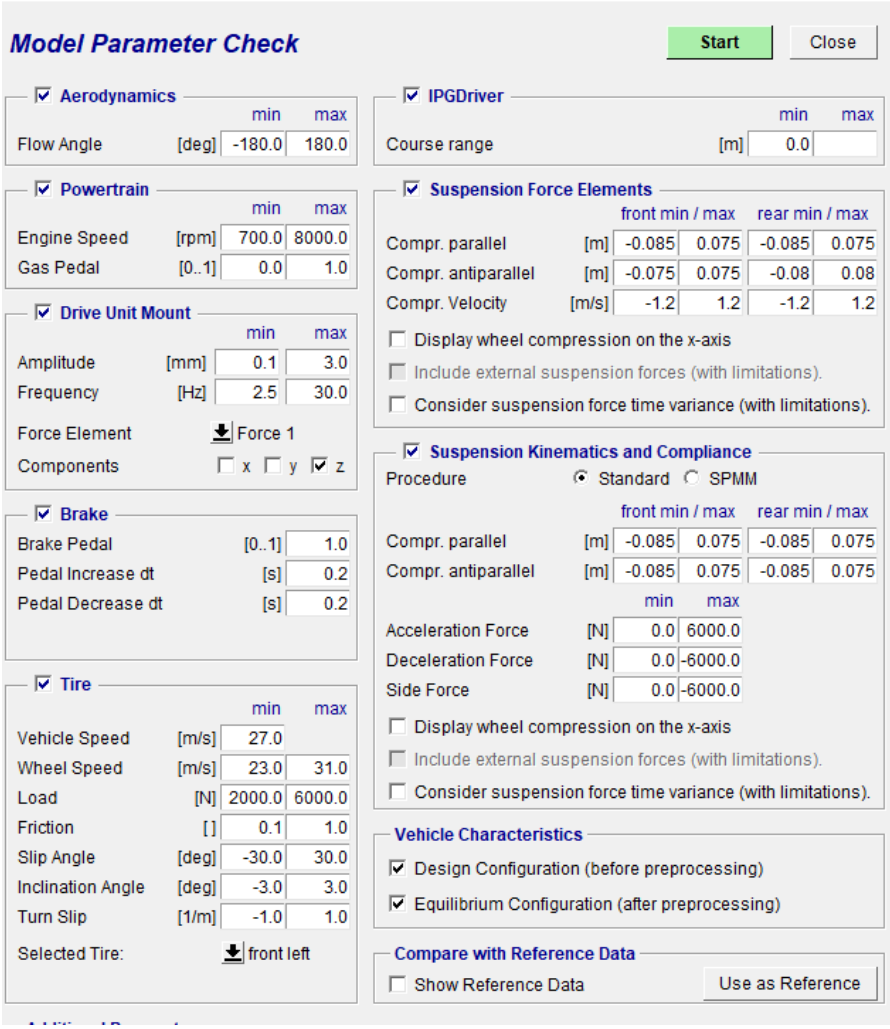


Figure 5.1: CarMaker model check

The vehicle Characteristics in Design Configuration are

- **Vehicle:** Demo_BMW_5
- **Susp Front:** McPherson_FrontAxle.skc
- **Tire front left and front right:** RT_225_55R17_p2.50
- **Tire rear left and rear right:** RT_225_55R17_p2.50

Basic Parameters

Parameter	Unit	Value
Total mass	kg	1600.004
Total weight	N	15689.639
Total roll inertia	kgm^2	696.215
Total pitch inertia	kgm^2	2700.750
Total yaw inertia	kgm^2	2986.624
Total CoG height (FrD)	m	0.562
Sprung mass	kg	1373.300
Sprung weight	N	13466.580
Sprung roll inertia	kgm^2	528.510
Sprung pitch inertia	kgm^2	2180.500
Sprung yaw inertia	kgm^2	2333.600
Sprung CoG height (FrD)	m	0.604
Unsprung mass (total)	kg	226.704
Wheelbase	m	2.975

Table 5.2: Basic Parameters

Axle Parameters

Parameter	Unit	Value
Track width	m	1.600
Toe Angle	deg	0.200
Camber Angle	deg	-0.000
Caster Angle	deg	-0.208
Caster Offset	mm	5.973
Kingpin Angle	deg	-2.561
Kingpin Offset	mm	-4.940
Normal force	N	0.000
Loaded tire radius	m	0.329
Tire rate	N/mm	386.204

(a) Front Axle Parameters

Parameter	Unit	Value
Track width	m	1.631
Toe Angle	deg	0.115
Camber Angle	deg	-1.490
Normal force	N	0.000
Loaded tire radius	m	0.329
Tire rate	N/mm	386.204

(b) Rear Axle Parameters

Table 5.3: Axle Parameters

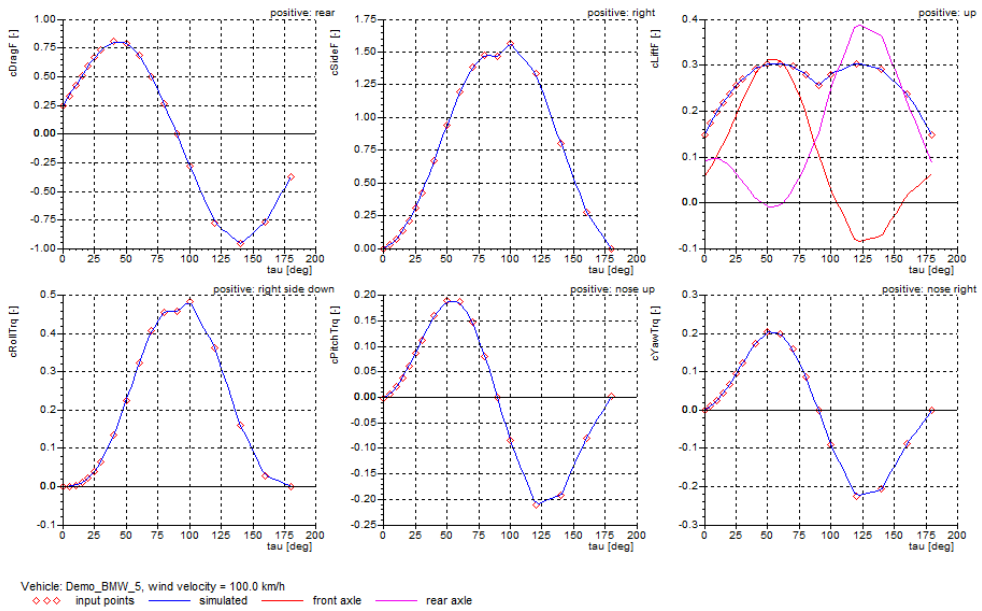


Figure 5.2: Aerodynamics

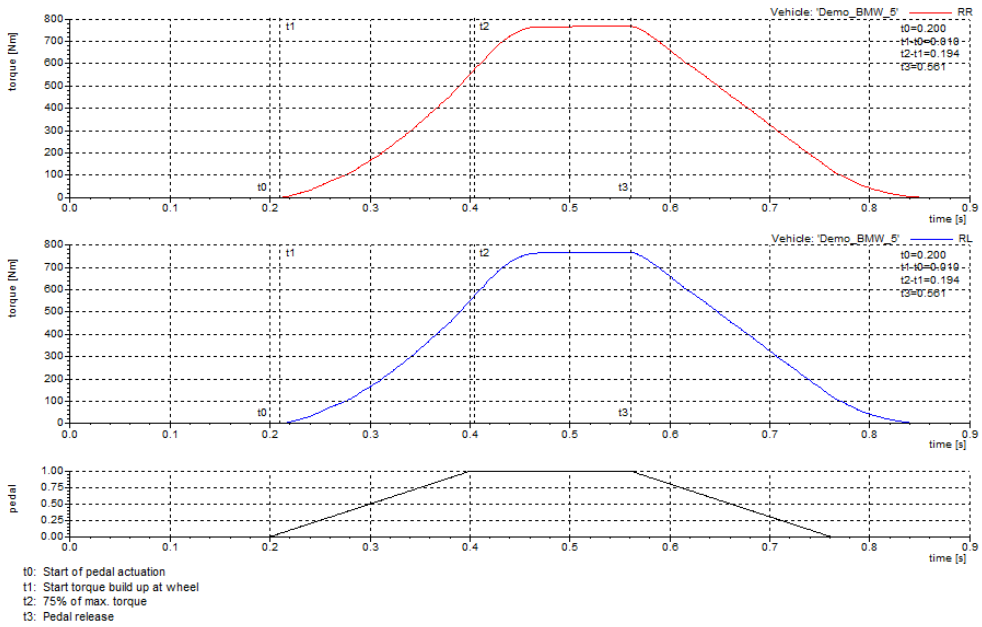


Figure 5.3: Rear axle Brake torque

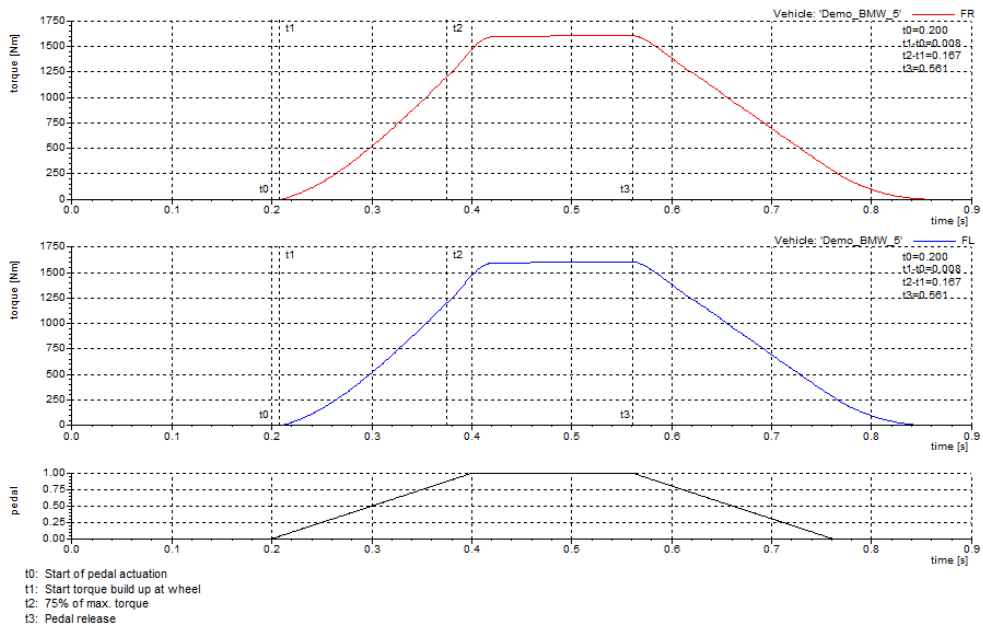


Figure 5.4: Front axle Brake torque

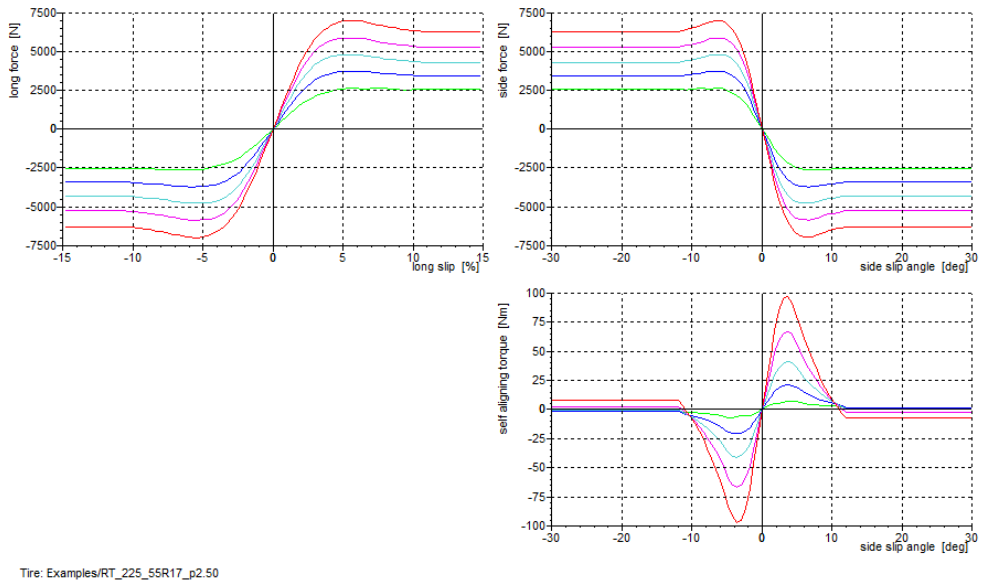


Figure 5.5: Tyre load characteristics as a function of slip with constant vertical load, and as a function of slip and vertical load with a constant side slip. Overview

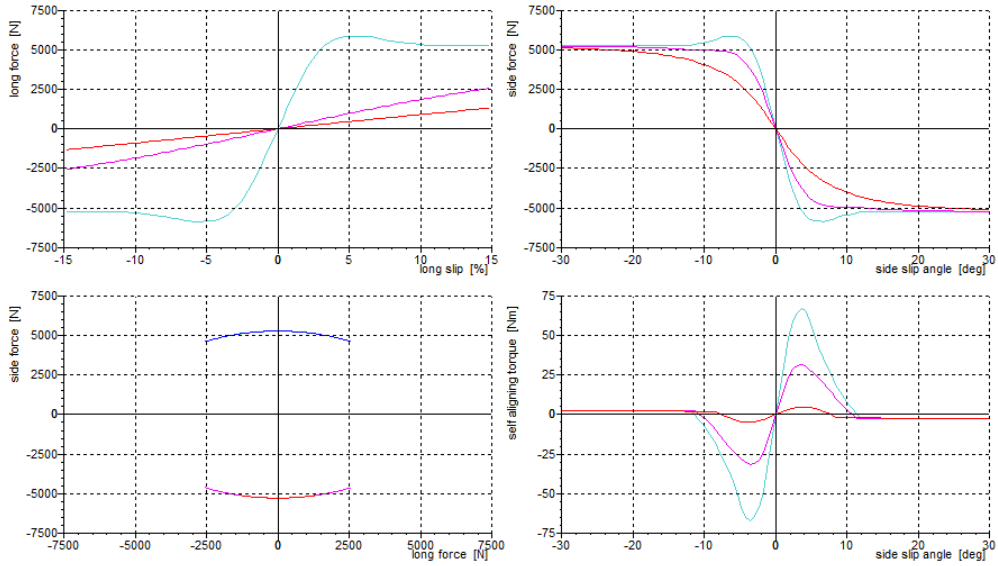


Figure 5.6: Tyre characteristics as a function of slip and side-slip with constant slip/side-slip, and as a function of side-slip with constant slip. Overview

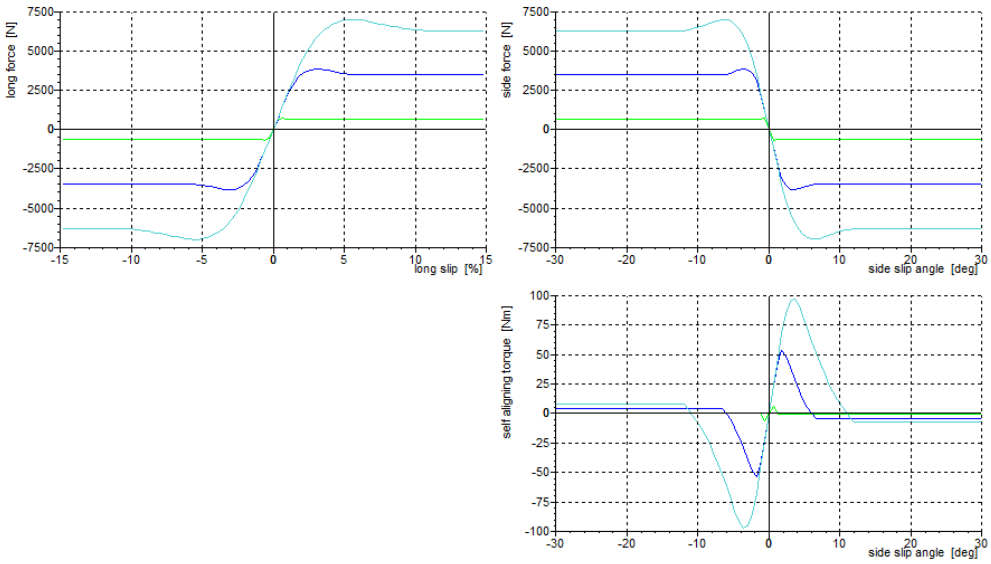


Figure 5.7: Tire friction characteristics as a function of slip with constant coefficient of friction (mu), and as a function of side-slip with constant sideways coefficient of friction (mus). Overview.

Figures from 5.2 to 5.7 show an overview of this 14-DOF vehicle dynamic simulated in CarMaker and connected to Simulink. The figures show the Aerodynamic, Axles brake torques, and tyre characteristics respectively.

5.2 Tune-able parameters and gains setting for steering and speed control

As described in chapter 4, for longitudinal and lateral controllers there are tun-able parameters that determine the controller behaviour and performance, in this section I will show the initial parameters setup, what are those parameters and how they were implemented in MATLAB/Simulink. In chapter (6) I will show the process and the tools used to tune them.

5.2.1 Steering controllers

In chapter (4), section (4.1) three steering controllers were defined the PID, Stanley and MPC, in table you can see those parameters and their initial values

Controller	Parameter	Initial Value	Description
PID	K_p (Proportional Gain)	1.0	Moderate response without excessive oscillations.
	K_i (Integral Gain)	0.1	Handle steady-state error and improve stability.
	K_d (Derivative Gain)	0.01	Dampen overshoot and minimize oscillations.
Stanley	k (Control Gain)	0.5	Balance tracking accuracy and controller responsiveness.
	Δt (Time Step)	0.02	Numerical stability and computational efficiency.
MPC	Prediction Horizon	10	Adequate future trajectory for optimal control.
	Control Horizon	3	Short-term control action for responsiveness.
	Q (Weight Matrix)	diag([1, 1, 1])	Prioritize tracking accuracy in state variables.
	R (Weight Matrix)	0.1	Penalize control effort to avoid aggressive maneuvers.

Table 5.4: Steering Controllers Parameters and Initial Values

The controllers algorithms, equations, and description were discussed previously,

refer to (4.1) to have more info.

5.2.2 Speed controllers

For speed controllers as shown in chapter(4), section(4.2), I used two types of speed controllers, the first one is the PID, the same initial values used in steering are set as well for speed, the second type is the simplified linear MPC, where the tunable paramters and their initial values are show in table 5.5

Parameter	Description	Initial Value
Sampling Time (T_s)	Time between consecutive control actions	0.1 seconds
Prediction Horizon (N)	Number of steps into the future considered	20
State Weights Matrix (Q)	Weights assigned to different state variables	$\text{diag}([1, 0.1])$
Input Weight (R)	Weight assigned to the control input	0.1
Terminal State Weight (S)	Weight for the terminal state (integral term)	0.1
Lower Bound on Throttle (lb)	Minimum allowable value for the control input (throttle)	0
Upper Bound on Throttle (ub)	Maximum allowable value for the control input (throttle)	1

Table 5.5: MPC-based Speed Controller Parameters

5.3 Tunable Parameters for Path Planning and Controllers

5.3.1 Cubic Spline Interpolation

The following table summarizes the tunable parameters for the cubic spline interpolation algorithm:

Parameter	**Description**
n	Degree of the spline (commonly cubic, $n = 3$)
h_i	Interval between data points
f_i	Slope or derivative at each data point

Table 5.6: Tunable Parameters for Cubic Spline Interpolation

5.3.2 Bezier Planning

The following table outlines the tunable parameters for the Bezier planning algorithm:

Parameter	Description
n	Degree of the Bezier curve (commonly cubic, $n = 3$)
Control Points	Positions of control points influencing the path
Start	Initial point for the path
Goal	Desired endpoint of the path

Table 5.7: Tun-able Parameters for Bezier Planning

6. DESCRIPTION OF WORK PHASES DURING DESIGN AND LESSONS LEARNED

6.1 Challenges and Potential Pitfalls

During the design process in Simulink, several challenges and potential pitfalls were encountered. This section discusses key issues faced and lessons learned throughout the development stages.

6.1.1 Tuning control parameters

Challenge: Achieving the target speed requires fine-tuning of the control parameters, especially in dynamic scenarios. I have to follow systematic approach to address the effect of each term/parameter to be tune and work based on that. It was discussed in section 5.2, For the tuning process, three approaches were employed. The first involved the use of a sliding gain block in Simulink wich can be seen in figure 6.1.

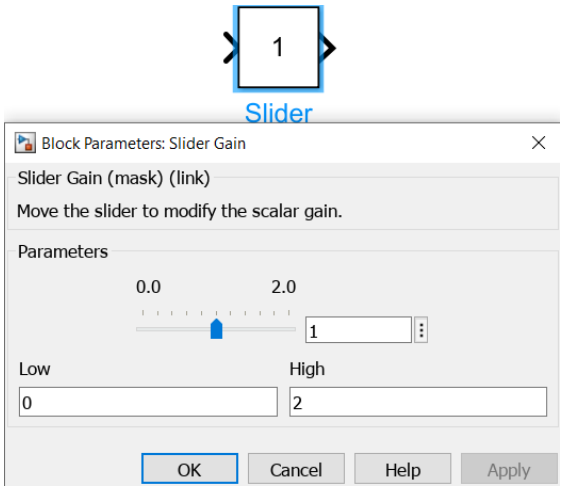


Figure 6.1: Simulink slider gain for parameter tuning

During model execution, the sliding gain was adjusted based on the desired behavior and value of the control signal. The second approach consisted of assigning minimum and maximum values, along with several intermediary values. Subsequently, the output values were recorded, and linear interpolation was performed on the obtained data. The third method utilized the MATLAB Parameter Estimator app. This approach allows for

setting the desired output, the signal to be controlled, and the parameters. While this method may be time-consuming in some cases, it provides highly accurate results. In contrast, the other two methods are less time-consuming but may not yield as accurate results.

6.1.2 Tuning Path Planning Parameters

Challenge: Tuning the parameters of the path planning algorithms to achieve optimal performance and trajectory smoothness presented difficulties, especially when dealing with dynamic environments.

Lesson Learned: It is imperative to approach parameter tuning methodically. Understanding how parameter changes affect the generated paths can be facilitated by using Simulink's visualisation tools and performing sensitivity analyses.

6.1.3 Controller-Path Planning Coordination

Challenge: Coordinating the actions of the steering and speed controllers with the output of the path planning algorithms to achieve a smooth and responsive vehicle motion proved challenging.

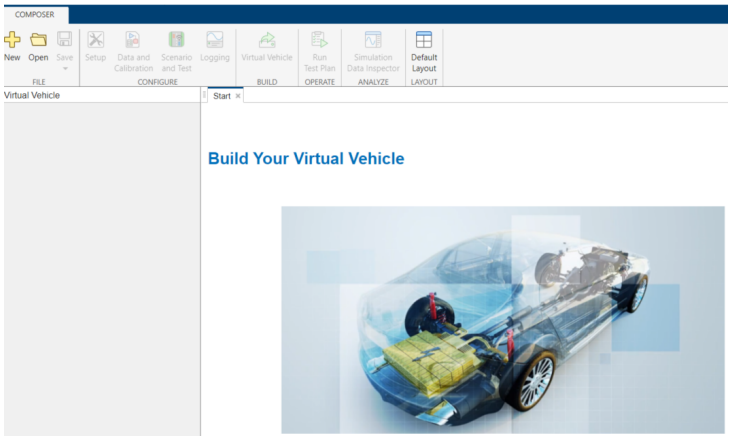
Lesson Learned: It is imperative to maintain close coordination between the control and path planning modules. Frequent feedback loops and testing in different conditions aid in locating coordination problems and improving the system as a whole.

6.2 MATLAB Useful Tools

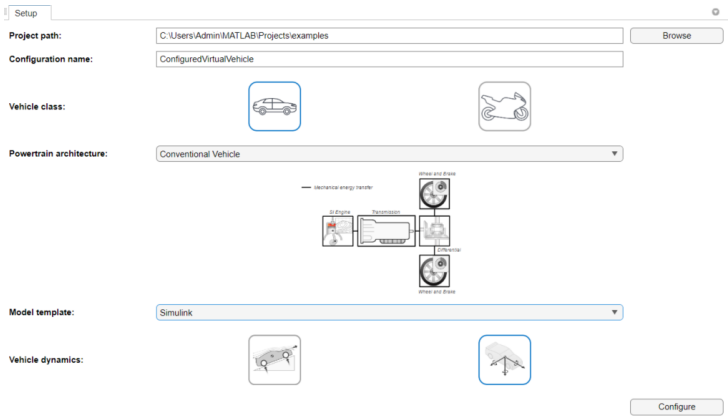
In this section, I will describe some of the useful tools in MATLAB that accelerated the controller design process. These tools either provided ready models for testing controllers or facilitated the tuning process to obtain finely-tuned final values. While there were other useful tools employed, this section focuses on the most frequently used ones.

6.2.1 Virtual Vehicle Composer

The Virtual Vehicle Composer is a powerful tool in MATLAB that allows engineers and researchers to create and analyze vehicle dynamics models. It provides a graphical user interface (GUI) shown in Figure 6.2 for assembling complex vehicle systems, where I can start a new vehicle setup as shown in figure 6.2.



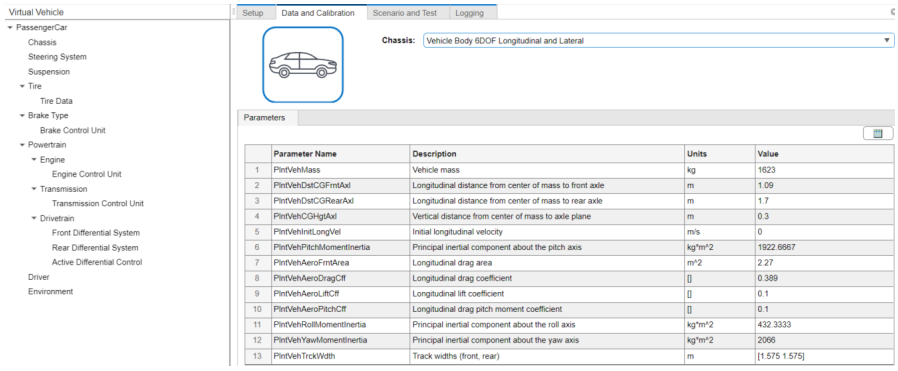
(a) Home Page



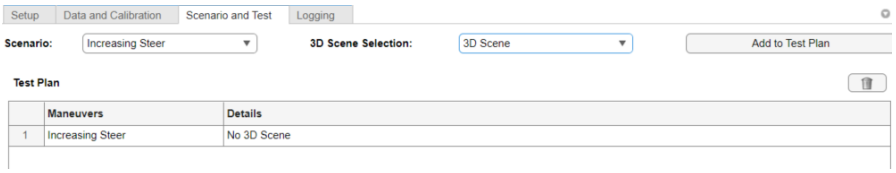
(b) Vehicle Setup

Figure 6.2: Virtual Vehicle Composer GUI

After that, further parameters can be set including the chassis, power-train, and other components. Users can define parameters, simulate vehicle behavior, and analyze performance, making it a valuable tool for designing and testing vehicle dynamics in a virtual environment. Figure 6.3 shows an overview of parameters setup GUI. It is also possible to setup a three dimensions (3D) environment simulation for the model as well.



(a) Data and Calibration



(b) Scenario and Test Setup

Figure 6.3: Data, Calibration, Scenario, and Test Setup

The outputs of this model is a MATLAB project that consists of the folder structure shown in figure 6.4.

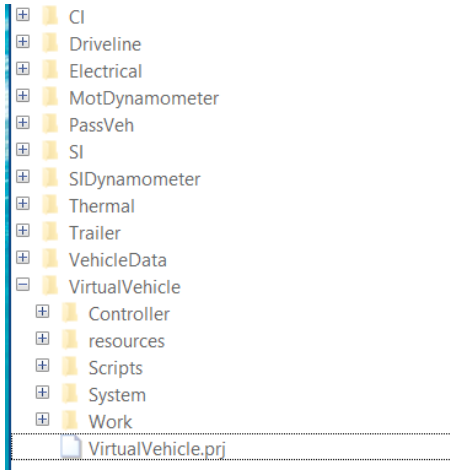


Figure 6.4: Virtual vehicle project folder structure

6.2.2 Driving Scenario Designer

This is an app designed for creating realistic driving scenarios in a virtual environment. With it I can generate complex scenarios with dynamic elements such as vehicles, pedestrians(like humans), and obstacles. This tool is particularly useful for testing and validating autonomous driving algorithms and control systems. As shown in figure 6.5 a vehicle with a road has been set, I set also a Camera and RADAR sensor for objects detection. In the figure it can be see the road parameters (yaw angle, lateral and longitudinal position), the road preview and sensor view.

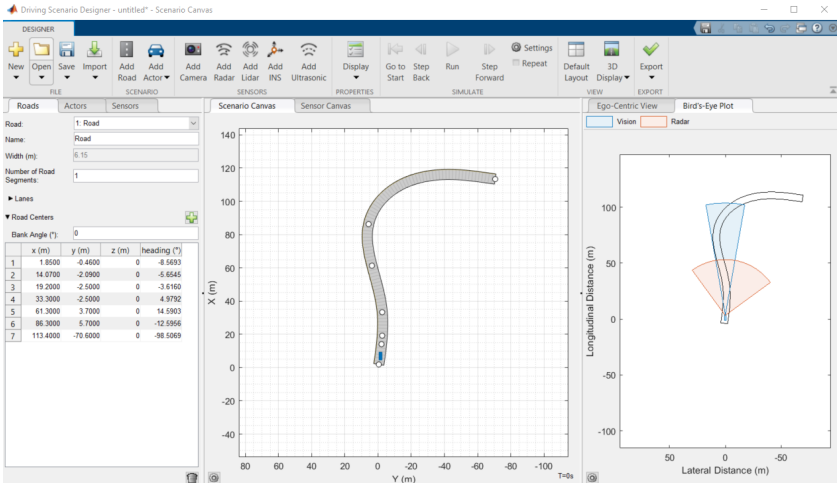


Figure 6.5: Driving scenarios builder in MATLAB

The model can be exported as a Simulink block as shown in figure 6.6, this Simulink can be then connected to the rest of the model. Further more we can edit the road in a 3D environment as shown in figure 6.7.

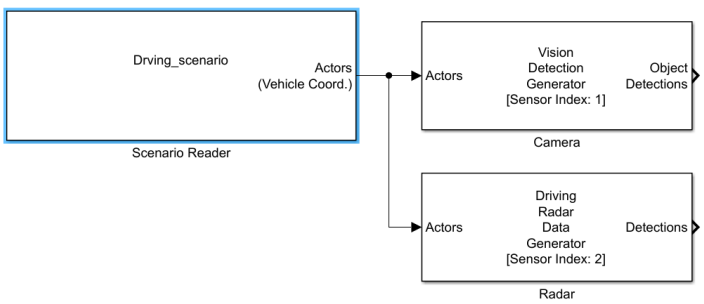


Figure 6.6: Exporting the scenario as a Simulink block

Figure 6.7: A 3D view of the road with the vehicle

6.2.3 Control System Tool box

Control System Toolbox provides wide variety of options, it can be used to design PID or MPC controllers, as well as tuning the parameters.I used it for designing, analyzing control systems, and optimize control system parameters. It provides a user-friendly interface for modeling, simulating, and tuning control systems. The interactive and graphical nature of the Control System Designer facilitates the rapid development of control strategies. Figure 6.8 shows the PID-controller designer app. I can set the response behaviour desired outputs, and overshoot analysis based on the requirements.

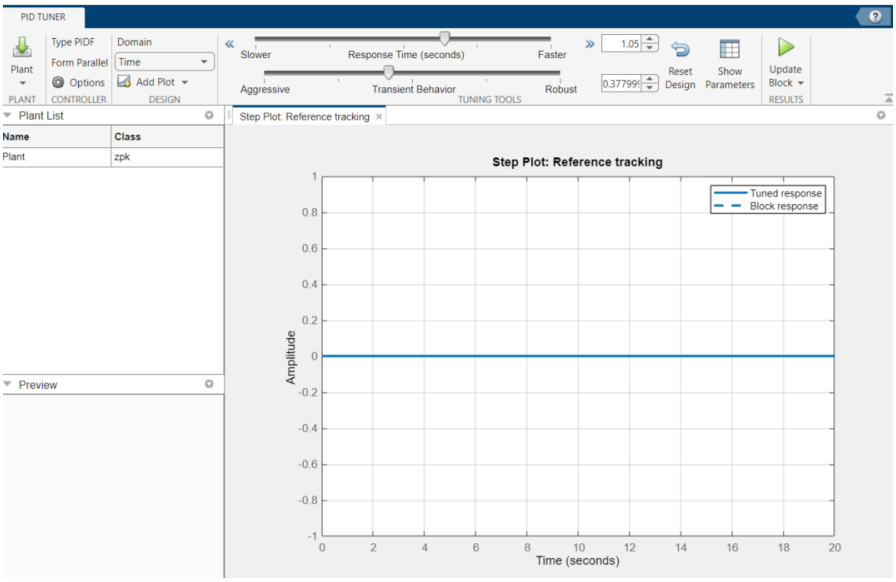


Figure 6.8: PID designer app

6.2.4 Parameter Estimation

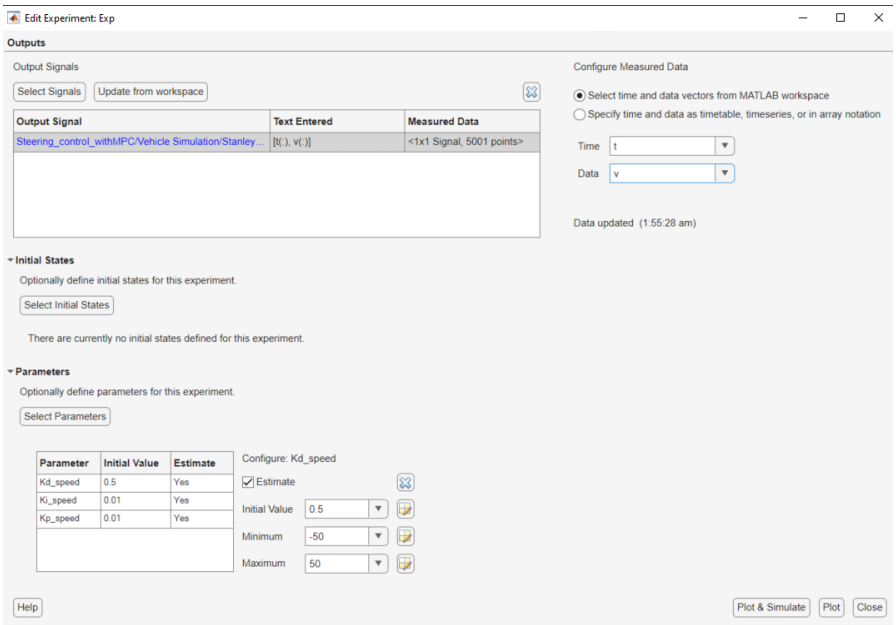


Figure 6.9: Set Experiment Parameters

MATLAB's Parameter Estimation app, provide a GUI for parameter selection and estimation. It is a powerful tool for identifying and estimating model parameters. It is particularly useful when dealing with complex models with uncertain parameters. I mentioned before that this tool was used to tune the parameters for the longitudinal and lateral controllers. The tool allows users to define models, specify input as a two dimensions (2D) array of time and the desired output, and then the output signal can be selected from the Simulink interface, and perform parameter estimation using various optimization algorithms. This capability is crucial for refining models, and aligning model behavior with real-world observations. After creating the experiment, you can edit the Experiment Parameters, select the parameters, set initial values, and define limits. Figure 6.9 provides an illustration of the process. The app also provides plot for the optimization process where you can see the parameters being updated as shown in figure 6.10 and the effect on the signal, so you can stop the optimization anytime.

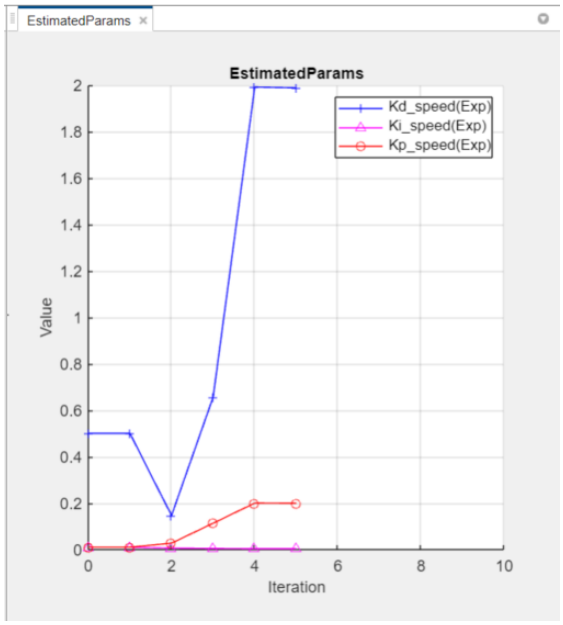


Figure 6.10: Estimated parameters values plot

6.3 Opportunities for Self-Checking Methods and Testing

Simulink model adviser

When it comes to modelling conditions and configuration settings that lead to an inaccurate or inefficient simulation of the system the model represents, the Model Advisor examines the model or subsystem. Verification of compliance with industry standards and guidelines can be facilitated by the Model Advisor checks. I can apply uniform modelling guidelines across development teams and projects by using the Model Advisor. Figure 6.11 shows the GUI of this tool.

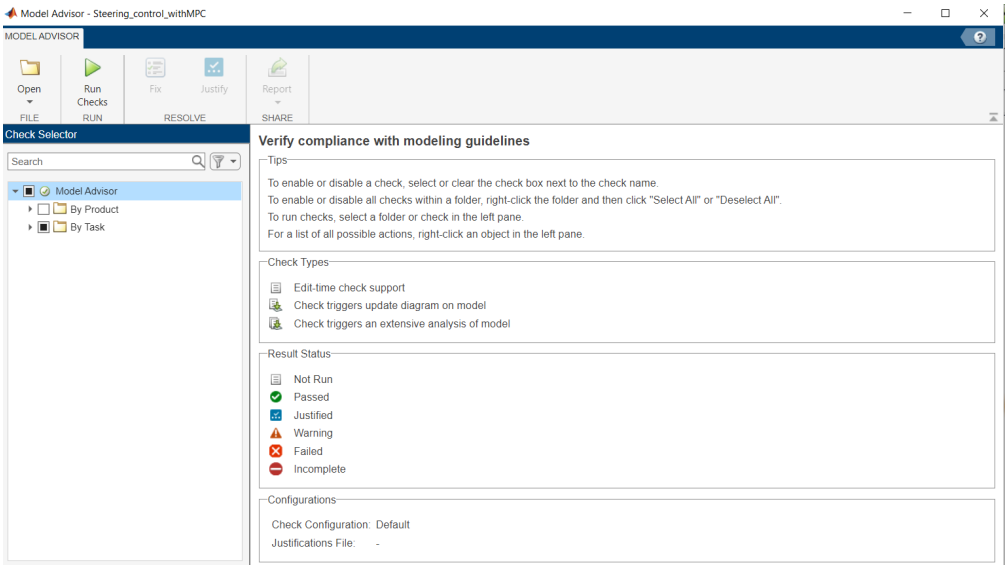


Figure 6.11: GUI of the model advisor

After the model has been thoroughly analysed, the Model Advisor generates a report outlining the less-than-ideal conditions, settings, and modelling methodologies along with, if appropriate, suggested solutions. I run the test on my model and get the following resutls, I will include here the relevant model checks.

Run Summary

Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
25	0	312	0	477	1240	2054

Modeling Physical Systems

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.1	0	0	0	0	1	1

Replacing Blocks That Will Be Removed

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.2	0	0	0	0	1	0

Inherit from Simulink.VariantControl

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.3	0	0	0	0	1	0

Simulation Runtime Accuracy Diagnostics

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.6	0	0	0	0	2	0

Simulink Model File Integrity

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.8	0	0	0	0	1	0

S-function Checks

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.9	0	0	0	0	1	0

Code Generation Efficiency

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.12	0	0	0	0	2	0

Model Referencing

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.15	0	0	0	0	6	2

Managing Library Links And Variants

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.16	0	0	0	0	4	0

Modeling Standards for Secure Coding (CERT C, CWE, ISO/IEC TS 17961)

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.19	6	0	3	0	3	15	27

Modeling Standards for DO-254

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.22	0	0	13	0	18	22	53

Modeling Standards for IEC 61508

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.23	2	0	32	0	29	36	99

Modeling Standards for IEC 62304

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.24	2	0	32	0	29	36	99

Modeling Standards for ISO 25119

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.26	2	0	32	0	29	36	99

Modeling Standards for EN 50128/EN 50657

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.27	2	0	32	0	29	36	99

Model Metrics

	Incomplete	Failed	Warning	Justified	Passed	Not Run
2.28	0	0	0	0	9	1

Modeling Standards for MAB

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.29	0	0	33	0	83	26	142

Modeling Standards for JMAAB v5.1

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.30	0	0	28	0	79	25	132

Modeling Standards for JMAAB v6

	Incomplete	Failed	Warning	Justified	Passed	Not Run	Total
2.31	0	0	36	0	88	28	152

7. DESCRIPTION OF IMPLEMENTATION

7.1 Models Build in Simulink environment

The first thing I implemented is a 2-DOF vehicle dynamic Simulink model, as stated before in chapter 5 section 5.1, I first calculated the yaw rate($\dot{\psi}$), and lateral accleration($\dot{v}_y$) in MATLAB function blocks, refer to 5.1 to see the logic implemented in MATLAB. Lateral velocity is calculated by integrating the lateral acceleration as shown in figure 7.1, there fore the longitudinal velocity can be calculated easily through $v_x = \sqrt{v^2 - v_y^2}$. Also by integrating $\dot{\psi}$ I got yaw angle ψ and it was implemented similar to the lateral velocity calculation.

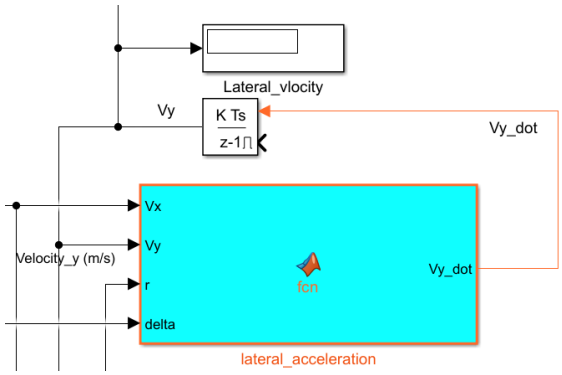


Figure 7.1: Lateral velocity derivation from lateral acceleration in Simulink

For global position calculation I used the rotational matrix around z axis mentioned here 3.41, and following the equation 7.1

$$\vec{s} = \int_{v_0}^{v_t} \vec{v} R_z \tag{7.1}$$

Where $\vec{s}$ is the position vector and $\vec{v}$ is the velocity vector, the Simulink implmentation is shown in figure 7.2.

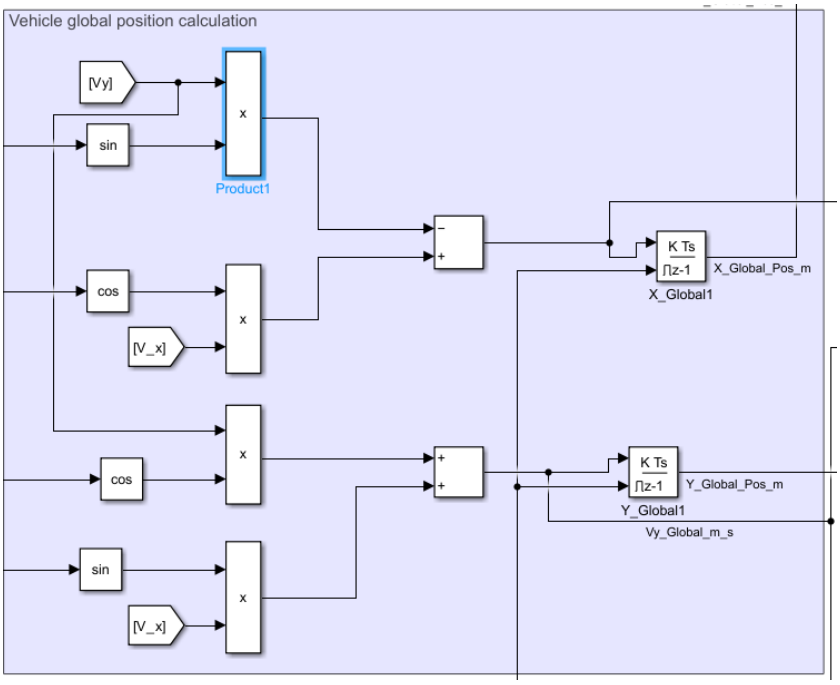


Figure 7.2: Global position calculation

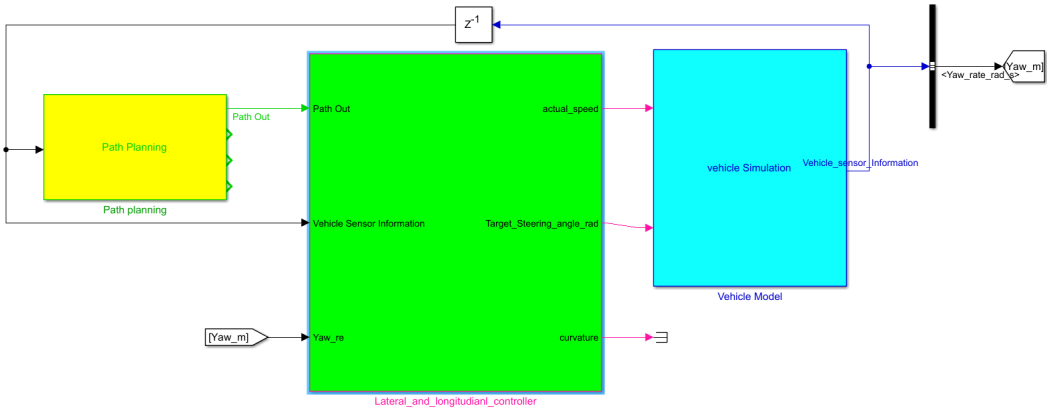


Figure 7.3: Overview of full Simulink Model

7.1.1 Longitudinal controller implementation

In this section I will show the implementation of both PID speed regulator, and MPC controller.

PID implementation

The PID algorithm is implemented as a MATLAB function block, the inputs are the control parameter that were tuned using parameter estimation app (see 6.2.4), the path curvature, and the minimum safe speed (in chapter 4 section 4.2.1 I explained the way I calculated both curvature and minimum safe speed for cornering). The algorithm followed was described in algorithm 3, the code can be seen in A.1.2.

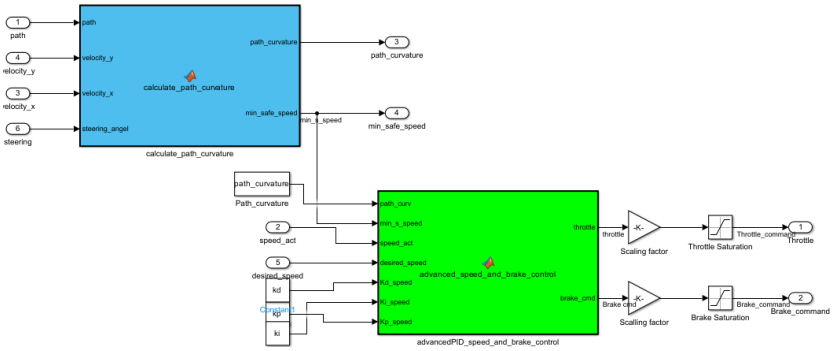


Figure 7.4: PID speed and brake regulators Simulink blocks

MPC speed regulator

The algorithm was described in (algorithm 4), I implemented the MPC using a MATLAB function block in Simulink the code can be seen in A.1.3. In figure 7.5 the model can be seen and in addition to that a basic logic was implemented to switch between the throttle and the brake command, the desired speed is calculated based on path curvature in separate block, then the speed difference is fed into the MPC controller, if the speed error is negative then a throttle should be applied otherwise, the brake should be applied.

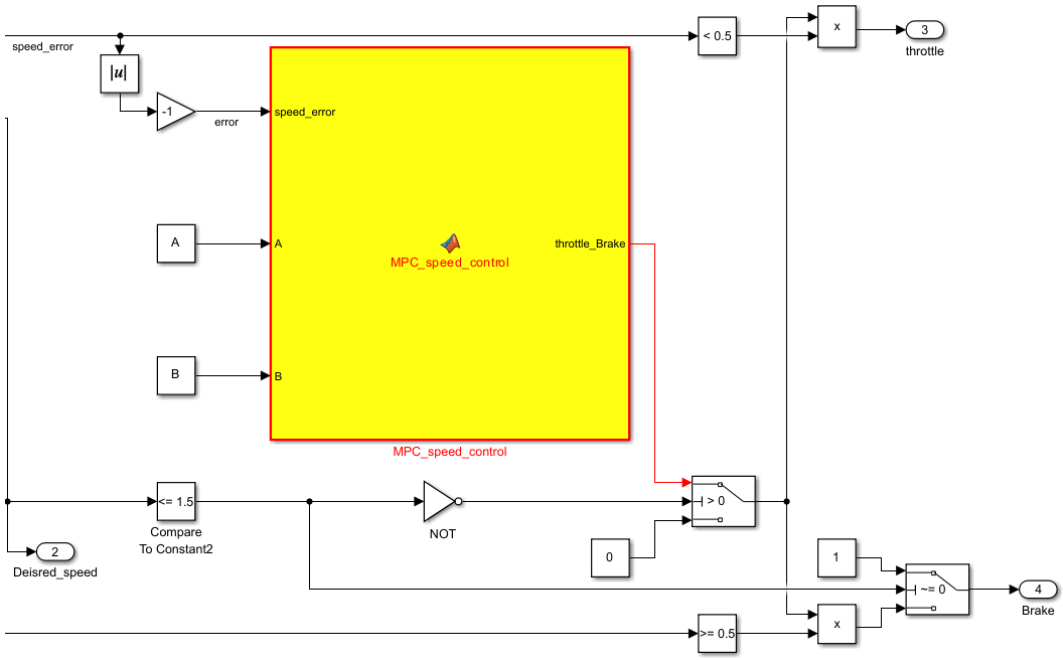


Figure 7.5: MPC speed regulator Simulink

To avoid oscillation in switching between the brake and the throttle, a constant value represents the controller steady state error is added to the switch logic so that the signal is only changed if we are above or below the steady state error.

7.1.2 Lateral controllers implementation

In this section I am describing the implementation of the three steering controllers, the PID, the Stanley, and the MPC.

PID steering controller

The controller was implemented with Simulink and the parameters were tuned using a the parameters estimation MATLAB app, a discrete integral and derivative blocks were used to avoid algebraic loop and to run smoothly in Simulink.

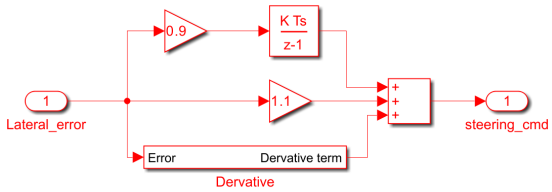


Figure 7.6: PID Steering Controller

The input to the controller is the path deviation (the lateral error or also called cross track error), lateral error is calculated as the perpendicular distance from the vehicle C.G to the instantaneous path segments near the vehicle. The perpendicular distance is calculated using the formula for the distance from a point to a line as shown in 7.2, where a,b,c are the line equation parameters, x_{cg} and y_{cg} are the vehicle center of gravity position. The code used for calculation can be seen in A.1.4

$$cte = \frac{a \cdot x_{cg} + b \cdot y_{cg} + c}{\sqrt{a^2 + b^2}} \tag{7.2}$$

Stanley controller

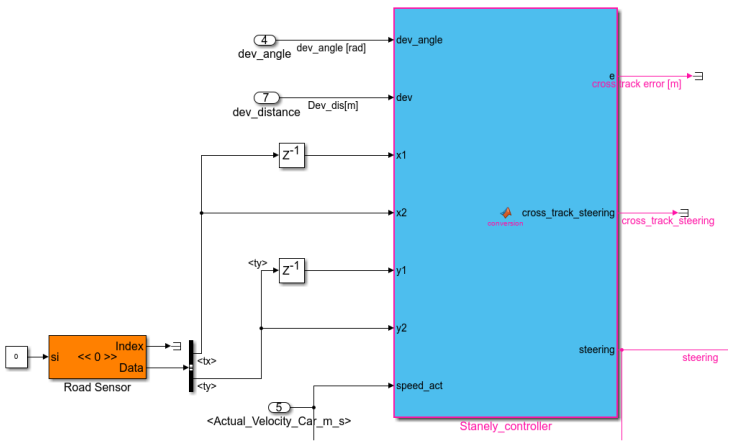


Figure 7.7: Stanley steering control

The equations and the algorithm is already discussed in section 4.1.2, in addition to the control a law mentiond in 4.1.2 I added tun-able parameters beside the Stanley gain, and made the Stanley gain dependent on the speed as it was advised by many papers as

shown in equaiton 7.3. The parameters are also tuned using the parameter estimation MATLAB app.

$$k_{stanley} = k_b + v^{k_{exp}} \tag{7.3}$$

The other input beside the lateral error, is the heading error (it is also called the deviation angle), the heading error can be calculated as a difference between the path heading and the vehicle heading, I calculated the path heading based on equation 7.4, where $\kappa$is the path curvature and v is the vehicle speed. Simulink implementation can be seen in 7.8, and the Stanley code can be seen in A.1.5 .

$$\psi_{path} = \int_{t_0}^{t} \kappa \cdot v dt \tag{7.4}$$

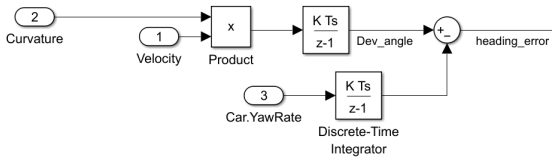


Figure 7.8: Path heading calculation

MPC steering controller

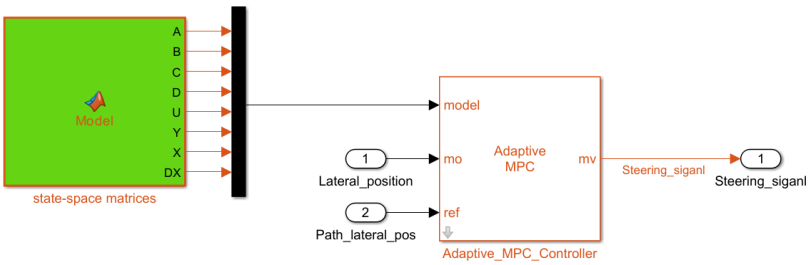


Figure 7.9: MPC controller Simulink blocks

This algorithm was described in section 4.1.3, following the algorithm specified in 1, I made the Simulink MATLAB function block based on it. To have a more precise control signal an adaptive MPC model is used, the model parameters are calculated based

on the state space system specified in the vehicle dynamic models, the code used to calculate and update the state space mercies can be seen here A.1.6, the state variables are defined as follows:

$$x_1 = \text{lateral velocity [m/s]}$$
$$x_2 = \text{yaw rate [rad/s]}$$
$$x_3 = \text{lateral position [m]}$$
$$x_4 = \text{yaw angle [rad]}$$

The input variables are defined as follows:

$$u_1 = \text{steering angle [rad]}$$
$$u_2 = \text{longitudinal velocity [m/s]}$$

The state-space model is defined as follows:

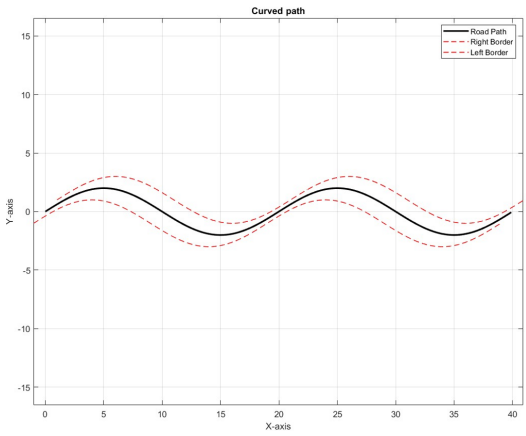
$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

where

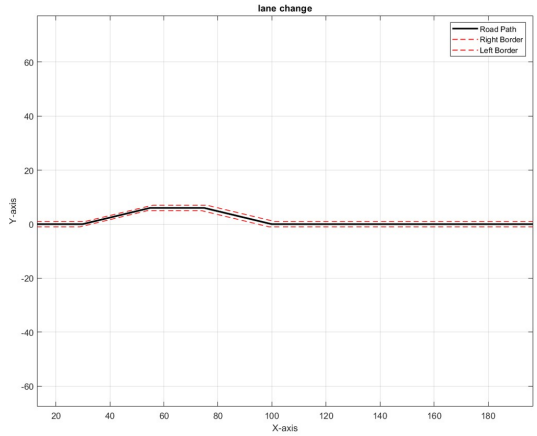
$$A = \begin{bmatrix} -2(cf + cr)/(mu_2) & -u_2 - 2(cflf - crlr)/(mu_2) \\ -2(cflf - crlr)/(Iu_2) & -2(cflf^2 + crlr^2)/(Iu_2) \end{bmatrix}$$
$$B = \begin{bmatrix} 2cf/m & 0 \\ 2cflf/I & 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Path planning implementation

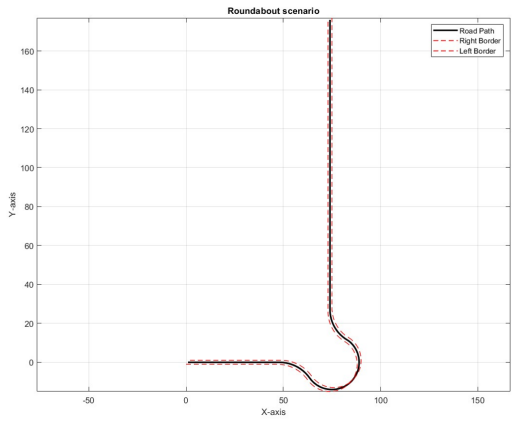
For path planning in Simulink, I first had to create a road (x, y, and yaw points in the path), for this purpose Driving Scenario Editor in MATLAB was used as described in 6.2.2, Figure 7.10 show some scenario designed for testing.



(a) Curved path



(b) Lane change path



(c) Roundabout path

Figure 7.10: Different Path Scenarios

After the road has been set, a path planning was carried out, by knowing the current global position of the vehicle, I determined the instantaneous path by calculating the distance between the vehicle and each point on the path and take the minimum, a MATLAB function to calculate the instantaneous path can be found here A.1.7. Also the algorithms for path planning for obstacles avoidance were described in section 4.3.

The logic followed is the following: Drive by following the path of the road, if obstacle detected and steering is possible to avoid the collision switch to one of the algorithm mentioned in 4.3, after avoiding the obstacle, return back to the original road path tracking. MATLAB function codes for obstacle avoidance path planning can be found here A.1.8

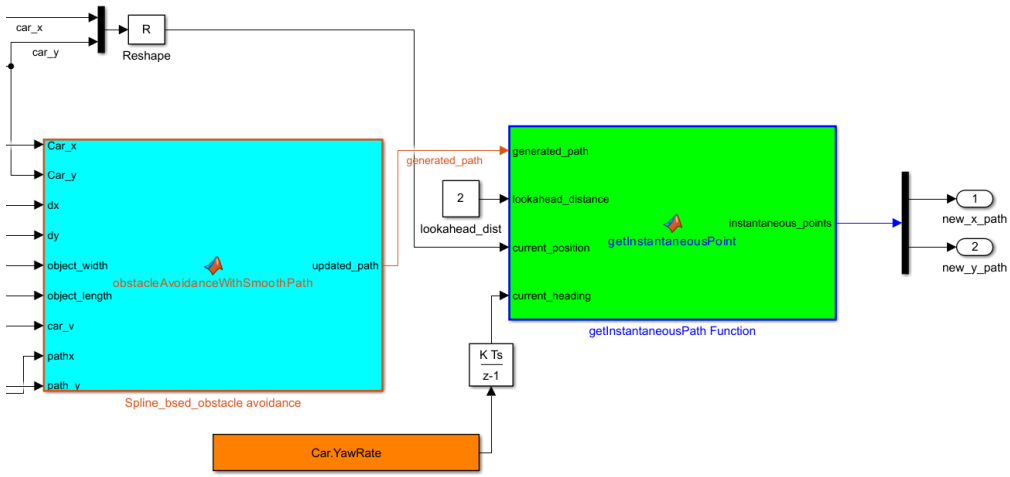


Figure 7.11: Simulink connection path planning for obstacle avoidance

Brake/Steering switching logic implementation

As stated, the desired speed is not only calculated based on the curvature and the minimum safe speed for cornering, but also based on the condition of the environment around the vehicle, the vehicle in front of it, and other parameters. This section describes the logic and the equations used to implement a collision avoidance algorithm that adjusts the desired speed according to the distance to the obstacle, the relative velocity, and the current speed. The code can be found here A.1.10.

The code defines a function f that returns the output speed v_s that avoids collision with the obstacle, based on the current speed v_c, the distance to the obstacle d_o, the relative velocity v_r, the desired speed v_d, and the minimum speed v_m. The function uses a nested function g that estimates the braking distance d_b required to stop the vehicle, given the speed v, the maximum deceleration a_m, and a safety factor s_f. The function uses the following equations:

$$d_b = \frac{v^2}{2a_m} s_f \qquad \text{Braking distance}$$

$$t_s = \frac{v_c}{a_m} \qquad \text{Time to stop}$$

$$a_r = -\frac{v_c}{t_s} \qquad \text{Required declaration}$$

$$d_p = v_c t_r \qquad \text{Projected distance the vehicle would cover during reaction time}$$

$$d_s = d_p + d_b \qquad \text{Safe following distance}$$

$$d_e = d_o - d_p \qquad \text{Effective distance to the obstacle}$$

$$v_c = v_c - v_r \qquad \text{Relative closing velocity}$$

$$v_s = \begin{cases} \max(v_m, v_c + a_r) & \text{if } d_e < d_s - 4 \text{ and } v_c > 0 \text{ and } d_o \neq 0 \\ 0 & \text{if } d_o > 0 \text{ and } d_o < 5 \text{ and } v_c > 0 \\ v_d & \text{otherwise} \end{cases}$$

7.1.3 Vehicle Visualization in Simulink

In this section, I am discussing the visualization of the vehicle in Simulink using the provided MATLAB function `plotVehicle`. This function is designed to plot the vehicle's position, orientation, and trajectory. The visualization includes the vehicle's body, rear and front wheels, as well as the path it follows.

MATLAB Code for Vehicle Visualization

To have better visualization and to closely monitor the vehicle performance I created a MATLAB that takes various parameters such as position, orientation, steering angle, and path information to create a 2D plot of the vehicle in Simulink, so I am not relaying only on the signals visualized though the scope. The MATLAB function, `plotVehicle` is attached in the appendices, refer to A.1.1.

Description of Visualization Elements

The function incorporates the following elements into the visualization:

Visualization Output

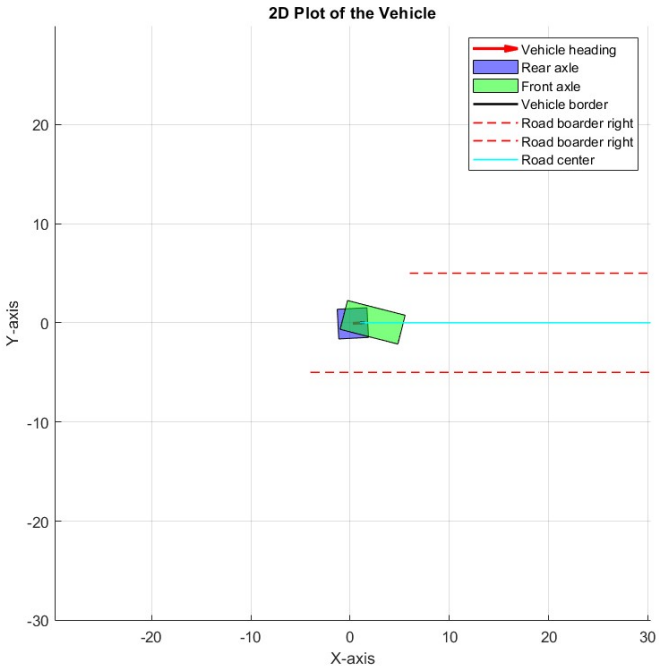


Figure 7.12: Vehicle Visualization in Simulink

The resulting plot provides a clear representation of the vehicle's current state, trajectory, and path borders.

- The vehicle's body, represented as an arrow indicating the direction of movement.
- Rear and front wheels displayed as colored patches.
- The vehicle's path, including the full path, left and right borders, plotted on the graph.

Adjustments to the axis, labels, and titles are made to ensure a clear representation of the vehicle's dynamics.

7.2 Integrating Simulink models with CarMaker

CarMaker software provides a Real-Time Simulation (SIL, Software-in-the-Loop) environment, allowing seamless integration with Simulink models for comprehensive

vehicle testing and validation. The software also features an intuitive Graphical User Interface (GUI) that simplifies the integration process. In figure 7.13, the basic home GUI can be seen, as mentioned in 5.1, it is possible to start building the car from scratch inside it, or just made the changes you need on the vehicle data-set from one of the vehicle examples provided in CarMaker, the software provides a wide variety of example vehicles, refer to 5.1 to see the vehicle I chose and made my changes.

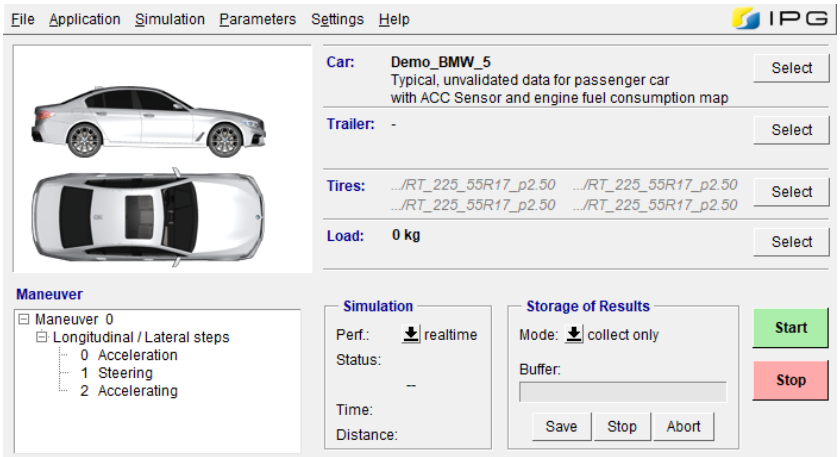


Figure 7.13: CarMaker home page

With these fascinating features, the CarMaker also provides a possibility to use Simulink based models, and connect them to it. In section 7.2.2 this process will be described.

7.2.1 CarMaker environment setup

CarMaker provides an access for many scenarios, roads, traffic objects, and vehicles. The vehicle chosen is described here 5.1. For the road scenario designing, it can be done from the maneuver tab as shown in figure 7.13. To test the steering controller effectively I chose a path with many curves, so the steering behaviour can be seen clearly. In addition to that traffic objects are added and manipulated to test the different scenarios for collision avoidance, figure 7.14 show the scenario editing window.

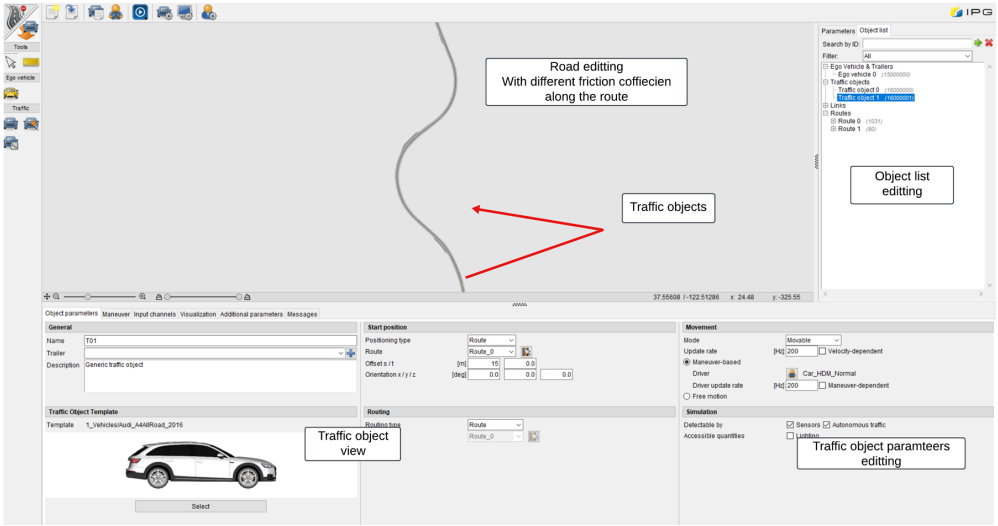


Figure 7.14: CarMaker scenario editing

Sensors Mounting

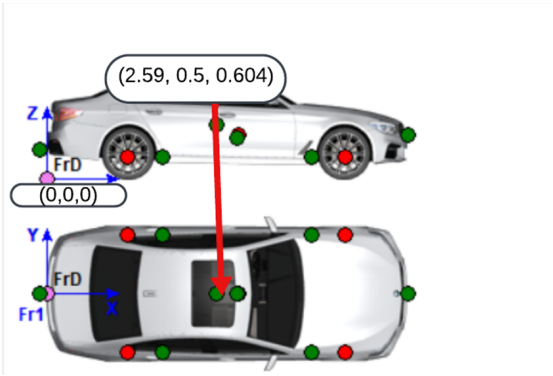


Figure 7.15: Vehicle frame reference points

In addition to the vehicle's basic sensors, several additional sensors have been strategically mounted on the car to enhance its perception capabilities. The vehicle frame reference points are shown in figure 7.15and based on those points I integrated the sensors in table (7.1) into the vehicle.

Sensor Type	Position (x)	Position (y)	Position (z)
Slip Angle Sensor	2.596	0.0	0.604
Radar Sensor	4.7	-0.5	0.3
Object Sensor	4.9	0.0	0.5
Road Sensor	4.9	0.0	0.5

Table 7.1: Sensor Positions on the Vehicle

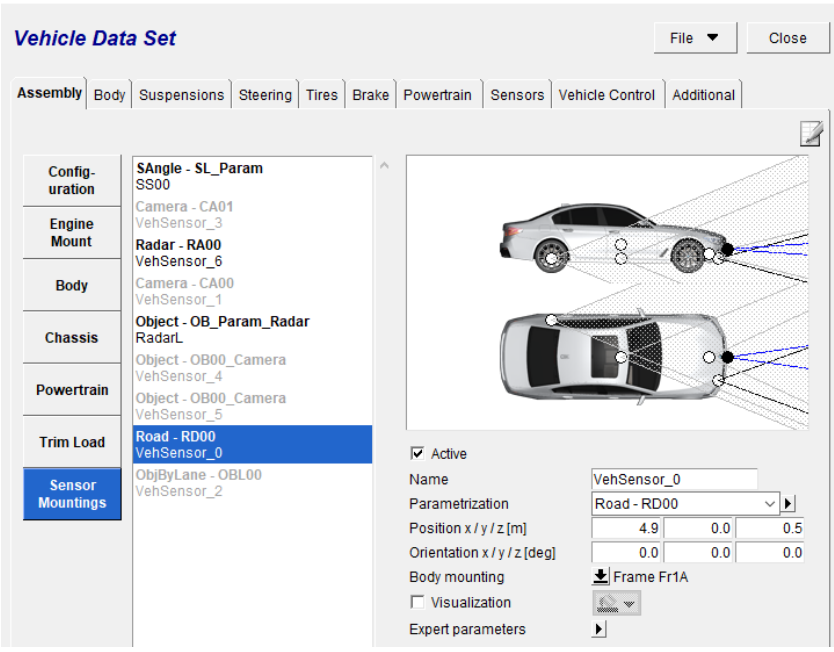


Figure 7.16: Sensors position setting CarMaker

These sensors play a crucial role in the vehicle's perception system, providing essential data for various applications such as slip angle estimation, object detection, radar sensing, and maintaining proper alignment with the road.

7.2.2 Connecting Simulink model to CarMaker

Connecting Simulink model to CarMaker is very easy, there is a MATLAB function that can be run or included in the MATLAB setup function, so it get run every-time a MATLAB is open, after creating the project this function can be found in the project directory ($Project_name/src_cm4sl/cmenv.m$), this function setup the CarMaker library into Simulink model, after that the generic model can be opened (see 7.17) and

inside CarMaker block the integration of Simulink controller can be done there as seen in figure 7.18

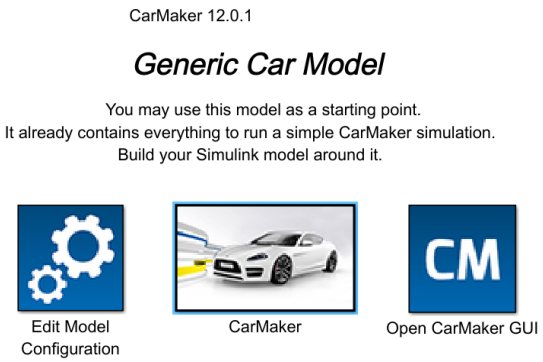


Figure 7.17: Generic model CarMaker

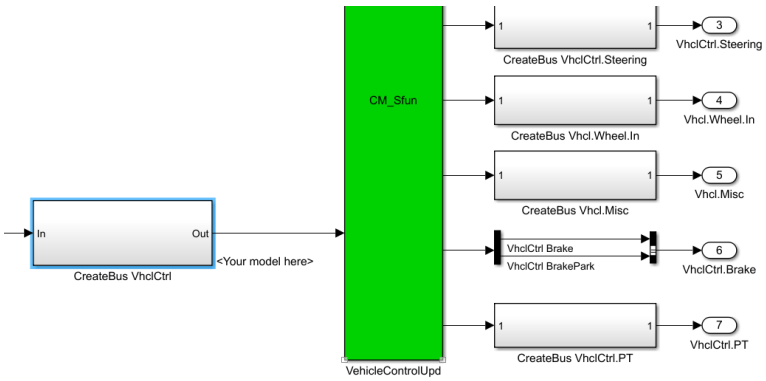


Figure 7.18: Integrating Simulink model with CarMaker

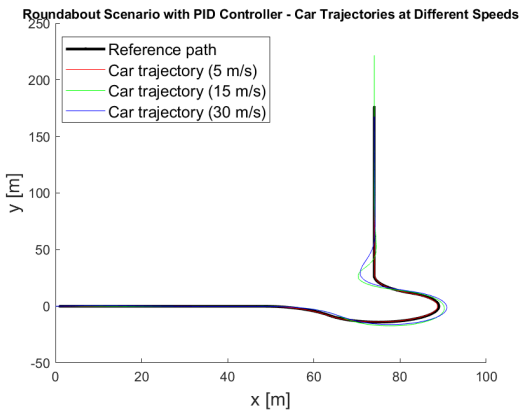
8. PRESENTATION OF RESULTS AND TESTING

8.1 MATLAB/Simulink Model Results

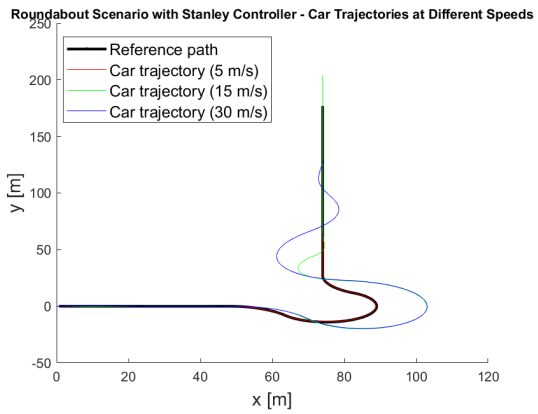
The model was tested first in Simulink in three different scenarios and three different speeds to evaluate controllers efficiency.

8.1.1 Path Tracking Results

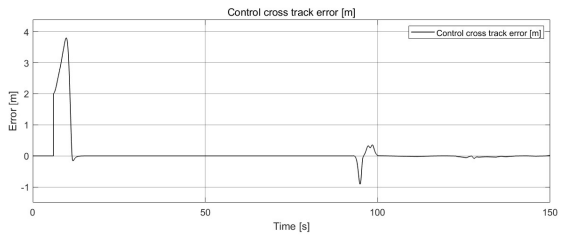
Applying the algorithm described in 4, and following the implementation described in 7, the final model was tested in Simulink, with the following results shown in figures (8.1 to 8.8).



(a) Car Trajectories PID Controller



(b) Car Trajectories Stanley Controller



(c) Trajectory errors Stanley controllers while avoiding two obstacles

Figure 8.1: Car Trajectories with Speed Variation

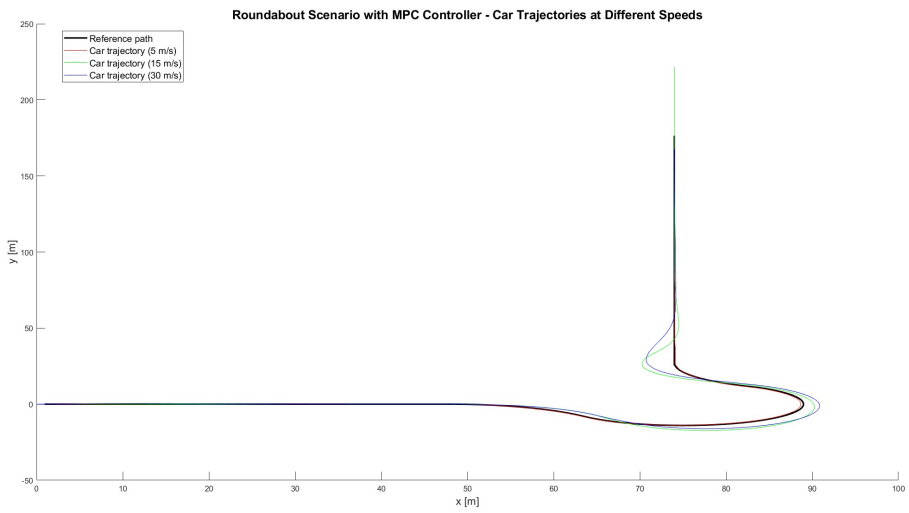


Figure 8.2: Vehicle trajectory response for the MPC controller.

Figures 8.2 and 8.1, show the results of path tracking, the there controllers show a very good result for tracking, however PID and Adaptive MPC provided better results for path tracking however the steering angle in the PID controller is oscillating a lot, the oscillation was reduced by adding a varying low pass filter, the steering signals of the PID and Stanley are shown in figure 8.5.

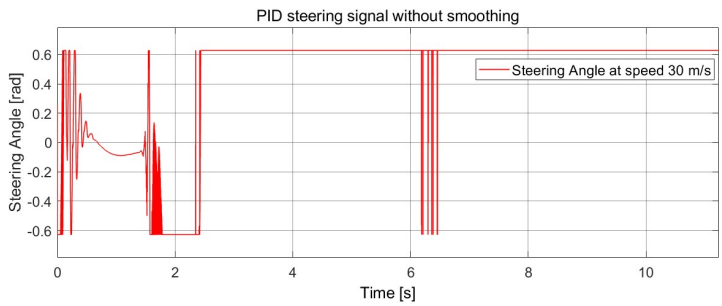


Figure 8.3: PID steering Angle without smoothing

After adding a varying low pass filter with a cut-off frequency of 25 rad/s the signal of the PID steering signal is smoothed, however a very low oscillation still exist but this so small that does not effect the performance of the controller. The Stanley controller showed a very smooth signal but the control cross-track error is bigger than the PID signal.

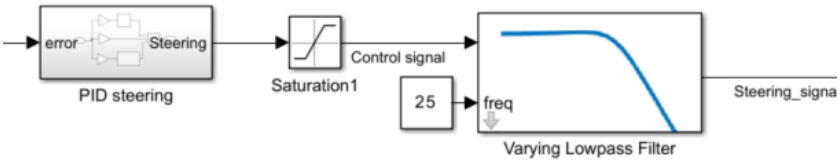
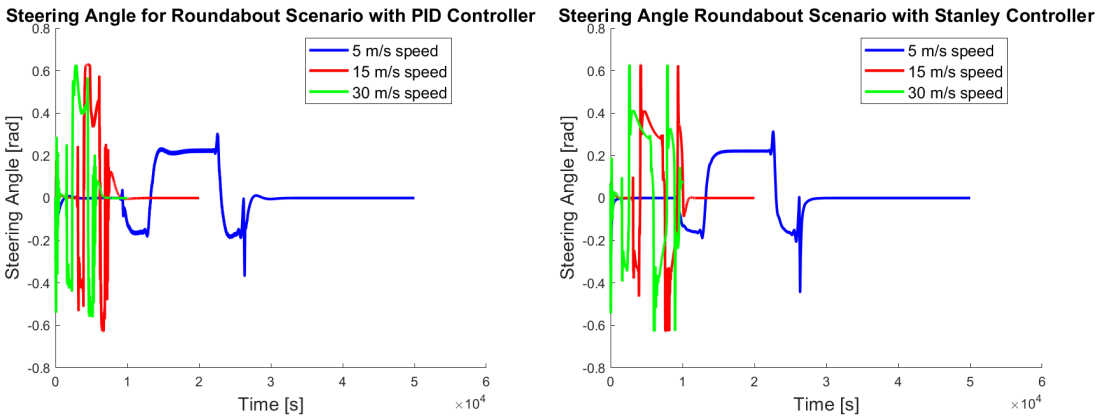


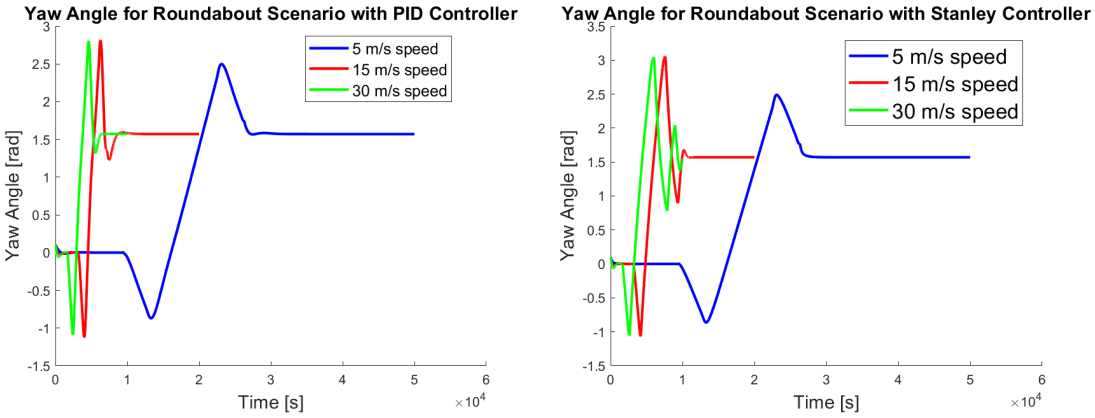
Figure 8.4: Smoothing the PID steering signal



(a) Steering Angle PID Controller (b) Steering Angle Stanley Controller

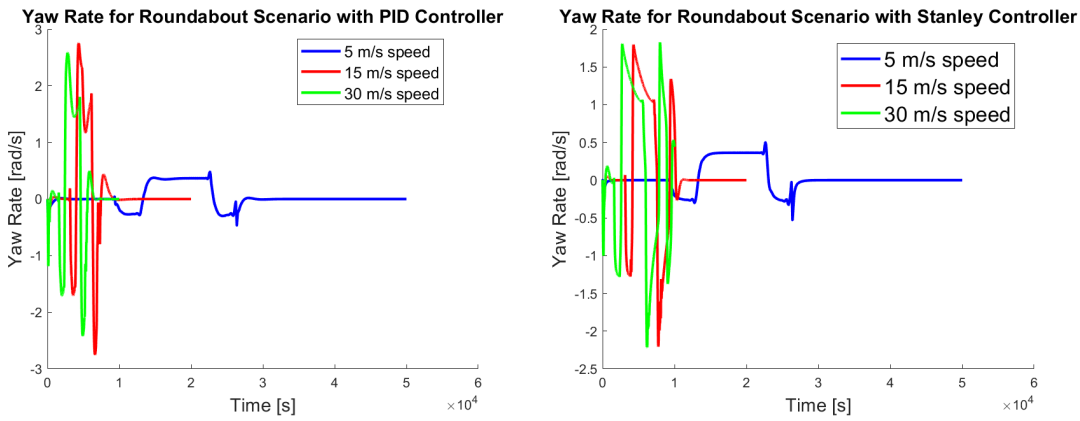
Figure 8.5: Steering Angle Results

I plotted the yaw angle along with the yaw rate to see the vehicle stability during the steering both the smoothed PID and the Stanley showed a reasonable yaw rate, indicating a stable behaviour as can be seen in figure 8.6 and figure 8.7.



(a) Yaw Angle PID Controller (b) Yaw Angle Stanley Controller

Figure 8.6: Yaw Angle Results

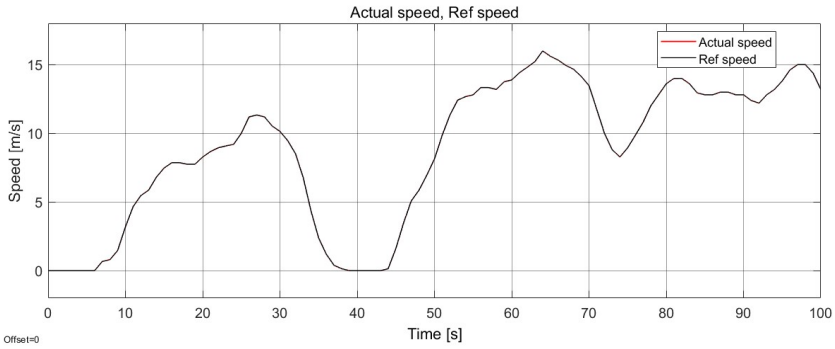


(a) Yaw Rate PID Controller (b) Yaw Rate Stanley Controller

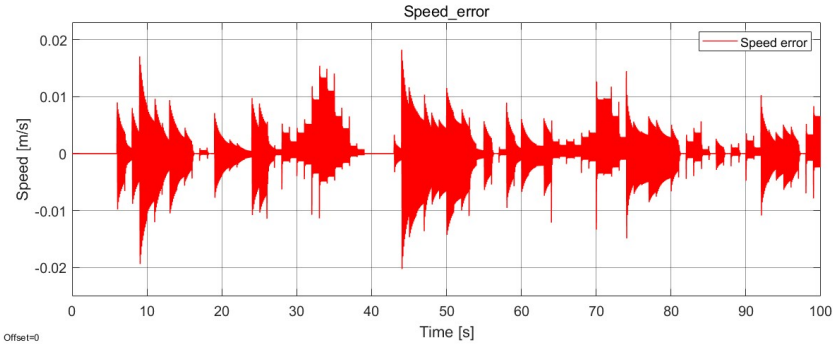
Figure 8.7: Yaw Rate Results

8.1.2 Longitudinal results

Both the Model predictive controller described in 4 and the PID controller described in 3 showed good results with very low error presented in figure 8.8b, the speed profile signal is the 'Motorway maximum speed of 150 km/h', the file of the speed profile is provided in the Simulink environment. The two models will be tested and validated again in CarMaker SW environment.



(a) Speed tracking for speed ref (Drive cycle source Motorway maximum speed of 150 km/h)



(b) Speed controller error

Figure 8.8: PID speed controller results

8.2 CarMaker testing and results

In this section, I present the results of testing the controllers on a vehicle model using Simulink and CarMaker. I use Simulink to implement the different controllers and interface it with CarMaker using S-functions and MATLAB API functions.

I test the performance of the controller on a curved road with different reference speeds and compare the results. I evaluate the controller using the following metrics:

- **Car Trajectories:** The deviation of the actual car position from the reference path. The reference path is either the path on the road or the free collision path created by one of the algorithms mentioned in Section 4.3. The implementation can be seen in Figure 8.9.

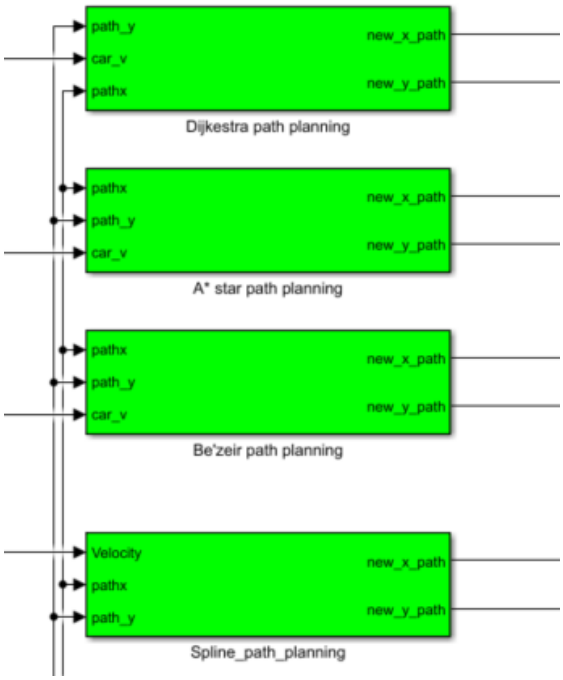


Figure 8.9: Path planning algorithm Simulink implementation

The logic followed to avoid the obstacles is described in Section 7.1.2, while the MATLAB functions for path generation can be found in the appendix.

- **Brake Commands:** The brake pedal position commanded by the controller or the virtual driver, by this metric we can determine the response of the vehicle to emergency braking where avoiding the the collision by steering is not a solution based on the logic explained earlier.

- **Yaw Angle and Yaw Rate:** The orientation and the angular velocity of the car around the vertical axis. Yaw rate and yaw angle is essential for assessing the stability of the vehicle, trajectory, and overall dynamic behavior of the vehicle.

- **Slip Angle:** The angle between the direction of the wheel and the direction of the car's motion. By having this as a metric I am assessing tire-road interaction, grip, steering, cornering performance, and overall vehicle stability.

- **Wheel Torques:** The torques applied to the four wheels by the brake system, so that I can have information about the wheel brake torques, the synchronization of brake signal applied to each tire, preventing instability or vehicle skidding.
- **Speed Tracking:** The difference between the actual speed and the reference speed, to asses the effectiveness of the longitudinal controllers.
- **Steering Angle:** The angle of the front wheels relative to the car's longitudinal axis, which will provide us information about the controller signal behaviour.
- **Throttle:** The throttle pedal position commanded by the controller or the driver, this is also a valuable metric for the longitudinal controller.
- **Trajectory Errors:** The lateral and longitudinal errors between the actual car position and the reference path.

The figures below illustrate the simulation results for different reference speeds. The specific speed settings are indicated in the legend of each corresponding figure.

The simulation scenario aimed to assess the model's performance under various conditions. A curved path with multiple turns was chosen to evaluate the path tracking algorithm. Traffic objects were strategically placed at the beginning and end of the path to introduce challenges. The scenario unfolds as follows:

- The initial section involves negotiating curves to evaluate the path tracking algorithm. - Traffic objects are introduced at the beginning, simulating a sudden brake by the first object. The vehicle's environment and distance allow for appropriate steering response. - Subsequent traffic objects can be navigated smoothly. - Upon reaching the end of the path, a vehicle in front applies brakes, creating a potential collision scenario. - The collision avoidance system responds by initiating braking, considering the vehicle's speed, distance, and surrounding environment.

These simulation scenarios comprehensively test the model's capabilities in path tracking, obstacle avoidance, and collision mitigation.

8.2.1 Lateral controllers evaluation(testing results)

PID results

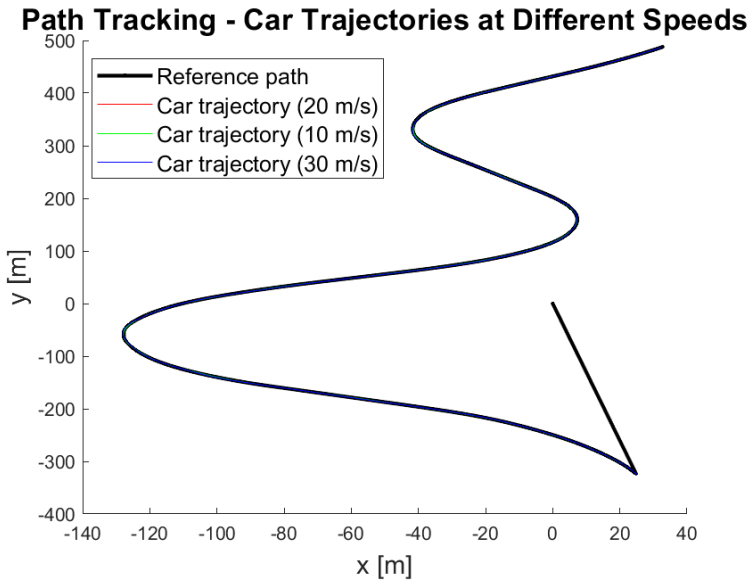


Figure 8.10: Car Trajectories at Different Speeds with PID steering

The smoothed PID controller demonstrated precise results in path tracking, successfully navigating the specified trajectory. However, a potential failure occurred in collision avoidance, as depicted in Figure 8.11.

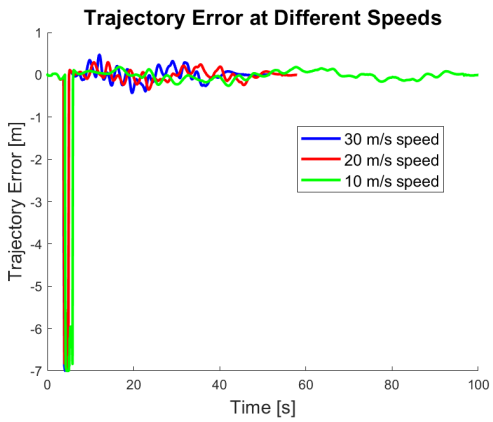


Figure 8.11: Trajectory Errors at Different Speeds

In this specific scenario, two obstacles were positioned closely together. The controller effectively avoided the first obstacle, as indicated by the path error in the figure (approximately 6 meters). This error represents the deviation from the original path to circumvent the obstacle and is the sum of the vehicle's path error and the newly generated path for collision avoidance.

Regrettably, the controller encountered challenges in avoiding the second obstacle, especially at higher speeds. The maximum error observed at a speed of 30 m/s was 0.8 meters. The steering signal behaivour can be seen in figure 8.12

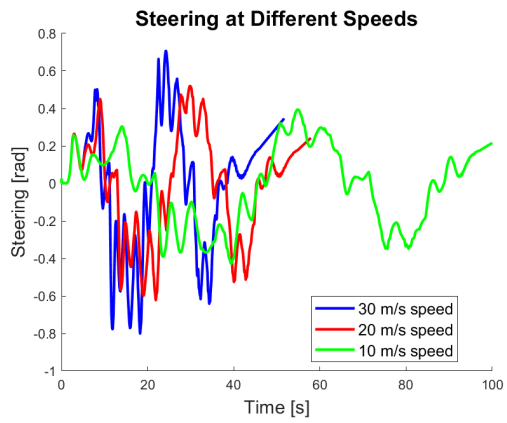


Figure 8.12: PID Steering at Different Speeds

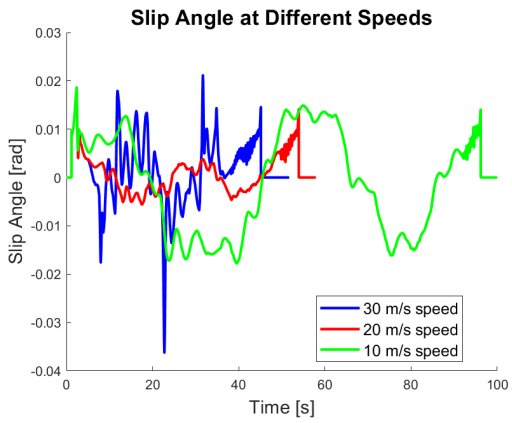


Figure 8.13: Slip Angle at Different Speeds

In Figure 8.13, the slip angle is illustrated. The PID controller exhibited minimal slip angles ranging from 0.03 to -0.05 radians, equivalent to around 2.3 degrees as the max-

imum value. These small values can be considered negligible, indicating high vehicle stability during both braking and cornering maneuvers.

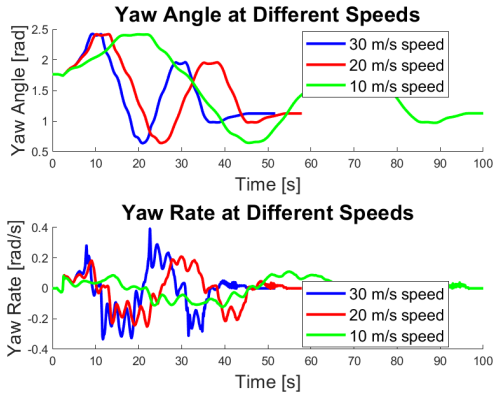


Figure 8.14: Yaw Angle and Yaw Rate at Different Speeds

The yaw rate signal shows a maximum of 0.4 rad/s around 22 deg/s at high speed, but the change in the yaw rate is big, which shows the drawback of the PID steering system.

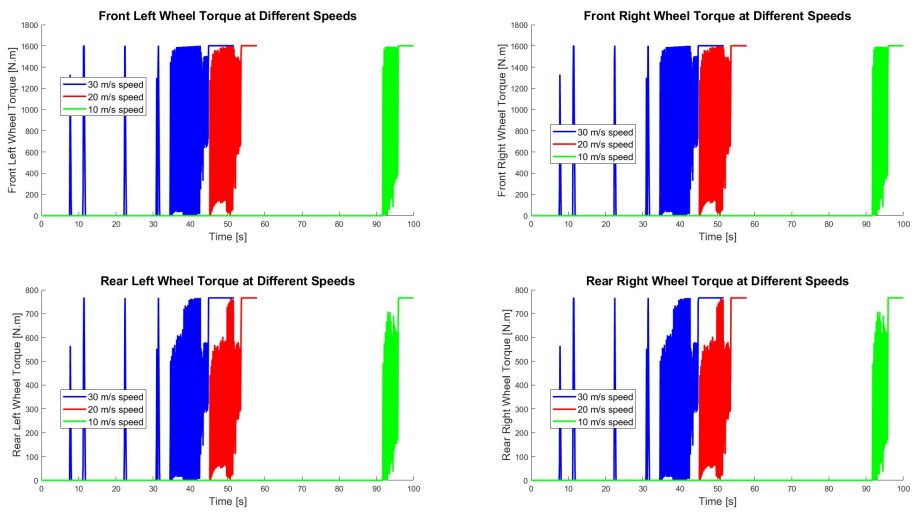


Figure 8.15: Wheel Torques at Different Speeds

For the brake signal at the end of the scenario can be seen in figure 8.16. As the speeds are different the vehicle reaches the end of the path at different times which can

be seen in the figure also. At high speeds the oscillation in the brake signal is high (this oscillation can be further smoothed by a varying low pass filter as was done for the PID steering signal) the oscillation happens for few seconds before the brake is set fully to the maximum.

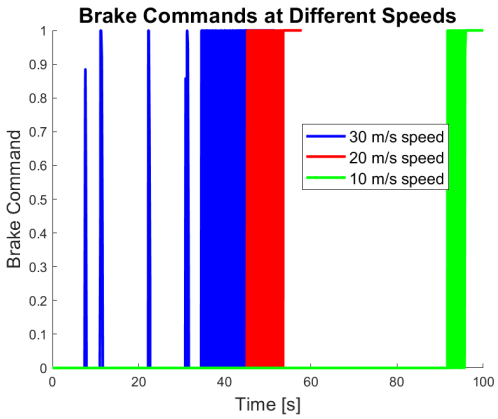


Figure 8.16: Brake Commands at Different Speeds

Similar to the brake signal command oscillation can be found at the end of the scenario, this oscillation is due to sudden braking from high speeds, it is smoothed by smoothing the brake signal with varying low-pass filtering. The figure 8.15 shows a synchronized response for braking in the four wheels.

Speed behaviour

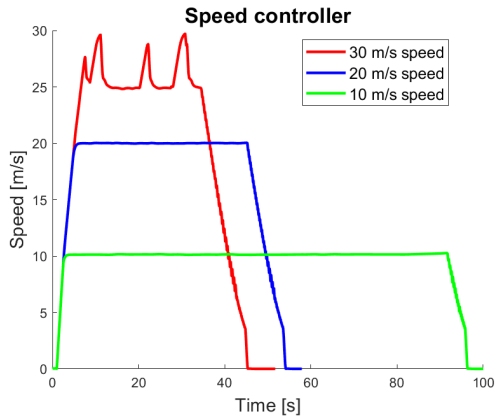


Figure 8.17: MPC Speed Tracking

In Figures 8.17 and 8.18, the behavior of speed and throttle during the scenario is depicted. The reduction in speed when the desired speed is 30 m/s is not solely attributable to the controller; rather, it is influenced by the curvature of the path and the speed of the traffic object in front of the vehicle. The desired speed is adjusted based on these factors in the implemented logic.

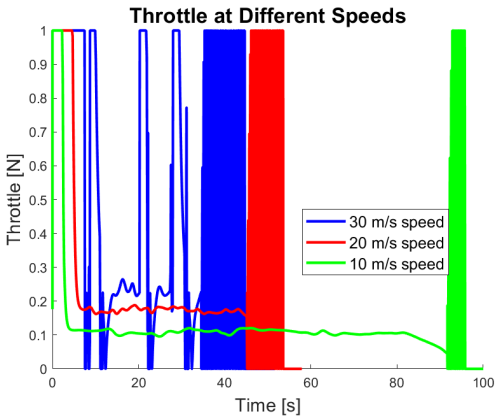


Figure 8.18: Throttle at Different Speeds

Stanley controller

Following the method described in Section 7.1.2, the Stanley controller was implemented and tested. However, the results indicate a failure in path tracking at high speeds, as illustrated in Figure 8.19 where, at a speed of 25 m/s (approximately 90 km/h), the car deviates from the trajectory (as indicated by the blue line in the figure).

At a lower speed of 20 m/s (about 72 km/h), the controller shows satisfactory results, although there are small oscillations in the steering signal, and the vehicle exhibits instability on low curved paths.

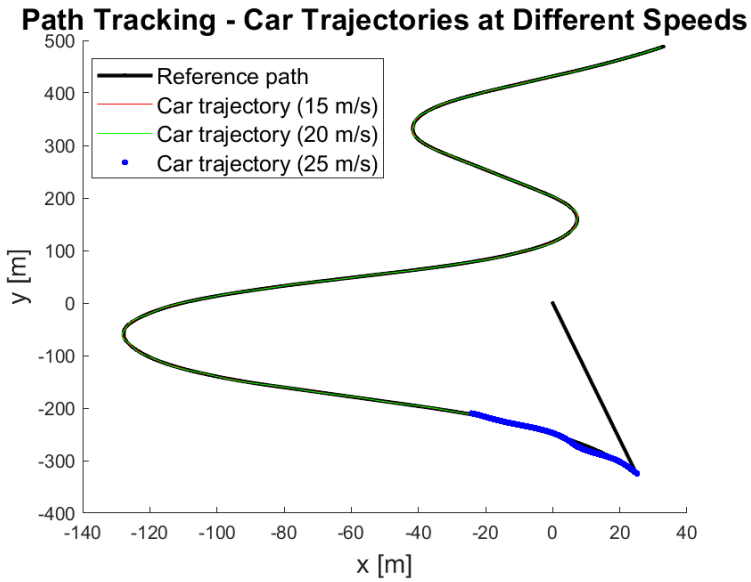


Figure 8.19: Car Trajectories at Different Speeds with the Stanley controller

The oscillation in the cross-track error at a speed of 72 km/h is depicted in Figure 8.20. At lower speeds, the behavior is nearly perfect, with very small oscillations that can be neglected (around zero). Furthermore, at speeds of 72 km/h and 54 km/h, the vehicle steers around the traffic object effectively with no collision.

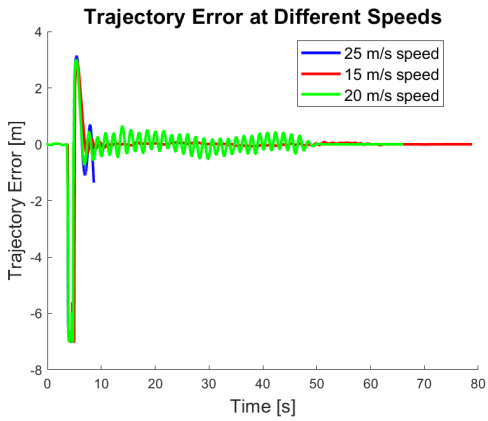
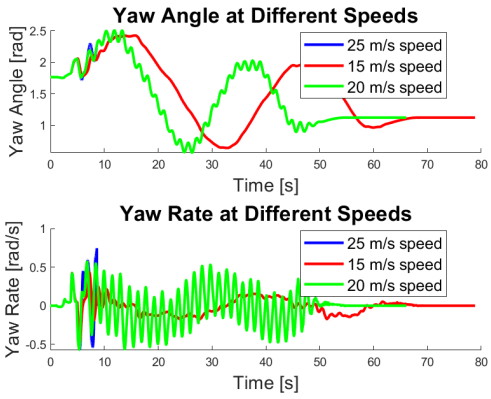
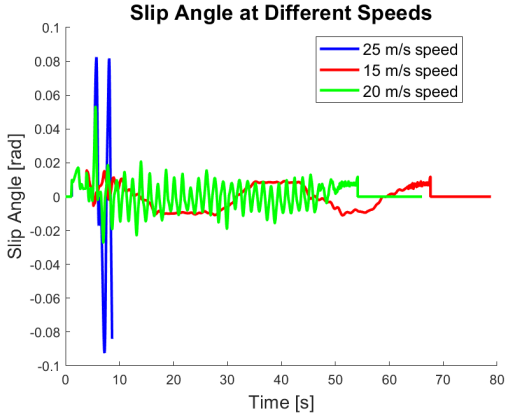


Figure 8.20: Trajectory Errors at Different Speeds with Stanley controller

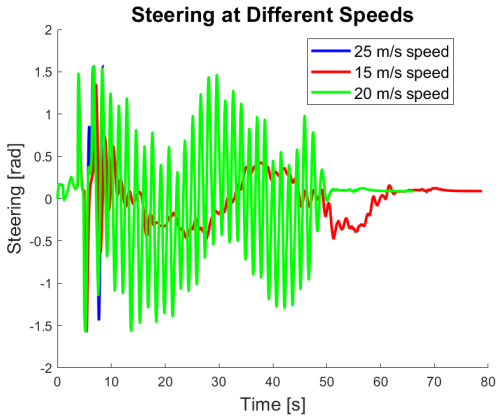
Figure 8.21, shows the yaw rate, yaw angle, slip angle, and steering angle at those different speeds.



(a) Yaw Angle and Yaw Rate



(b) Slip Angle



(c) Steering at Different Speeds

Figure 8.21: Yaw, slip Angles, yaw rate and Steering with Stanley controller

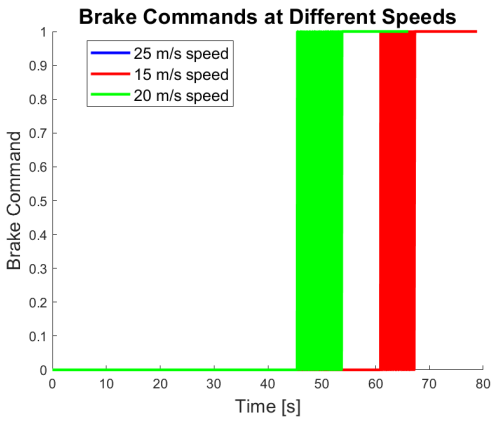


Figure 8.22: Brake Commands at Different Speeds

The brake signal is the same as the PID, since the signal comes from either a PID controller or MPC controller as described in chapter 4 and is not effected by the type of the lateral controller.

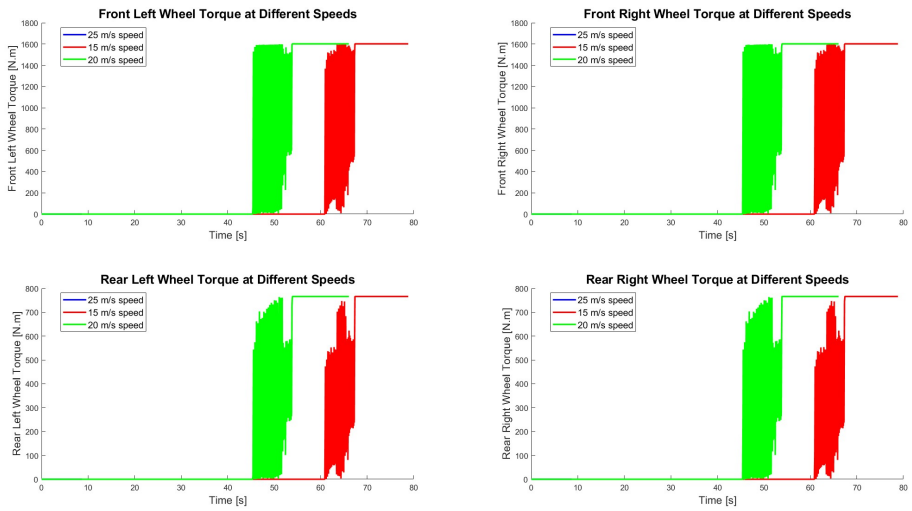
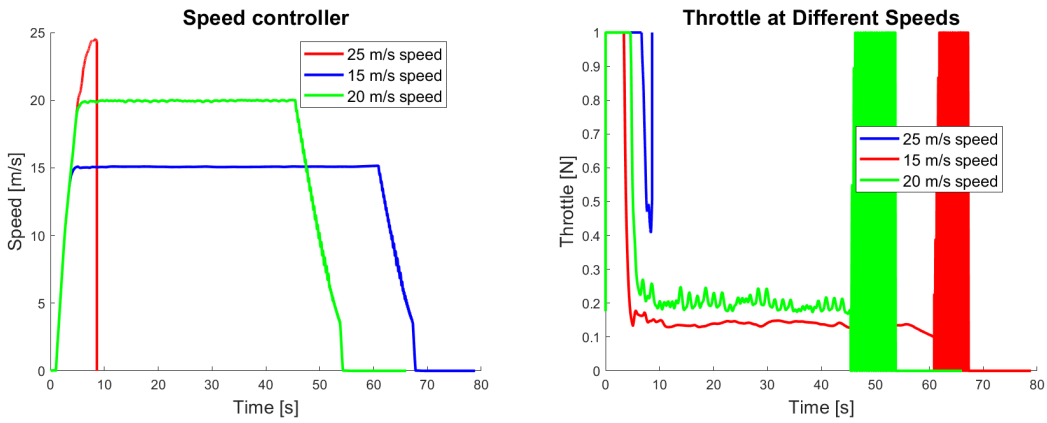


Figure 8.23: Wheel Torques at Different Speeds



(a) MPC Speed Tracking (b) Throttle at Different Speeds

Figure 8.24: MPC Speed Tracking and Throttle at Different Speeds with the Stanley controller

The red line in figure 8.24b shows a drop as the lateral controller failed to follow the path and the vehicle went off the road.

Stanley controller new version

As stated in 8.2.1, the algorithm used there is not sufficient for high speeds, so I made a modification based on this paper [13], as follows The final control signal is calculated as follows,

$$\delta(t) = (\psi(t) - \psi_{ss}(t)) + \arctan\left(\frac{k \cdot e(t)}{k_{\text{soft}} + v(t)}\right) \tag{8.1}$$

$$+ k_{d,\text{yaw}}(r_{\text{traj}} - r_{\text{meas}}) + k_{d,\text{stear}}(\delta_{\text{meas}}(i) - \delta_{\text{meas}}(i+1)) \tag{8.2}$$

I calculated the path yaw rate as a product of the path curvature κ and vehicle speed.

$$r_{traj} = v \cdot \kappa_{path} \tag{8.3}$$

the paper suggests using dynamic equations for steady state error as seen in 8.4.

$$\psi_{\text{ss}} = \frac{mv(t)r_{\text{traj}}(t)}{C_y(1 + b/a)} = k_{\text{ag}}v(t)r_{\text{traj}}(t) \tag{8.4}$$

In the Stanley controller, a set point is defined as $k_{\text{ag}} = \frac{m}{C_y(1+\frac{b}{a})}$. During simulations, using this in the controller did not impact stability but effectively turned the vehicle into turns to nullify cross-track error. However, this approach did not yield satisfactory results. The first term was subsequently replaced by the deviation angle with an amplifying factor ,where $K_{deviation}$ = 9.015, a tun-able parameter fine-tuned using MATLAB's parameter estimator.(Deviation angle (heading error) calculation can be found at 7.1.2)

To prevent the gain term $v(k_t)$ from becoming excessively sensitive to noise on $e(t)$ during low-speed driving, a tuned gain, k_{soft}, is introduced to the denominator. This permits control to be arbitrarily soft at low speeds, and in experiments, $k_{\text{soft}} = -0.57064$ was found to be appropriate. The final values of the tun-able parameters are

Parameter	**Value**
k_{dyawrate}	0.23163
k_{stanley}	28.491
k_{steering}	5.2446
k_{soft}	-0.57064

Table 8.1: Tunable Parameters in the Updated Stanley Controller

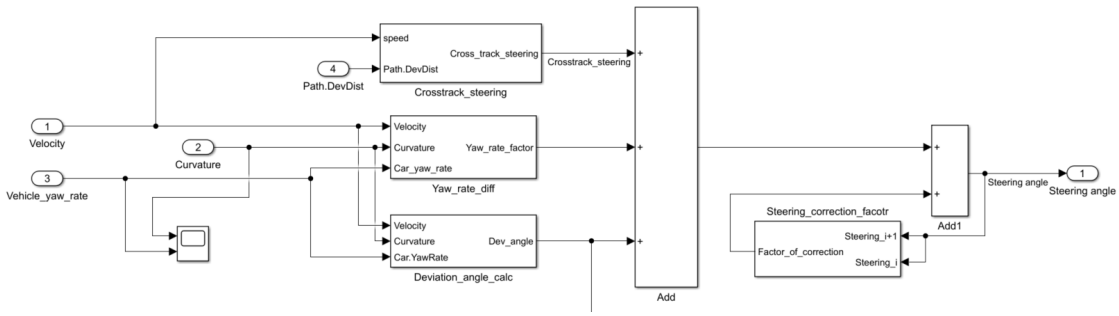
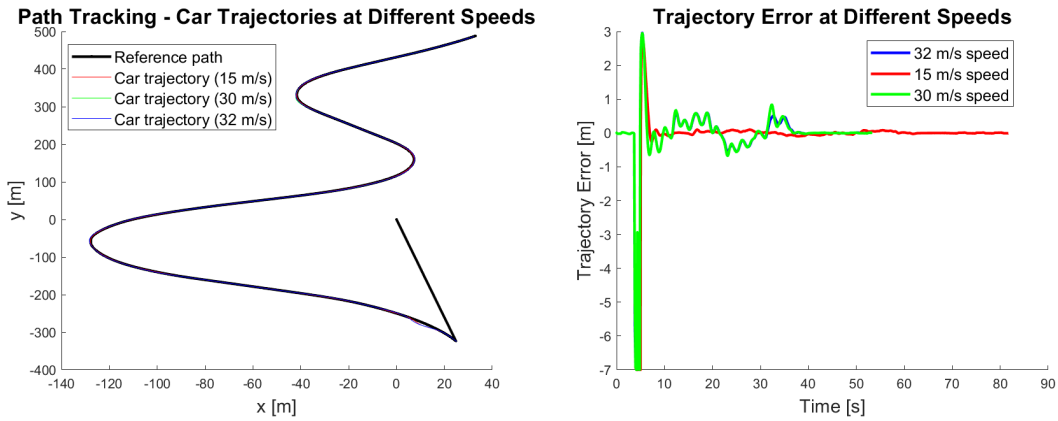


Figure 8.25: updated Stanley controller

The last term of Equation 8.1 represents the steering correction factor, calculated as the subtraction of two states in the steering signal. In figure 8.25, a Simulink implementation for the updated Stanley algorithm can be seen there.

Results

The enhanced Stanley steering controller demonstrates exceptional precision in path tracking and effective collision avoidance through steering maneuvers. It adeptly navigates the track at a speed of 115 km/h, exhibiting minimal deviation and negligible oscillations. The controller exhibits robust performance across a broad speed range, ensuring reliable operation at both high and low speeds.

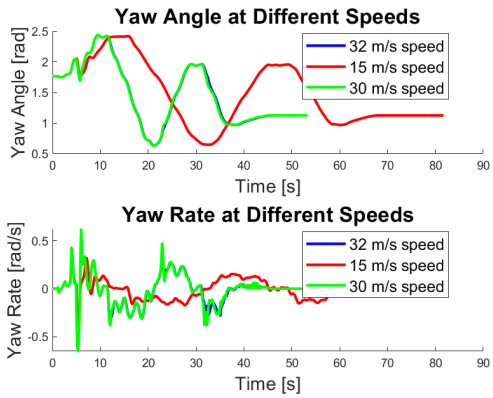


(a) Car Trajectories at Different Speeds (b) Trajectory Errors at Different Speeds

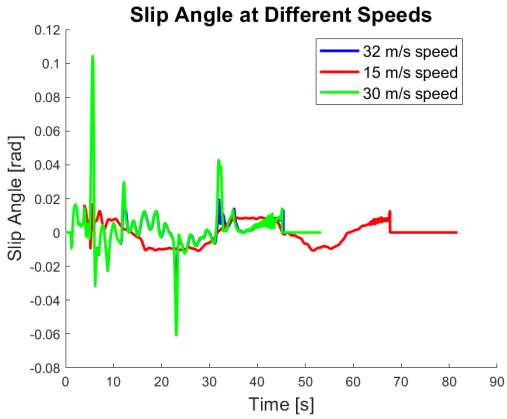
Figure 8.26: Stanley Steering Controller Performance

In Figure 8.26a, the controller consistently adheres to the designated path across

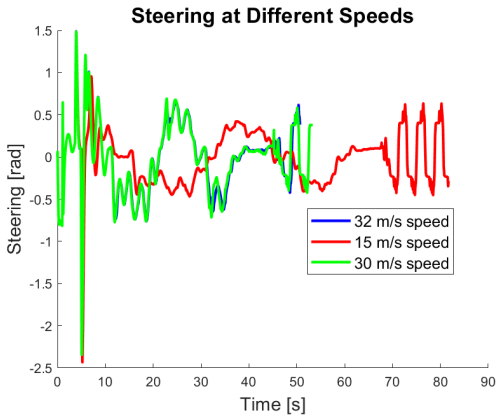
varying speeds. The trajectory error, as illustrated in Figure 8.26b, remains impressively low, with a maximum error of 0.55 m recorded during continuous path following at 115 km/h, even through challenging curved corners. Notably, the errors of -6 m and 3 m in Figure 8.26b represent successful avoidance of two traffic objects.



(a) Yaw Angle and Yaw Rate



(b) Slip Angle at Different Speeds



(c) Steering wheel [rad]

Figure 8.27: Updated Stanley Controller Results: Yaw Angle, Slip Angle, and Steering Angle

The figure 8.27, shows the yaw angle, yaw rate, slip angles, and the steering wheel in rad. The results indicate a stable response from the vehicle for the steering either during the vehicle path following or traffic objects avoidance.

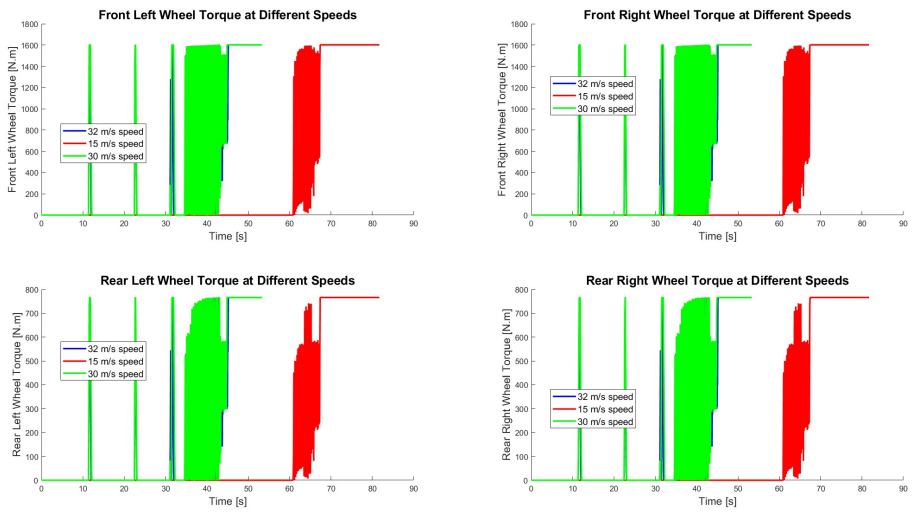
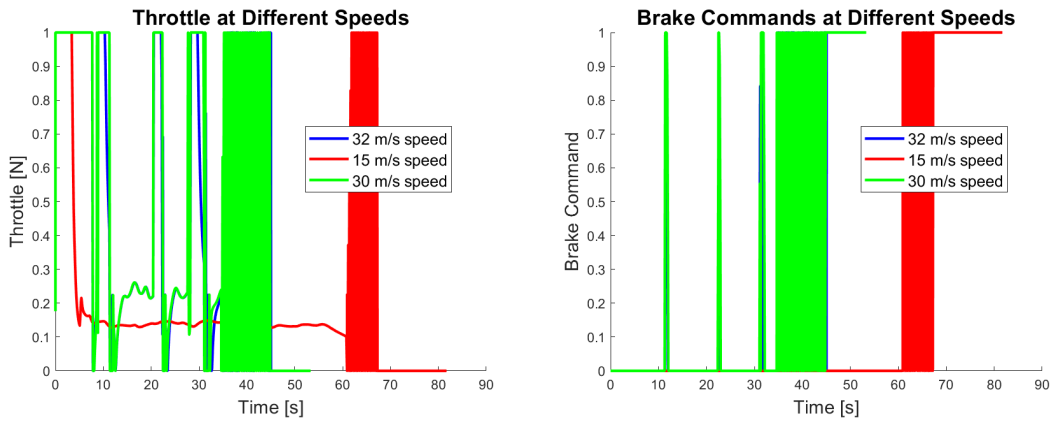


Figure 8.28: Wheel Torques at Different Speeds



(a) Throttle at Different Speeds (b) Brake Commands at Different Speeds

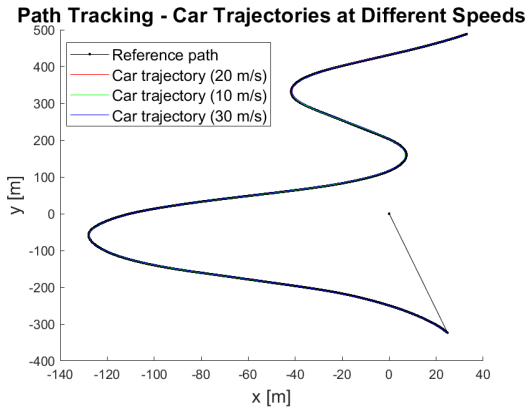
Figure 8.29: Throttle and Brake Commands with Stanley Steering Controller

From figure 8.29 and figure 8.28, we can see changing the lateral control did not effect the longitudinal controller, and the vehicle at the end of the scenario is able to brake fully.

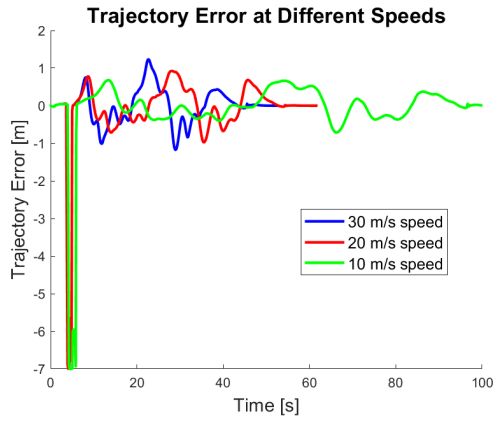
A-MPC results

Utilizing the algorithm outlined in 1 and implemented as detailed in the 7.1.2 section, the Adaptive Model Predictive Controller (AMPC) demonstrates effective collision avoidance while efficiently following the designated path. However, at elevated speeds, the cross-track error tends to increase, as depicted in Figure 8.30a and Figure 8.30b.

Despite the noticeable rise in cross-track error, with a maximum error of 1.4 m at a speed of 30 m/s during path following, the controller exhibits resilience by successfully returning to the road. This robust performance showcases the controller's ability to navigate challenging scenarios and recover from deviations.



(a) Car Trajectories at Different Speeds



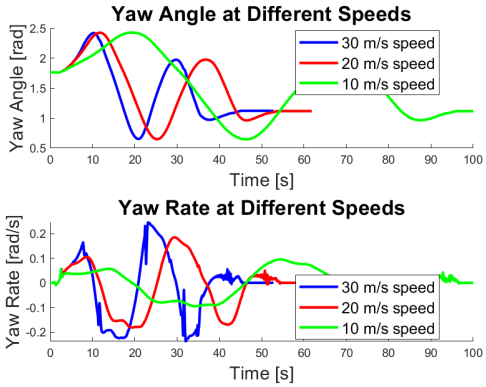
(b) Adaptive model predictive controller (A-MPC) controller Trajectory Errors

Figure 8.30: Car Trajectories and Trajectory Errors with A-MPC Controller

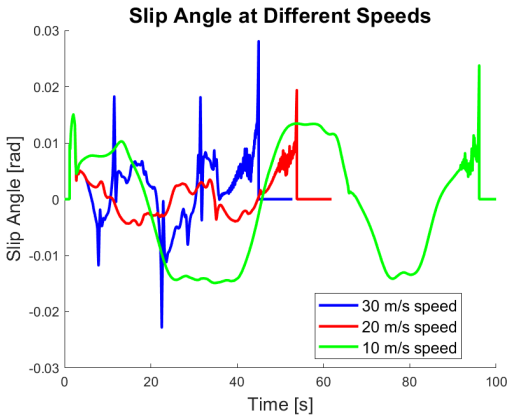
The dynamics of the Adaptive Model Predictive Controller (A-MPC) can be comprehensively assessed through the examination of key variables, including the steering angle, yaw angle, yaw rate, and slip angle, as illustrated in Figure 8.31. This depiction offers insights into the controller's overall stability and responsiveness across various driving conditions.

In the presented figure, the system's behavior exhibits notable stability, with discernible responses in all monitored angles. While slight oscillations are observed in both the slip angle and yaw rate, it is essential to note that these oscillations remain within negligible ranges. Their impact on the overall system is minimal and can be effectively disregarded, attesting to the robust and consistent performance of the A-MPC

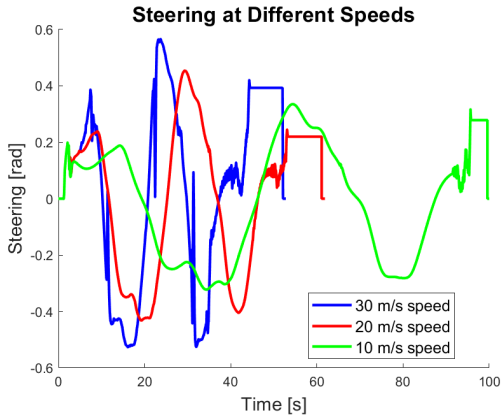
in maintaining vehicle dynamics.



(a) Yaw Angle and Yaw Rate



(b) Slip Angle at Different Speeds



(c) Steering at Different Speeds

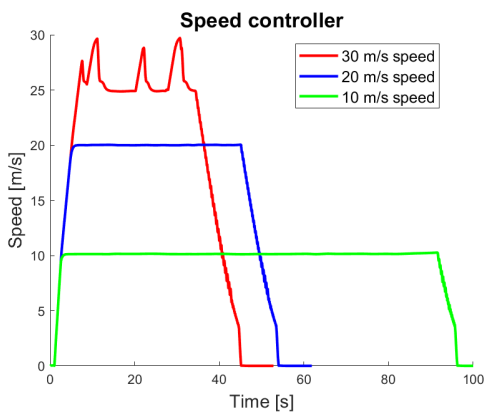
Figure 8.31: Yaw Angle, Slip Angle, and Steering Angle with A-MPC Controller

The presented set of figures provides a comprehensive overview of the dynamic interaction between lateral and longitudinal control in the vehicle system, particularly emphasizing the impact on brake commands, throttle responses, and wheel torques. This exploration is crucial for understanding how changes in lateral control, as exemplified by the MPC algorithm, influence the vehicle's overall behavior without directly affecting the longitudinal control elements.

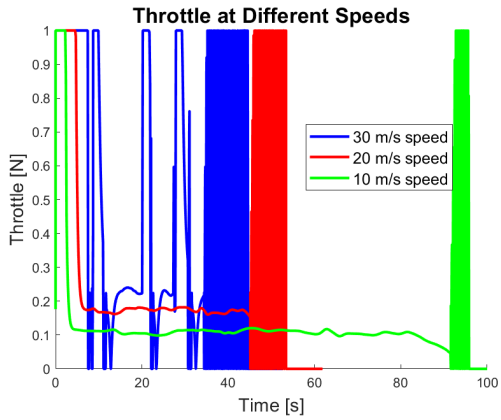
In Figures 8.32a, 8.32b, and 8.32c, the effects of the MPC lateral control strategy on speed tracking, throttle input, and brake commands are clearly delineated. Notably, variations in lateral control, such as steering adjustments, induce changes in the vehi-

cle's trajectory and lateral dynamics. However, it is essential to recognize that these alterations in lateral behavior do not directly translate to modifications in longitudinal control parameters, namely braking and throttle commands.

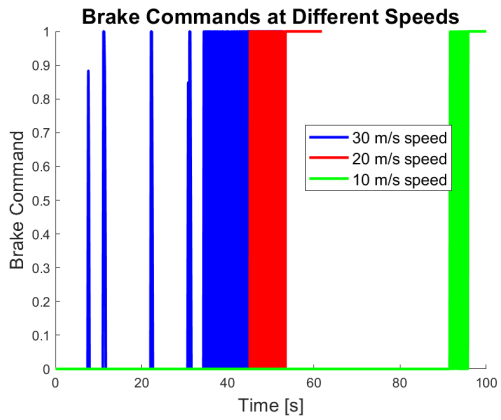
Furthermore, Figure 8.33 delves into the consequential impact on wheel torques resulting from the interplay between lateral and longitudinal control. This figure illustrates the torques applied to the four wheels, revealing how the braking system responds to the lateral adjustments enforced by the MPC algorithm. Understanding these intricate relationships is pivotal for refining and optimizing the integration of lateral and longitudinal control strategies, ultimately enhancing the overall performance and safety of the vehicle



(a) MPC Speed Tracking



(b) Throttle at Different Speeds



(c) Brake Commands at Different Speeds

Figure 8.32: MPC Speed Tracking, Throttle, and Brake Commands

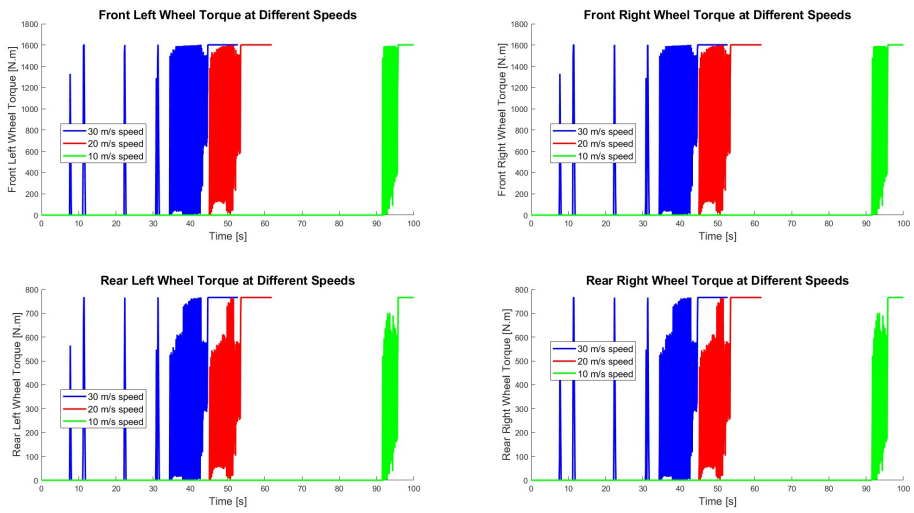


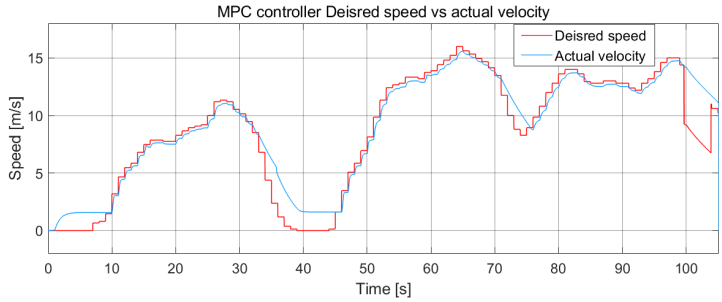
Figure 8.33: Wheel Torques at Different Speeds

8.2.2 Longitudinal Controller

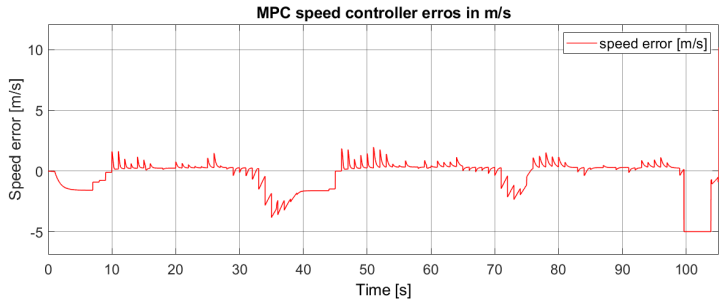
In this section, the two algorithms mentioned for speed tracking in 4 will be tested and validated. Both PID and MPC showed accurate results in the Simulink environment for the 2-DOF model. The controllers were tested using a speed profile with varying speed as shown in figure 8.34a.

MPC Results

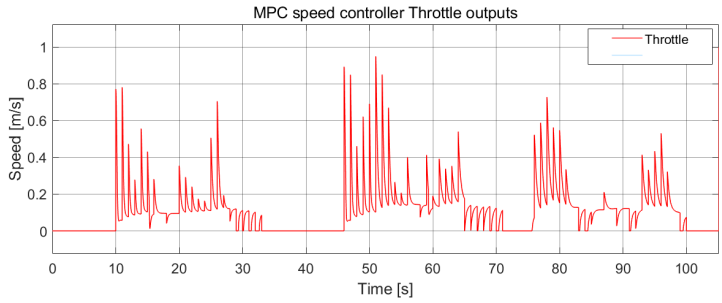
The MPC controller was validated in real-time simulation with a 14-DOF dynamical model, and the results are satisfactory. This is evident not only for varying speeds, but also for tracking constant speeds as shown in the lateral controller results sections (Figures 8.32a and 8.17),.



(a) MPC Speed Tracking Result



(b) MPC Speed Error Result

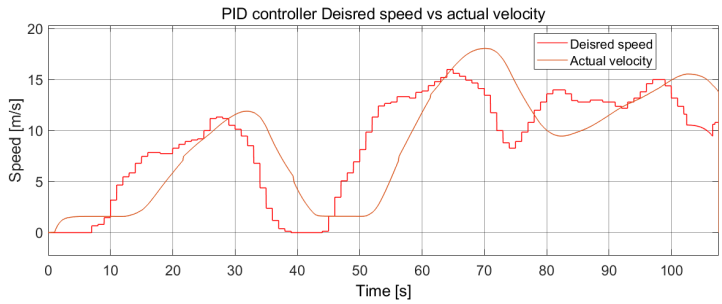


(c) MPC Throttle Result

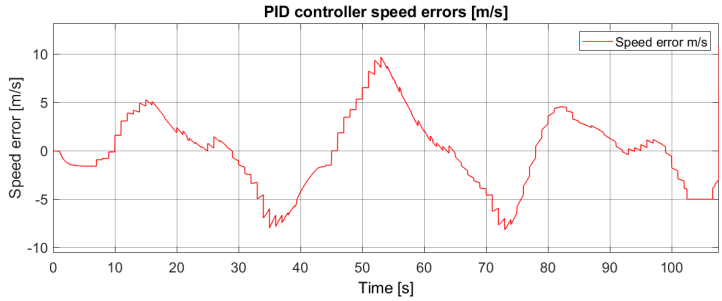
Figure 8.34: MPC Results in the RTL Simulation

In contrast to the PID results depicted in Figure 8.35a, the MPC exhibited superior performance. The maximum error in speed for MPC was -2.8 m/s, while the PID showed a maximum error of 9.8 m/s. Although the PID controller provides a smooth response in the throttle, as illustrated in Figure 8.35c, the MPC's ability to minimize speed tracking errors stands out.

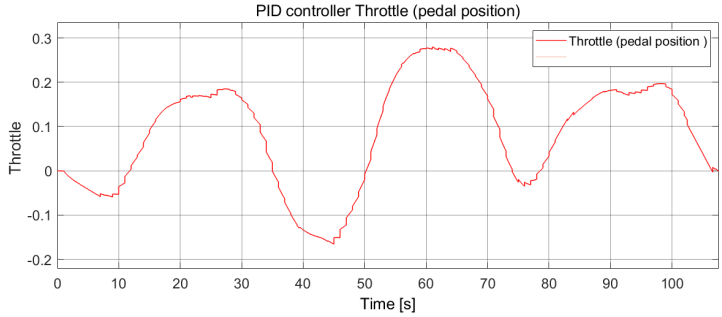
PID Results



(a) PID Speed Tracking Result



(b) PID Speed Error Result



(c) PID Throttle Result

Figure 8.35: PID Results in the RTL Simulation

9. ANALYSIS OF THE IMPLEMENTATION, SUMMARY OF APPLICATION, AND POSSIBILITIES FOR FURTHER DEVELOPMENT

This thesis discusses various tracking and path-planning algorithms for preventing vehicle collisions in both lateral and longitudinal motion directions. From the reviews in the preceding sections, it is clear that a sophisticated system is necessary for successful collision avoidance.

It takes into account a number of variables, including computation costs, constraints (vehicle dynamics, obstructions, and environment), and the use of the most recent computing and sensing technologies. The chosen method for the path planning strategy should be able to offer the host vehicle during navigation flexible worldwide routes to the destination. Furthermore, during the planning phase, it should offer an emergency trajectory that avoids collisions with obstacles and utilises minimal memory and space.

It should also be capable of functioning in intricate, highly non linearity scenarios, take into account moving obstacles, and assess the risk of lane departure following collision avoidance. Following that, path-tracking strategies will function as a controller that follows the path, allowing the vehicle to move along the feasible planned trajectories. The path tracking should have a controller that takes the vehicle model's non linearity into account in order to function as trustworthy automated motion guidance. In order to provide multi-actuator intervention during collision avoidance navigation, it should also have a quick computational time. All things considered, effective path-planning and tracking techniques will make a strong collision avoidance system possible in a variety of traffic situations. The updated Stanley algorithm exhibits impressive path tracking performance, maintaining accuracy even at very high speeds (with a maximum cross-track error of approximately 0.55 m at 115 km/h). While other controllers also demonstrate accurate results, the Stanley controller stands out for its effectiveness, particularly in high-speed scenarios. Regarding longitudinal collision avoidance, the implemented logic successfully prevents collisions and initiates braking even at elevated speeds, video [14] show a demo on the accuracy of Stanley path tracking at constant high speed (115 km/h) with not smoothed braking at the end, while video [15] shows a demo on the updated braking algorithm with smoothing low-pass filter.

In terms of path planning, both Spline-based and Bézier-based methods offer a

smooth path with low computational requirements. These approaches provide computationally efficient solutions, and their resulting paths are not only smooth but also differentiable up to the second order.

Over the past ten years, there has been significant advancement in both academic and commercial collision avoidance technologies in both lateral and longitudinal motion directions. Nonetheless, the vehicle is an object that requires comprehensive study. In order to further increase road safety and ride comfort, future collision avoidance technology should be able to control both lateral and longitudinal movement at the same time.

9.1 Opportunities for Further Development

The obtained results from real-time simulations indicate promising performance; however, there are several avenues for further development to enhance the model's capabilities and robustness. These opportunities include:

1. Signal Smoothing:

To have the best performance possible, the control signals like the steering and the brake/throttle must be further refined and smoothed. To reduce any remaining oscillations or disturbances in the output signals, advanced signal smoothing techniques like adaptive filters or varying low-pass filters.

2. Machine Learning for Object Identification:

The integration of machine learning algorithms for object identification enhance the system's perception capabilities. the identification which can be used for better logic implementation can be done through on machine learning algorithms with low computational powers.

3. HIL (Hardware-in-the-Loop) Testing:

To make sure the model is validated in a real-world environment, hardware-in-the-loop (HIL) testing is necessary. Connecting the simulation model to actual hardware components, such as sensors, actuators, and the vehicle's electronic control unit (ECU),

will produce more realistic results. This allows an assessment of the interactions between the hardware and software and produces more accurate results.

4. VIL (Vehicle-in-the-Loop) Testing:

Taking the validation process a step further, Vehicle-in-the-Loop (VIL) testing involves integrating the developed control algorithms into an actual vehicle. This real-world testing scenario allows for a comprehensive evaluation of the system's performance under varied environmental conditions, road scenarios, and unexpected events. VIL testing is crucial for verifying the algorithm's effectiveness in a practical driving context.

5. Robustness Testing:

To find possible weaknesses in the system and strengthen its resistance to uncertainty, comprehensive robustness testing must be carried out. This entails subjecting the system to difficult scenarios, severe weather, and a variety of road conditions in order to guarantee that the control algorithms remain safe and effective in a variety of circumstances.

Real-world road traffic is getting more complicated every day, despite the industry's tremendous advancements in intelligent connected vehicle collision avoidance. This prediction illustrates the need for continued application of current autonomous driving and V2X communications technologies to enhance collision avoidance as part of ADAS.

Most major automakers have aggressively marketed their developments in developing state-of-the-art collision avoidance systems in recent years. It is important to note that the concept of the Cooperative Vehicle Infrastructure System (CVIS), which incorporates V2X communication into vehicles and transportation, will eventually benefit from state-of-the-art research on collision avoidance. Thus, collaboration between the corporate, academic, and governmental sectors is necessary for the real-time collision avoidance system to be implemented globally.

Since it will ensure fewer road casualties and fewer road fatalities, this could be one way to achieve the ultimate goal of having no road deaths.

BIBLIOGRAPHY

[1] Yaser Abd-AlfatahS Arbre. “a-gokart-autok-jarm-C5-B1-dinamikajanak-attekintese”. In: *Research gate* (2021).

[2] *Analyses of Rear-End Crashes and Near-Crashes in the 100-Car Naturalistic Driving Study to Support Rear-Signaling Countermeasure Development*. url: https://www.nhtsa.gov/sites/nhtsa.gov/files/analyses20of20rear-end20crashes20and20near-crashes20dot20hs2081020846.pdf.

[3] Yutao Ba et al. “Crash prediction with behavioral and physiological features for advanced vehicle collision avoidance system”. In: *Transportation Research Part C: Emerging Technologies* 74 (2017), pp. 22–33. issn: 0968-090X. doi: https://doi.org/10.1016/j.trc.2016.11.009. url: https://www.sciencedirect.com/science/article/pii/S0968090X1630225X.

[4] Guoxing Bai et al. “Path Tracking of Wheeled Mobile Robots Based on Dynamic Prediction Model”. In: *IEEE Access* 7 (2019), pp. 39690–39701. url: https://api.semanticscholar.org/CorpusID:88500550.

[5] Karl-Ernst Biebler and Michael Wodny. “Interpolating cubic splines”. In: 2013. url: https://api.semanticscholar.org/CorpusID:124125783.

[6] Mattias Brännström, Erik Coelingh, and Jonas Sjöberg. “Decision-making on when to brake and when to steer to avoid a collision”. In: *International Journal of Vehicle Safety* 7 (2011), pp. 87–87. url: https://api.semanticscholar.org/CorpusID:109598886.

[7] Matthew Brown et al. “Safe driving envelopes for path tracking in autonomous vehicles”. In: *Control Engineering Practice* 61 (2017), pp. 307–316. issn: 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2016.04.013. url: https://www.sciencedirect.com/science/article/pii/S0967066116300831.

[8] Shuo Cheng et al. “Longitudinal Collision Avoidance and Lateral Stability Adaptive Control System Based on MPC of Autonomous Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 21 (2020), pp. 2376–2385. url: https://api.semanticscholar.org/CorpusID:191138174.

[9] Jessica B. Cicchino. “Effectiveness of forward collision warning and autonomous emergency braking systems in reducing front-to-rear crash rates”. In: *Accident Analysis Prevention* 99 (2017), pp. 142–152. issn: 0001-4575. doi: https://doi.org/10.1016/j.aap.2016.11.009. url: https://www.sciencedirect.com/science/article/pii/S0001457516304006.

[10] *Eurostat an official website of the European Union*. url: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20210624-1 (visited on 11/04/2023).

[11] Reza Hajiloo et al. “Integrated Steering and Differential Braking for Emergency Collision Avoidance in Autonomous Vehicles”. In: *IEEE Transactions on Intelligent Transportation Systems* 22.5 (2021), pp. 3167–3178. doi: 10.1109/TITS.2020.2984210.

[12] Corey D. Harper, Chris T. Hendrickson, and Constantine Samaras. “Cost and benefit estimates of partially-automated vehicle collision avoidance technologies.” In: *Accident; analysis and prevention* 95 Pt A (2016), pp. 104–15. url: https://api.semanticscholar.org/CorpusID:2102688.

[13] G.M. Hoffmann et al. “Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing”. In: *2007 American Control Conference* (2007), pp. 2296–2301. url: https://api.semanticscholar.org/CorpusID:15501370.

[14] Demo for Stanley path tracking at high constant speed 115 km/h, non smoothed braking at the end. url: https://1drv.ms/v/s!AtjJB6NL2c6y2zbkiW7GkE8pszNj?e=qrRFOA.

[15] Demo for updated algorithm Stanley path tracking with smoothed braking at the end. url: https://1drv.ms/v/s!AtjJB6NL2c6y2zieb5n9FOeQ9pl7?e=1zz9yw.

[16] Khisbullah Hudha et al. “Modelling, validation and roll moment rejection control of pneumatically actuated active roll control for improving vehicle lateral dynamics performance”. In: *International Journal of Engineering Systems Modelling and Simulation* 1 (2009), pp. 122–136. url: https://api.semanticscholar.org/CorpusID:110104754.

[17] *Human Performance Evaluation of Light Vehicle Brake Assist Systems*. url: https://www.nhtsa.gov/sites/nhtsa.gov/files/811251.pdf (visited on 06/24/2021).

[18] Jie Ji et al. "Path Planning and Tracking for Vehicle Collision Avoidance Based on Model Predictive Control With Multiconstraints". In: *IEEE Transactions on Vehicular Technology* 66.2 (2017), pp. 952–964. doi: 10.1109/TVT.2016.2555853.

[19] Edward Kasprzak and David Gentz. "The Formula SAE Tire Test Consortium—Tire Testing and Data Handling". In: 2006. url: https://api.semanticscholar.org/CorpusID:110602158.

[20] Edward M. Kasprzak, Kemper E. Lewis, and Douglas L. Milliken. "Inflation Pressure Effects in the Nondimensional Tire Model". In: *SAE Transactions* 115 (2006), pp. 1781–1792. issn: 0096736X, 25771531. url: http://www.jstor.org/stable/44667880 (visited on 12/28/2023).

[21] *Matlab and python modeling (Github link)here*. url: %5Curl%7Bhttps://github.com/Abd-Alfatah/Vehicle-modeling.git%7D.

[22] Amir Mukhtar, Likun Xia, and Tong Boon Tang. "Vehicle Detection Techniques for Collision Avoidance Systems: A Review". In: *IEEE Transactions on Intelligent Transportation Systems* 16 (2015), pp. 2318–2338. url: https://api.semanticscholar.org/CorpusID:5325247.

[23] Hans B. Pacejka and Egbert Bakker. "THE MAGIC FORMULA TYRE MODEL". In: *Vehicle System Dynamics* 21 (1991), pp. 1–18. url: https://api.semanticscholar.org/CorpusID:108456393.

[24] Stephen Zoepf Per Enge Nick Enge. *Electrical Vehicle Engineering*. McGraw Hill, 2021, pp. 65–78.

[25] Guo Qing, Zhang Zheng, and Xu Yue. "Path-planning of automated guided vehicle based on improved Dijkstra algorithm". In: *2017 29th Chinese Control And Decision Conference (CCDC)* (2017), pp. 7138–7143. url: https://api.semanticscholar.org/CorpusID:38682534.

[26] P. Raja and S. Pugazhenthi. “Optimal path planning of mobile robots: A review”. In: *International Journal of Physical Sciences* 7 (2012), pp. 1314–1320. url: https://api.semanticscholar.org/CorpusID:62102698.

[27] Rajesh Rajamani. *Vehicle Dynamics and Control*. Mechanical Engineering Series. New York, NY: Springer, 2006. url: https://www.academia.edu/31492223/Rajesh_Rajamani_Vehicle_Dynamics_and_Control_Mechanical_Engineering_Series_pdf.

[28] Scott Schnelle et al. “A feedforward and feedback integrated lateral and longitudinal driver model for personalized advanced driver assistance systems”. In: *Mechatronics* 50 (2018), pp. 177–188. url: https://api.semanticscholar.org/CorpusID:115986710.

[29] Jitendra Shah et al. “Autonomous rear-end collision avoidance using an electric power steering system”. In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 229.12 (2015), pp. 1638–1655. doi: 10.1177/0954407014567517. eprint: https://doi.org/10.1177/0954407014567517. url: https://doi.org/10.1177/0954407014567517.

[30] Javier Alonso-Mora; Paul Beardsley; Roland Siegwart. “Cooperative Collision Avoidance for Nonholonomic Robots”. In: 2018-April. url: https://ieeexplore.ieee.org/document/8322444/.

[31] DIRK E. SMITH and JOHN M. STARKEY. “Effects of Model Complexity on the Performance of Automated Vehicle Steering Controllers: Model Development, Validation and Comparison”. In: *Vehicle System Dynamics* 24.2 (1995), pp. 163–181. doi: 10.1080/00423119508969086. eprint: https://doi.org/10.1080/00423119508969086. url: https://doi.org/10.1080/00423119508969086.

[32] Shankar C. Subramanian Vignesh Rajaram. “Heavy vehicle collision avoidance control in heterogeneous traffic using varying time headway”. In: 2017-11-10. url: https://doi.org/10.1016/j.mechatronics.2017.11.010 (visited on 05/27/2023).

[33] Thomas A. Wenzel et al. “Dual extended Kalman filter for vehicle state and parameter estimation”. In: *Vehicle System Dynamics* 44 (2006), pp. 153–171. url: https://api.semanticscholar.org/CorpusID:110654890.

[34] Yair Wiseman. “Efficient Embedded Computing Component for Anti-Lock Braking System”. In: 2018. url: https://api.semanticscholar.org/CorpusID:53664244.

[35] *Word health organization, injuries caused by road traffic*. url: https://www.who.%20int/news-room/fact-sheets/detail/road-traffic-injuries,%202020.%201 (visited on 05/27/2023).

[36] Jindong Zhang et al. “Vehicle routing in urban areas based on the Oil Consumption Weight -Dijkstra algorithm”. In: *Iet Intelligent Transport Systems* 10 (2016), pp. 495–502. url: https://api.semanticscholar.org/CorpusID:111750008.

ACRONYMS

2D two dimensions. 76, 91

3D three dimensions. 72, 74, 75, 132

A-MPC Adaptive model predictive controller. 116, 117, 133

DOF Degree of freedom. 1, 13, 14, 16, 18, 22, 31, 60, 61, 67, 82, 119

GUI Graphical user interface. 61, 71, 72, 76–78, 93, 132

MPC Model predictive controller. 1, 37, 41–44, 49, 67, 68, 75, 83–85, 87, 98, 107, 111, 116–120, 131–133

PID Proportional, integral, derivative controller. 1, 37–39, 45, 46, 48, 131

LIST OF FIGURES

2.1 Road accidents: number of fatalities in the EU, 2009 - 2019 [10] 6
2.2 Collision avoidance scenario and decision making tree for the proposed method 7
2.3 The architecture of an Emergency Collision Avoidance System work . . 9
2.4 Longitudinal controller for Emergency collision system 11
2.5 Combined controller logic 12

3.1 Tire suspension model in ij corners 28
3.2 Vehicle top view 29
3.3 Wheel rotational model for braking and driving 31
3.4 Free body diagram for vehicle showing the elementary dynamical forces on vehicle body 34

4.1 Implementation of a PID controller for steering 39
4.2 Stanley geometry (terms description) 40
4.3 MPC Block diagram 42
4.4 PID speed controller implementation in MATLAB/Simulink 45
4.5 Path curvature calculation Simulink block 48

5.1 CarMaker model check 62
5.2 Aerodynamics 64
5.3 Rear axle Brake torque 64
5.4 Front axle Brake torque 65
5.5 Tyre load characteristics as a function of slip with constant vertical load, and as a function of slip and vertical load with a constant side slip. Overview 65
5.6 Tyre characteristics as a function of slip and side-slip with constant slip/side-slip, and as a function of side-slip with constant slip. Overview 66
5.7 Tire friction characteristics as a function of slip with constant coefficient of friction (mu), and as a function of side-slip with constant sideways coefficient of friction (mus). Overview. 66

6.1 Simulink slider gain for parameter tuning 70
6.2 Virtual Vehicle Composer GUI 72
6.3 Data, Calibration, Scenario, and Test Setup 73
6.4 Virtual vehicle project folder structure 73
6.5 Driving scenarios builder in MATLAB 74
6.6 Exporting the scenario as a Simulink block 74
6.7 A 3D view of the road with the vehicle 75
6.8 PID designer app 75
6.9 Set Experiment Parameters 76
6.10 Estimated parameters values plot 77
6.11 GUI of the model advisor 78

7.1 Lateral velocity derivation from lateral acceleration in Simulink 82
7.2 Global position calculation 83
7.3 Overview of full Simulink Model 83
7.4 PID speed and brake regulators Simulink blocks 84
7.5 MPC speed regulator Simulink 85
7.6 PID Steering Controller 86
7.7 Stanley steering control 86
7.8 Path heading calculation 87
7.9 MPC controller Simulink blocks 87
7.10 Different Path Scenarios 89
7.11 Simulink connection path planning for obstacle avoidance 90
7.12 Vehicle Visualization in Simulink 92
7.13 CarMaker home page 93
7.14 CarMaker scenario editing 94
7.15 Vehicle frame reference points 94
7.16 Sensors position setting CarMaker 95
7.17 Generic model CarMaker 96
7.18 Integrating Simulink model with CarMaker 96

8.1 Car Trajectories with Speed Variation 97
8.2 Vehicle trajectory response for the MPC controller. 98
8.3 PID steering Angle without smoothing 98

8.4 Smoothing the PID steering signal 99
8.5 Steering Angle Results 99
8.6 Yaw Angle Results 100
8.7 Yaw Rate Results 100
8.8 PID speed controller results 101
8.9 Path planning algorithm Simulink implementation 102
8.10 Car Trajectories at Different Speeds with PID steering 104
8.11 Trajectory Errors at Different Speeds 104
8.12 PID Steering at Different Speeds 105
8.13 Slip Angle at Different Speeds 105
8.14 Yaw Angle and Yaw Rate at Different Speeds 106
8.15 Wheel Torques at Different Speeds 106
8.16 Brake Commands at Different Speeds 107
8.17 MPC Speed Tracking 107
8.18 Throttle at Different Speeds 108
8.19 Car Trajectories at Different Speeds with the Stanley controller 109
8.20 Trajectory Errors at Different Speeds with Stanley controller 109
8.21 Yaw, slip Angles, yaw rate and Steering with Stanley controller 110
8.22 Brake Commands at Different Speeds 110
8.23 Wheel Torques at Different Speeds 111
8.24 MPC Speed Tracking and Throttle at Different Speeds with the Stanley controller 111
8.25 updated Stanley controller 113
8.26 Stanley Steering Controller Performance 113
8.27 Updated Stanley Controller Results: Yaw Angle, Slip Angle, and Steering Angle 114
8.28 Wheel Torques at Different Speeds 115
8.29 Throttle and Brake Commands with Stanley Steering Controller 115
8.30 Car Trajectories and Trajectory Errors with A-MPC Controller 116
8.31 Yaw Angle, Slip Angle, and Steering Angle with A-MPC Controller . . 117
8.32 MPC Speed Tracking, Throttle, and Brake Commands 118
8.33 Wheel Torques at Different Speeds 119
8.34 MPC Results in the RTL Simulation 120

8.35 PID Results in the RTL Simulation 121

B.1 3-DOF vehicle dynamic 156

Appendices

A. CODE SNIPPETS

A.1 MATLAB codes

A.1.1 Vehicle Visualization Simulink

```
function plotVehicle(Pos_X, Pos_Y, Yaw_Angle, Steering_angle,
    Full_Path, Path, Right_Border, Left_Border)
    % Clear previous vehicle
    cla;

    X1 = Pos_X;
    Y1 = Pos_Y;
    Yaw = Yaw_Angle;
    centerP = [X1; Y1];

    % Dimensions of the car
    car_length = 6;
    car_width = 3;

    % Basis of the car
    p1 = [car_length / 2; car_width / 2];
    p2 = [car_length / 2; -car_width / 2];
    p3 = [-car_length / 2; -car_width / 2];
    p4 = [-car_length / 2; car_width / 2];

    % Rear wheel positions
    prw1 = [car_length / 4; car_width / 2];
    prw2 = [car_length / 4; -car_width / 2];
    prw3 = [-car_length / 4; -car_width / 2];
    prw4 = [-car_length / 4; car_width / 2];

    % Front wheel positions
    pfw1 = [car_length / 2; car_width / 2];
    pfw2 = [car_length / 2; -car_width / 2];
    pfw3 = [-car_length / 2; -car_width / 2];
    pfw4 = [-car_length / 2; car_width / 2];

    % Camera points
    rl = [1 ; 2];
```

```
    rr = [1 ; -2];
    fl = [8; 5];
    fr = [8; -5];
    pend = [2;0];

    % Rotation Matrix
    Rmatrix = [cos(Yaw) -sin(Yaw); sin(Yaw) cos(Yaw)];
    RmatrixS = [cos(Steering_angle + Yaw) -sin(Steering_angle + Yaw);
     sin(Steering_angle + Yaw) cos(Steering_angle + Yaw)];

    % Roateted points
    p1r = centerP + Rmatrix * p1;
    p2r = centerP + Rmatrix * p2;
    p3r = centerP + Rmatrix * p3;
    p4r = centerP + Rmatrix * p4;

    rl = centerP + Rmatrix * rl;
    rr = centerP + Rmatrix * rr;
    fl = centerP + Rmatrix * fl;
    fr = centerP + Rmatrix * fr;

    % Rear wheel
    prw1r = centerP + Rmatrix * prw1;
    prw2r = centerP + Rmatrix * prw2;
    prw3r = centerP + Rmatrix * prw3;
    prw4r = centerP + Rmatrix * prw4;

    % Front wheel
    pendr = centerP + Rmatrix * pend;
    pfw1r = pendr + RmatrixS * pfw1;
    pfw2r = pendr + RmatrixS * pfw2;
    pfw3r = pendr + RmatrixS * pfw3;
    pfw4r = pendr + RmatrixS * pfw4;

    % Connecting points together
    Hitbox_car = [p1r p2r p3r p4r p1r];
    Hitbox_rearwheel = [prw1r prw2r prw3r prw4r prw1r];
    Hitbox_frontwheel = [pfw1r pfw2r pfw3r pfw4r pfw1r];

```

```
    % Plot vehicle as an arrow
    vx = cos(Yaw);
    vy = sin(Yaw);
    quiver(X1, Y1, vx, vy, 'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);

    % Plot the rear wheel as a patch
    patch(Hitbox_rearwheel(1, :), Hitbox_rearwheel(2, :), 'b', '
   FaceAlpha', 0.5);

    % Plot the front wheel as a patch
    patch(Hitbox_frontwheel(1, :), Hitbox_frontwheel(2, :), 'g', '
   FaceAlpha', 0.5);

    % Plot the path
    plot(Path(1, :), Path(2, :), 'k', 'LineWidth', 1.5);

    % Plot the right and left borders
    plot(Right_Border(1, :), Right_Border(2, :), 'r--', 'LineWidth',
   1);
    plot(Left_Border(1, :), Left_Border(2, :), 'r--', 'LineWidth', 1)
   ;

    % Plot the full path
    plot(Full_Path(1, :), Full_Path(2, :), 'c-', 'LineWidth', 1);

    % Adjust the axis
    axis([X1 - 30 X1 + 30 Y1 - 30 Y1 + 30]);

    % Labeling
    xlabel('X-axis');
    ylabel('Y-axis');
    title('2D Plot of the Vehicle');
    grid on;
end
```

Listing A.1: MATLAB code for Vehicle Visualization in Simulink

A.1.2 PID speed regulator

```
function [throttle, brake_cmd] = advanced_speed_and_brake_control(
   path_curv, min_s_speed, speed_act, desired_speed,Kd_speed,Ki_speed
```

```
,Kp_speed)
persistent integrator_speed;
persistent prev_speed_error;
persistent integrator_brake;
persistent prev_brake_error;

if isempty(integrator_speed)
    integrator_speed = 0;
    prev_speed_error = 0;
    integrator_brake = 0;
    prev_brake_error = 0;
end
% Speed Control (PID)

speed_error = desired_speed - speed_act;
integrator_speed = integrator_speed + speed_error;
derivative_speed = speed_error - prev_speed_error;

throttle = Kp_speed * speed_error + Ki_speed * integrator_speed +
Kd_speed * derivative_speed;

% Braking Control (PID)
brake_error = -speed_error;
Kp_brake = 0.05;
Ki_brake = 0.01;
Kd_brake = 0.05;
integrator_brake = integrator_brake + brake_error;
derivative_brake = brake_error - prev_brake_error;
brake_cmd = Kp_brake * brake_error + Ki_brake * integrator_brake
+ Kd_brake * derivative_brake;

% Predictive Braking based on Path Curvature
path_curvature = max(path_curv);
min_safe_speed = max(min_s_speed);
predictive_brake_factor = speed_act/100;
if path_curvature > 0 && speed_act < min_safe_speed
    brake_cmd = brake_cmd + predictive_brake_factor * (
min_safe_speed - speed_act);
end
```

```
    % Update persistent variables
    prev_speed_error = speed_error;
    prev_brake_error = brake_error;
end
```

Listing A.2: MATLAB code for PID speed regulators

A.1.3 MPC speed regulators

```
function throttle_Brake = MPC_speed_control(speed_error,A,B)
    % Model Predictive Controller (MPC) for speed contol
    %speed_ref  = speed_rf+0.4;% correction factor
    %speed_error= current_speed - speed_ref;
    % Define MPC parameters
    Ts = 0.1;  % Sampling time
    N = 10;    % Prediction horizon
    Q = 1;     % State weight % simple model
    R = 1;     % Input weight
    S = 0.1;   % Terminal state weight (integral term)

    % State-space model (is simplified for low computation power
   requirments)

    % Define cost and constraints for fmincon
    costFunction = @(u) (speed_error + B * u)' * Q * (speed_error + B
    * u) + u' * R * u + S * sum(u);
    Aeq = [];
    Beq = [];
    Aineq = [];
    Bineq = [];
    lb = 0; % Lower bound on throttle_Brake
    ub = 1;  % Upper bound on throttle_Brake

    % Steady-state error constraint
    Aineq = [Aineq; B];
    Bineq = [Bineq; -speed_error];

    % Initial guess
    u0 = 0;

    % Solve optimization problem
```

```
    options = optimoptions('fmincon', 'Display', 'off', 'Algorithm', 'sqp');
    u_opt = fmincon(costFunction, u0, Aineq, Bineq, Aeq, Beq, lb, ub, [], options);

    % Extract control input (throttle_Brake) from the optimization result
    throttle_Brake = u_opt;
end
```

Listing A.3: MATLAB code for MPC speed regulator

A.1.4 cross-track error (lateral error)

```
function [cte,denominator]= calculate_cross_track_error(x_vehicle, y_vehicle, x_point, y_point,x2,y2)
    % Calculate cross-track error for a given vehicle position and single point
    function cte = calculate_perpendicular_distance(x_vehicle, y_vehicle, x_point, y_point)
    % Calculate the perpendicular distance from the vehicle to the point

    % Coordinates of the point
    x1 = x_point;
    y1 = y_point;

    % Calculate the cross-track error using the formula for the distance from a point to a line
    numerator = (x2 - x1) * (y1 - y_vehicle) - (x1 - x_vehicle) * (y2 - y1);
    denominator = sqrt((x2 - x1)^2 + (y2 - y1)^2);

    % Ensure that the denominator is not zero to avoid division by zero
    if denominator ~= 0
        cte = (numerator / denominator);
    else
        cte = 0;
    end
end
```

```
    cte = calculate_perpendicular_distance(x_vehicle, y_vehicle,
   x_point, y_point);
end
```

Listing A.4: MATLAB code for cross-track error (lateral error)

A.1.5 Stanley steering

```
function [e, cross_track_steering, steering]  = conversion(dev_angle,
   dev, x1,x2,y1,y2,speed_act)
k_exp = 1.36;
k_def = 6;
k = k_def + abs(speed_act)^k_exp;
if (x2 ==0 && x1 ==0)|| x2==x1
    a = 0;
    b = 1;
    c = y2+a*x2;
else
    a = -(y2 - y1)/(x2-x1);
    b = 1;
    c = y2+a*x2;
end

if speed_act ~=0
    %e =abs(a*x_car+b*y_car+c)\sqrt(a^2+b^2); %abs(Dev_dis);
    e =-1*dev;% deviation distance
    cross_track_steering = atan(e*k / (1+speed_act));% I added 1 here
    to %stablize the vehicle when the speed is so small
    %heading_err = atan(-a/b)-yaw;
    heading_err = dev_angle; % heading error OR deviation angle
    steering = heading_err + cross_track_steering;
else
    steering = 0;
end
end
```

Listing A.5: MATLAB code for Stanley steering

A.1.6 MPC state space model

```
function [A,B,C,D,U, Y, X, DX]= Model

```

```
% Define the vehicle parameters
m = 2100;    % Carbody.mass [Kg]
lf = 1.2;  % Distance from centre of gravity to the front axle [m]
lr = 1.6;   % Distance from centre of gravity to the Rear axle  [m]
I = m*(lf+lr)^2/12;   % Carbody.inertia.z [Kgm2], assuming a
    rectangular shape

cr = 160000; % Slip stiffness rear axle [N/rad]
cf = 100000; % Slip stiffness Front axle [N/rad]

% Define the state variables
% x1: lateral velocity [m/s]
% x2: yaw rate [rad/s]
% x3: lateral position [m]
% x4: yaw angle [rad]

% Define the input variables
% u1: steering angle [rad]
% u2: longitudinal velocity [m/s]

% Define the state-space model
A = [-2*(cf+cr)/(m*u2), -u2-2*(cf*lf-cr*lr)/(m*u2); -2*(cf*lf-cr*lr)
    /(I*u2), -2*(cf*lf^2+cr*lr^2)/(I*u2)];
B = [2*cf/m, 0; 2*cf*lf/I, 0];
% Define some parameters
nx = 4; % Number of states
ny = 4; % Number of outputs
nu = 4; % Number of inputs

C = eye(ny,nx); % Output matrix
D = zeros(ny,nu); % Feedthrough matrix
% Define your state vector
x = zeros(1, nx);  % Initialize the state vector with zeros

% Define your input vector
u = zeros(1, nu);  % Initialize the input vector with zeros

% Define your output vector
y = zeros(1, ny);  % Initialize the output vector with zeros
```

```

% Define the state update function (f) based on your model
function dx = state_update(u, x)
    % Define your state update equations here
    % For example, if you have a discrete-time model A and B matrices:
    A = [-2*(cf+cr)/(m*u(2)), -u(2)-2*(cf*lf-cr*lr)/(m*u(2)); -2*(cf*lf-cr*lr)/(I*u(2)), -2*(cf*lf^2+cr*lr^2)/(I*u(2))];
    B = [2*cf/m, 0; 2*cf*lf/I, 0];

    % Calculate state update using the model
    dx = A .* x + B .* u;
end

% Calculate DX (state update) based on the current state and input
DX = state_update(u, x) - x;

% Reshape signals to the required lengths
X = x;  % Vector signal of length nx
Y = y;  % Vector signal of length ny
U = u;  % Vector signal of length nu
DX = DX;  % Vector signal of length nx

% Now,I have X, Y, U, and DX signals ready for the MPC controller.
% Reshape the model parameters into a row-major order vector
end
```

Listing A.6: MATLAB code for MPC state pace model

A.1.7 instantaneous path generation

```
function instantaneous_points = getInstantaneousPoint(generated_path, lookahead_distance, current_position, current_heading)
    % Parameters
    num_points = size(generated_path, 2);
    rotated_point = zeros(1,2);
    instantaneous_point= zeros(1,2);
    disp(generated_path)
    % Find the closest point on the generated path to the current position
```

```
    distances = sqrt((generated_path(1, :) - current_position(1)).^2 + (generated_path(2, :) - current_position(2)).^2);
    [~, closest_point_index] = min(distances);
     % Extract the last point along the path based on lookahead distance
    cumulative_distances = cumsum(sqrt(diff(generated_path(1, :)).^2 + diff(generated_path(2, :)).^2));
     % Check if cumulative_distances is non-zeros
    if all(cumulative_distances == 0)
        % If there is only one point in the path, use that point
        lookahead_index = closest_point_index;
    else
        % Otherwise, find the index based on lookahead distance
        lookahead_index = closest_point_index + find(cumulative_distances >= lookahead_distance, 1, 'first');
    end
    % Ensure lookahead_index is within valid range
    lookahead_index = min(lookahead_index, num_points);

    % Extract the last point from the path
    instantaneous_point = generated_path(:, lookahead_index);

    % Transform the point to the vehicle's local coordinate system
    delta_x = instantaneous_point(1) - current_position(1);
    delta_y = instantaneous_point(2) - current_position(2);

    rotated_point = [cos(-current_heading) -sin(-current_heading); sin(-current_heading) cos(-current_heading)] * [delta_x; delta_y];

    % Adjust the rotated point to be relative to the current position
    instantaneous_points = [current_position(1) + rotated_point(1) current_position(2) + rotated_point(2)];
end
```

Listing A.7: MATLAB code for instantaneous path generation

A.1.8 Spline based path planning

```
function updated_path = obstacleAvoidanceWithSmoothPath(Car_x,Car_y, dx, dy, object_width, object_length, car_v, pathx, path_y)
    function generated_path = generateSmoothPath(control_points)
```

```
    % Parameters
    num_points = length(control_points);
    generated_path = zeros(2,41);
    % Ensure at least two control points are provided
    if num_points < 2
        error('At least two control points are required.');
    end

    % Extract x and y coordinates of control points
    x_points = zeros(1, num_points);
    y_points = zeros(1, num_points);

    for i = 1:num_points
        x_points(i) = control_points{i}(1);
        y_points(i) = control_points{i}(2);
    end

    % Generate a smooth path using spline interpolation
    t = 1:num_points;
    ts = 1:0.1:num_points; % Increase the resolution for a smoother
   path
    smooth_x = spline(t, x_points, ts);
    smooth_y = spline(t, y_points, ts);

    % Combine x and y coordinates into a path matrix
    generated_path = [smooth_x; smooth_y];
end

% Parameters
    if car_v > 14
        safety_distance = car_v; % Adjust based on your requirements
    else
        safety_distance = 14;
    end

    % Check if the vehicle is close to the object
    distance_to_object = sqrt(dx^2 + dy^2);
    if (distance_to_object - object_length < safety_distance &&
   distance_to_object > 0) && dx > 0
```

```
        % Calculate the control points for generating a smooth path
        vehicle_point = [Car_x; Car_y];
        middle_point = [(Car_x + Car_x + dx) / 2; (Car_y + Car_y + dy) / 2];
        safe_point = [Car_x + dx - safety_distance; Car_y + dy];
        obstacle_point = [Car_x + dx + object_length; Car_y + dy+ object_width/2];
        post_obstacle_point = [Car_x + dx + object_length + safety_distance; Car_y + dy];

        control_points = {vehicle_point, middle_point, safe_point, obstacle_point, post_obstacle_point};

        % Generate a smooth path using spline interpolation
        generated_path = generateSmoothPath(control_points);

        % Update the path to follow the generated smooth path
        updated_path = generated_path;
    else
        % If not close to the object, maintain the current path
        updated_path = [ones(1,41)*pathx; ones(1,41)*path_y];
    end
end
```

Listing A.8: MATLAB code for Spline based path planning

A.1.9 Bezeir path planning

```
function [initial_position, final_position] = calculateInitialFinalPositions(current_position, heading, dy, dx, path_length)
    % Calculate initial position (current position of the vehicle)
    initial_position = current_position;
    obstacle_distance = sqrt(dx^2+dy^2);
    % Calculate final position (position after the obstacle)
    final_position = current_position + [obstacle_distance * cos(heading); obstacle_distance * sin(heading)];

    % Ensure the final position is within a reasonable distance
    if norm(final_position - current_position) < obstacle_distance
        final_position = current_position + [obstacle_distance; 0];
```

```
    end

    % Ensure the final position is within a reasonable distance along the path
    if norm(final_position - current_position) > path_length
        final_position = current_position + path_length * [cos(heading); sin(heading)];
    end
end
```

Listing A.9: MATLAB code for Final posit on calculation for path planning

```
function updated_path = obstacleAvoidanceWithPathPlanning(initial_position, final_position, obstacle_width, obstacle_length, car_width, car_length, dx, dy, car_v, path_x, path_y)
    function generated_path = generateObstacleFreeBezierPath(initial_position, final_position, obstacle_width, obstacle_length, car_width, obstacle_distance)
    % Calculate control points, obstacle position, and obstacle radius
    function control_points = calculateControlPoints(initial_position, final_position, obstacle_width, car_width, obstacle_distance)
        % Calculate control points for Bezier interpolation
        control_points = [
            initial_position;
            initial_position + [obstacle_distance/2; 0];
            initial_position + [obstacle_distance; obstacle_width/2];
            final_position - [obstacle_distance; obstacle_width/2];
            final_position - [obstacle_distance/2; 0];
            final_position
        ];

        % Adjust control points to ensure a smooth path
        control_points(3, 2) = control_points(1, 2) + car_width/2;
        control_points(4, 2) = control_points(6, 2) - car_width/2;
    end
    function obstacle_position = calculateObstaclePosition(initial_position, final_position, obstacle_distance)
        % Calculate obstacle position based on initial and final positions
```

```
        obstacle_position = (initial_position + final_position) / 2;
        obstacle_position(1) = obstacle_position(1) + obstacle_distance;
    end

    function result = bezierInterpolation(t, points)
            % Bezier interpolation function
            n = length(points) - 1;
            result = zeros(size(t));

            for i = 0:n
                result = result + nchoosek(n, i) * (1 - t).^(n - i) .* t.^i * points(i + 1);
            end
        end
        function collision = checkCollision(x, y, obstacle_position, obstacle_radius, resolution)
        % Collision checking function
        distances = sqrt((x - obstacle_position(1)).^2 + (y - obstacle_position(2)).^2);
        collision = any(distances <= obstacle_radius + resolution);
    end

    function adjusted_points = adjustControlPoints(control_points, obstacle_position, obstacle_radius, resolution)
        % Adjust control points to avoid the obstacle
        adjusted_points = control_points;

        % Find the control point closest to the obstacle
        distances = sqrt((control_points(1, :) - obstacle_position(1)).^2 + (control_points(2, :) - obstacle_position(2)).^2);
        [~, closest_point_index] = min(distances);
        disp(distances)
        % Move the closest control point away from the obstacle
        direction = (control_points(closest_point_index, :) - obstacle_position) / norm(control_points(closest_point_index,:) - obstacle_position);
        adjusted_points(closest_point_index,:) = obstacle_position + (obstacle_radius + resolution) * direction;
```

```
    end
    control_points = calculateControlPoints(initial_position, final_position, obstacle_width, car_width, obstacle_distance);
    obstacle_position = calculateObstaclePosition(initial_position, final_position, obstacle_distance);
    obstacle_radius = (obstacle_width + obstacle_length) / 4; % Assuming circular obstacle

    % Parameters
    num_points = size(control_points, 2);

    % Ensure at least two control points are provided
    if num_points < 2
        error('At least two control points are required.');
    end

    % Collision checking parameters
    collision_resolution = 0.1; % Resolution for collision checking

    % Extract x and y coordinates of control points
    x_points = control_points(1, :);
    y_points = control_points(2, :);

    % Bezier interpolation
    t = linspace(0, 1, num_points); % Parameter values
    Bx = bezierInterpolation(t, x_points);
    By = bezierInterpolation(t, y_points);

    % Check for collisions during interpolation
    % Check for collisions during interpolation using a while loop
    i = 1;
    while i <= length(Bx)
        if checkCollision(Bx(i), By(i), obstacle_position, obstacle_radius, collision_resolution)
            % Adjust the control points to avoid the obstacle
            control_points = adjustControlPoints(control_points, obstacle_position, obstacle_radius, collision_resolution);

            % Recalculate Bezier interpolation
```

```
            x_points = control_points(1, :);
            y_points = control_points(2, :);
            Bx = bezierInterpolation(t, x_points);
            By = bezierInterpolation(t, y_points);

            % Restart the collision checking loop
            i = 1;
        else
            % Increment the loop index if no collision
            i = i + 1;
        end
    end

    % Combine x and y coordinates into a path matrix
    generated_path = [Bx; By];
    end

    % Parameters
    if car_v > 14
        safety_distance = car_v; % Adjust based on your requirements
    else
        safety_distance = 14;
    end

    % Check if the vehicle is close to the object
    distance_to_object = sqrt(dx^2 + dy^2);
    obstacle_distance = distance_to_object;
    if (distance_to_object - obstacle_length < safety_distance &&
   distance_to_object > 0) && dx > 0
        % Calculate the control points for generating a smooth path

        % Generate an obstacle-free path using Bezier interpolation
        generated_path = generateObstacleFreeBezierPath(
   initial_position, final_position, obstacle_width, obstacle_length,
    car_width, obstacle_distance);

        % Update the path to follow the generated obstacle-free path
        updated_path = generated_path;
    else
```

```
        % If not close to the object, maintain the current path
        updated_path = [ones(1,2)*path_x; ones(1,2)*path_y];
    end
end
```

Listing A.10: MATLAB code for Bezeir path planning

A.1.10 Desired speed adjusting

```
function desired_speed = DesiredSpeedCollisionAvoidance(current_speed
    , distance_to_obstacle, relative_velocity, speed_to_reach,
    min_speed)

    function braking_distance = calculateBrakingDistance(speed,
    max_deceleration, safety_factor)
    % Estimate the braking distance
        braking_distance = (speed^2) / (2 * max_deceleration) *
    safety_factor;
    end

    % Parameters
    max_deceleration = 5.0; % Maximum deceleration
    reaction_time = 1.0; % Reaction time of the driver
    safety_factor = 1.5;
    % Calculate the remaining braking distance
    braking_distance = calculateBrakingDistance(current_speed,
    max_deceleration, safety_factor);
    % Calculate the time required to stop the vehicle
    time_to_stop = current_speed / max_deceleration;

    % Calculate the required deceleration to stop before the obstacle
    required_deceleration = -current_speed / time_to_stop;

    % Calculate the projected distance the vehicle would cover during
     reaction time
    reaction_distance = current_speed * reaction_time;

    % Calculate the safe following distance based on reaction time
    and braking distance
    safe_following_distance = reaction_distance + braking_distance;

```

```
    % Calculate the effective distance to the obstacle considering reaction time
    effective_distance = distance_to_obstacle - reaction_distance;

    % Calculate the relative closing velocity
    closing_velocity = current_speed - relative_velocity;

    % Determine if collision is imminent and adjust speed accordingly
    if (effective_distance < safe_following_distance-4 && closing_velocity > 0)&& distance_to_obstacle ~=0
        % Apply maximum deceleration to avoid collision
        desired_speed = max(min_speed, current_speed + required_deceleration);
    elseif distance_to_obstacle>0 && distance_to_obstacle <5 && current_speed > 0
        desired_speed = 0;
    else
        desired_speed = speed_to_reach;
    end
end
```

Listing A.11: Desired speed adjusting

A.2 PID speed and brake regulators

```
function [throttle, brake_cmd] = advanced_speed_and_brake_control(
    path_curv, min_s_speed, speed_act, desired_speed,Kd_speed,Ki_speed
    ,Kp_speed)
    persistent integrator_speed;
    persistent prev_speed_error;
    persistent integrator_brake;
    persistent prev_brake_error;

    if isempty(integrator_speed)
        integrator_speed = 0;
        prev_speed_error = 0;
        integrator_brake = 0;
        prev_brake_error = 0;
    end
    % Speed Control (PID)

    speed_error = desired_speed - speed_act;
    integrator_speed = integrator_speed + speed_error;
    derivative_speed = speed_error - prev_speed_error;

    throttle = Kp_speed * speed_error + Ki_speed * integrator_speed +
     Kd_speed * derivative_speed;

    % Braking Control (PID)
    brake_error = -speed_error;
    Kp_brake = 0.05;
    Ki_brake = 0.01;
    Kd_brake = 0.05;
    integrator_brake = integrator_brake + brake_error;
    derivative_brake = brake_error - prev_brake_error;
    brake_cmd = Kp_brake * brake_error + Ki_brake * integrator_brake
    + Kd_brake * derivative_brake;

    % Predictive Braking based on Path Curvature
    path_curvature = max(path_curv);
    min_safe_speed = max(min_s_speed);
    predictive_brake_factor = speed_act/100;
```

```
    if path_curvature > 0 && speed_act < min_safe_speed
        brake_cmd = brake_cmd + predictive_brake_factor * (
   min_safe_speed - speed_act);
    end
    % Update persistent variables
    prev_speed_error = speed_error;
    prev_brake_error = brake_error;
end
```

Listing A.12: Desired speed adjusting

B. MODELS PICTURES

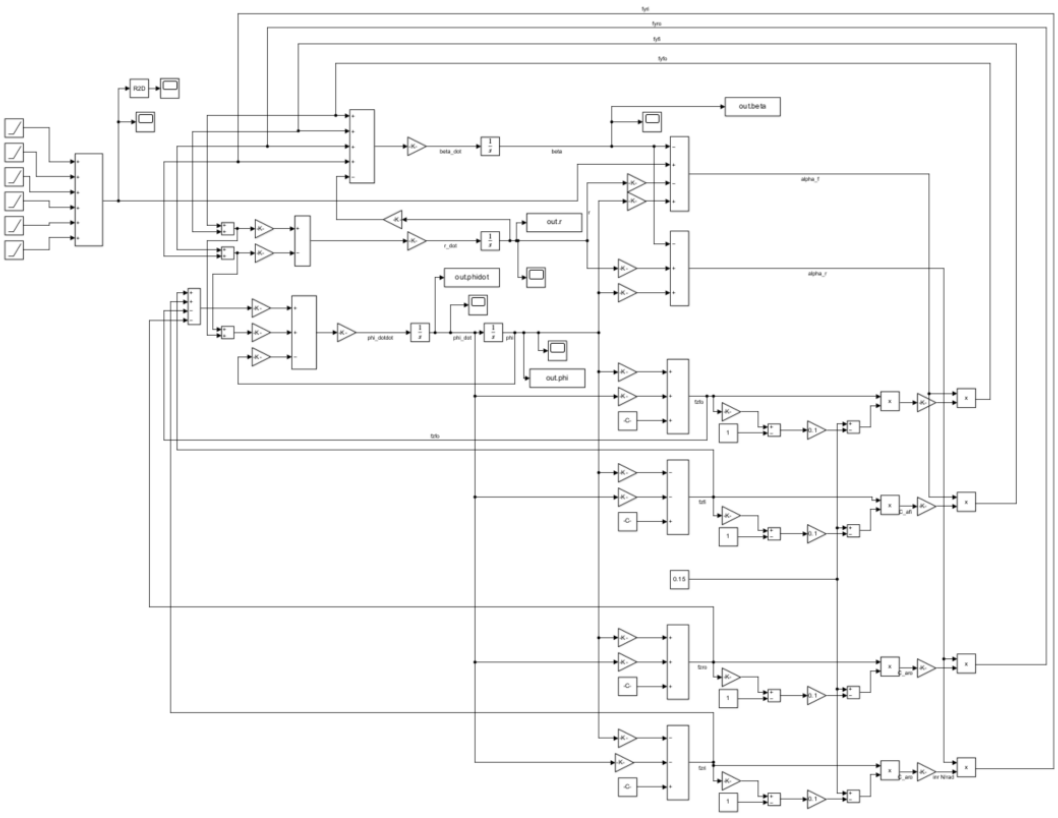


Figure B.1: 3-DOF vehicle dynamic