Obstacle Avoidance and Control of Autonomous Cars Using Model Predictive Control

Dual Degree Project (Stage 1) Report

SUBMITTED BY

Om Prabhu

Roll No : 19D170018

SUPERVISORS

Prof. Avinash Bhardwaj

Mechanical Engineering, IIT Bombay

Prof. K.S. Mallikarjuna Rao

Industrial Engineering & Operations Research, IIT Bombay

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Department of Mechanical Engineering

October 2023

Notation

Contents

List of Figures

Chapter 1

Introduction

1.1 Background of the Project

Model predictive control (MPC) is an optimal control technique in which the calculated control actions minimize a cost function for a constrained dynamic system over a finite time horizon. It has seen major applications in the process control industry, with a major application lying in advanced process control of industrial applications such as oil refineries, kilns and boiler plants. In recent years, it has also been extensively employed in power system balancing models and in power electronics.

MPC controllers predict changes in the dependant variables of a system caused by changes in the independent variables. In most target applications, the autonomous vehicle industry alike, independent variables are often either the setpoints of PID controllers (e.g., acceleration, angular velocity) or the final control element (e.g., steering wheel, throttle/brake system). It is often a computation hassle to adjust the parameters for individual PID controllers at every time step of the process. MPC eliminates this by allowing us to model the entire control system as a single optimization problem and solve it for each finite time horizon.

While most processes are not real in the larger perspective, they can be approximately linearized over a small operating range. This allows for reduced time complexity, by reducing the time horizon of the optimization algorithm, and allows for use of the superposition principle of linear algebra. This enables the effect of changes in multiple independent variables to be added together for computation of the output response, simplifying the control problem to a series of direct algebraic manipulations that are fast and robust.

1.2 Motivation for the Project

Recent trends have witnessed a paradigm shift in the automotive industry, with a significant rise in investments on research regarding autonomous state estimation, prediction and path tracking of vehicles. The current methodology for autonomous driving control involves an explicit segregation of the driving task into lateral and longitudinal control. Lateral control systems, the most common example being adaptive cruise control, focus exclusively on the forward and backward motion of the vehicle (controlled mainly by the throttle and brake system). Longitudinal control deals with the control of steering angles for sideways motion.

The currently accepted control methodology for lateral control is using PID controllers, which calculate the throttle/brake response from an error calculation (most often based on the reference velocity. There are several methods that may be employed for longitudinal control, particularly the pure pursuit control (PPC) method and the Stanley controller. PPC involves the use of the distance between the vehicle's current position from a specific look-ahead point to adjust the controller gain, while the Stanley controller employs use of the heading and cross-track errors to determine the steering output.

In this type of segregated control system, there is almost no interaction between the lateral and longitudinal controller. This can pose severe safety risks. For example, while executing a roundabout maneuver, the vehicle's lateral controller does not always reduce the vehicle velocity before the steering wheel is turned by the longitudinal control system. Additonally, the use of multiple PID controllers can often result in computational difficulties.

The use of MPC allows both the control systems to be integrated into a single optimization problem, with the lateral control system being modeled as a set of constraints on the vehicle's lateral acceleration and velocity. The MPC algorithm can also handle non-linear systems like tire force models, allowing for accurate trajectory tracking. Finally, with some additional processing, MPC can also integrate input from the vehicle's image sensors (cameras, LIDAR and SONAR sensors) to dynamically compute obstacle constraints within the environment of the optimization problem itself. These potential benefits offer compelling reasons for the exploration for MPC as a control system for path tracking of autonomous vehicles.

1.3 Report Outline

The report structure is as follows:

Chapter 1: This chapter includes a brief introduction to the topic and includes the motivation for the project as well as objective for the research work.

Chapter 2: This chapter introduces some important theory that is used to carry out transformations from controller outputs to the dynamic states of a vehicle. This is not explicitly included as part of the code supplied with the simulations discussed in Chapter 4, but is included in the relevant function blocks.

Chapter 3: This chapter gives a brief literature review of previous works in MPC for autonomous vehicle path tracking, including their results, performance and limitations wherever applicable.

Chapter 4: This chapter discusses the current work done as part of the project, including exploration of and some adjustments to implementations of existing simulations, as well as an original model for vehicle path tracking.

Chapter 5: This chapter outlines the future scope and possibilities for the project.

Chapter 6: This chapter lists all the reference materials used so far during the project.

Chapter 2

Theory

This section will discuss some important theory that is relevant to the literature and models discussed in Sections 3 and 4 respectively. The first part of the section briefly covers 2-degree of freedom vehicle dynamics and some of the existing control methods for autonomous vehicle control. The concept of model predictive control is then introduced, including the basic structure of an optimization problem involving MPC, its relevance to the control task of driving an autonomous vehicle, and a general framework for an MPC control system used in autonomous vehicles.

2.1 A Brief Overview of Vehicle Dynamics

2.1.1 The Simple Bicycle Model

2.2 Existing Methods for Control of Self-Driving Vehicles

2.2.1 Proportional-Integral-Derivative Control

2.2.2 Pure Pursuit Control

2.2.3 The Stanley Controller

2.3 Introduction to Model Predictive Control

2.3.1 MPC in Self-Driving Cars

2.3.2 General MPC Framework for Self-Driving Vehicles

Chapter 3

Literature Review

3.1 P. Falcone et al (2007) [1]

3.2 T. Takahama et al (2010) [2]

3.3 T. M. Vu et al (2010) [3]

Chapter 4

Current Work

Note: Some models discussed in this section have certain functions and function blocks that are derivative of existing custom Simulink® libraries and code excerpts from GitHub repositories. In such a case, the relevant sources have been duly cited wherever appropriate.

This section discusses several optimization models for predictive control for self-driving cars. The first sub-section briefly covers re-implementations of some existing models in Simulink® in order to explore the characteristics of an MPC controller and the effect of design parameters on model response. The second sub-section illustrates a Python implementation of an MPC controller for obstacle avoidance.

The discussions in this section are mostly limited to the results and performance of the models, and code excerpts are provided wherever necessary. A detailed code implementation of all the simulations discussed below can be found in the following GitHub repository: `omprabhu31/mpc-self-driving-cars`.

4.1 Re-implementation of existing models

4.1.1 Basic MPC controller model for lateral and steering control

This section discusses the implementation of a basic MPC controller that has been designed to control an autonomous vehicle steering system. The driving task consists of performing a smooth overtake maneuver without having a very high rate of change of the steering angle. Note that the only feedback to the MPC controller is the plant output (i.e., the lateral position and yaw angle). The vehicle states (i.e., velocity, heading and position coordinates) are not part of the feedback loop. As a result, it is not possible for this type of MPC controller to handle varying velocity input. The control loop for this scenario has been illustrated in Figure 4.1.

The simulation discussed above has been implemented by consulting a web-based Mathworks tutorial[4] available on the Internet. The 'reference signal' and 'plant' simulation blocks have been used directly from a Simulink® library called `Meldas_library`. These blocks provide functions for transformation of the controller output (i.e., the steering angle) to the plant output using standard vehicle dynamics equations.

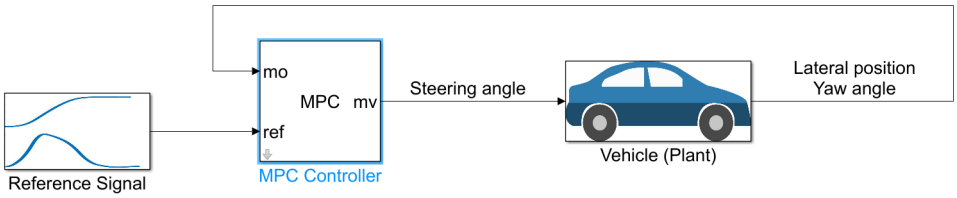

Figure 4.1: Basic MPC Control Loop

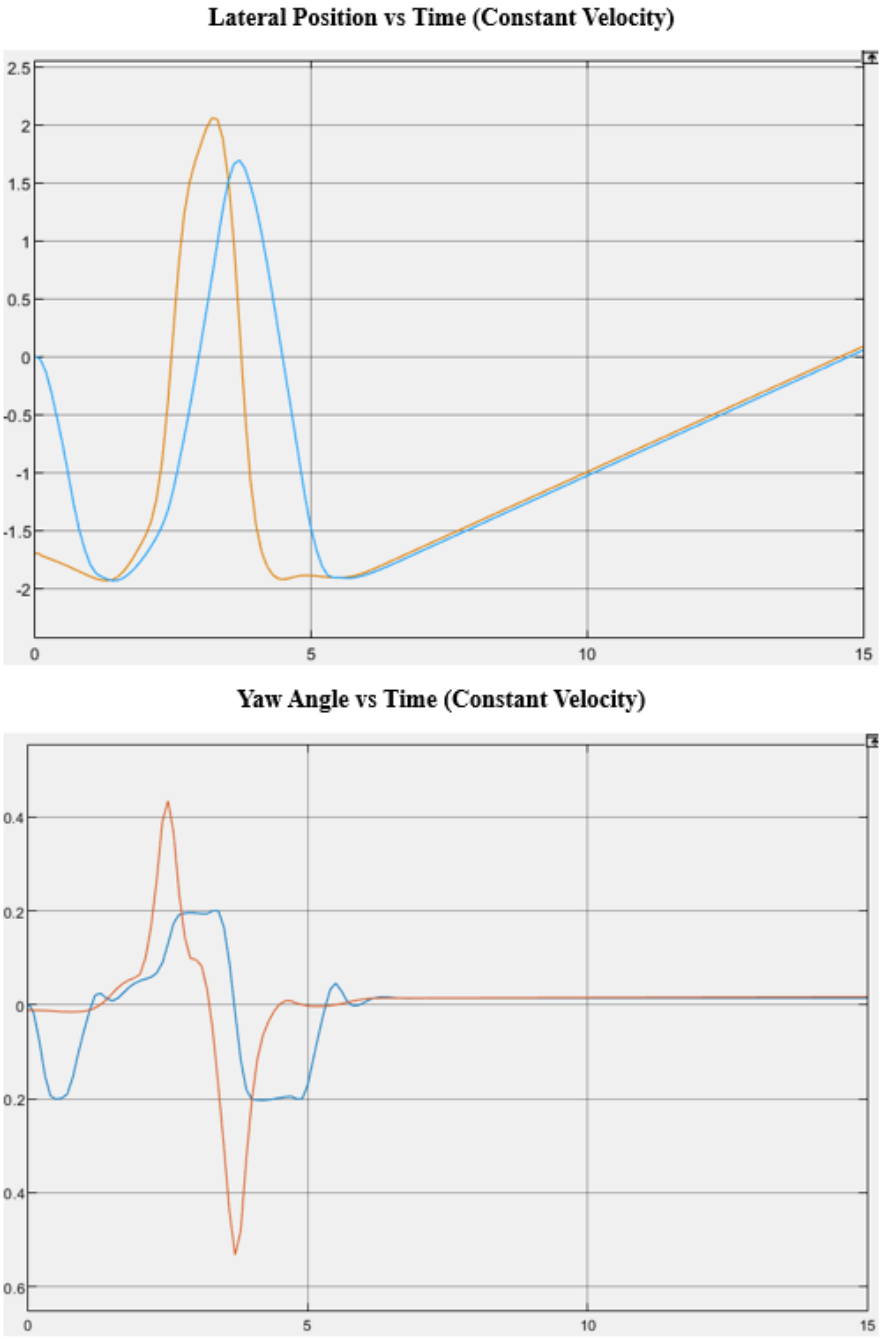

Figure 4.2: Basic MPC Controller Response to Constant Velocity Input

Figures 4.2 and 4.3 illustrate the response of this control system to a fixed velocity input and a sinusoidal velocity input respectively. As observed, the controller has near perfect tracking for a constant velocity input. If the desired velocity or prediction horizon is increased, the controller displays a somewhat tardy response, but still provides an acceptable response for a basic overtake maneuver.

However, for a time-varying velocity input, the controller response is not ideal. The control output (i.e., steering angle) and plant output both diverge with time. To tackle this problem involving varying velocity, we need to implement an adaptive MPC controller which allows for feedback of vehicle states back to the controller.

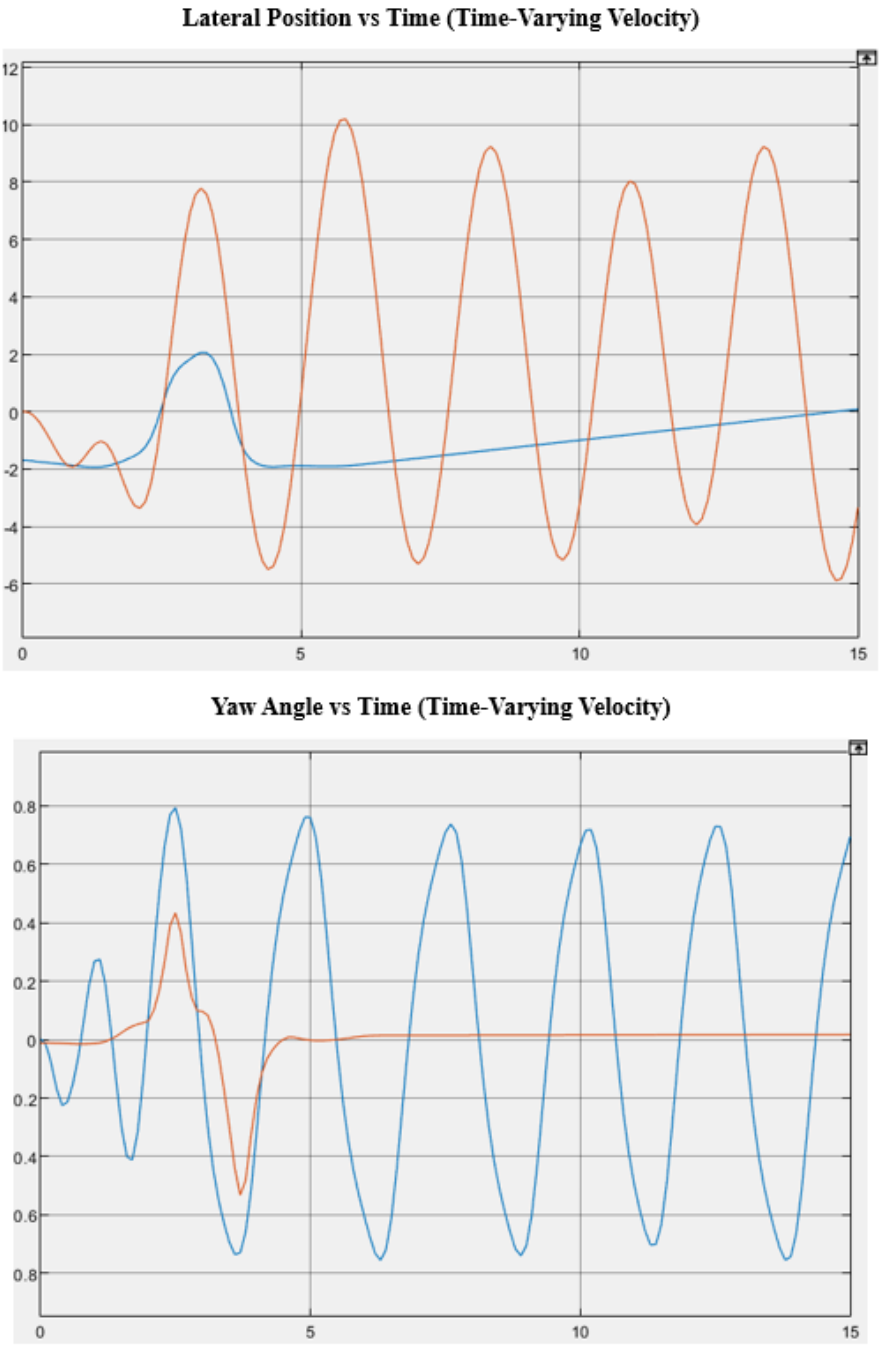

Figure 4.3: Basic MPC Controller Response to Varying Velocity Input

4.1.2 Adaptive MPC controller model for lateral and steering control

This section discusses the implementation of an adaptive MPC controller that has been designed to control an autonomous vehicle steering system. The driving task still remains that of performing an overtake maneuver. The control loop for this scenario has been illustrated in Figure 4.4. Note that, unlike the basic MPC controller, we also introduce the vehicle states as feedback to the controller. Hence, it is possible for us to implement control of vehicle velocity as well as steering angle by updating the vehicle dynamics model at each time step.

The above simulation has been implemented by consulting a web-based Mathworks tutorial[5] available on the Internet. The 'reference signal' and 'plant' simulation blocks have been used directly from a Simulink® library called `Meldas_library`. These blocks provide functions for transformation of the controller output (i.e., the steering angle) to the plant output using standard vehicle dynamics equations.

The `tf_fn` block carries out a transformation from the vehicle states to the state-space matrices of the vehicle model. It makes use of standard vehicle dynamics equations involving the vehicle parameters, states and controller sample time to compute a discrete-time plant model.

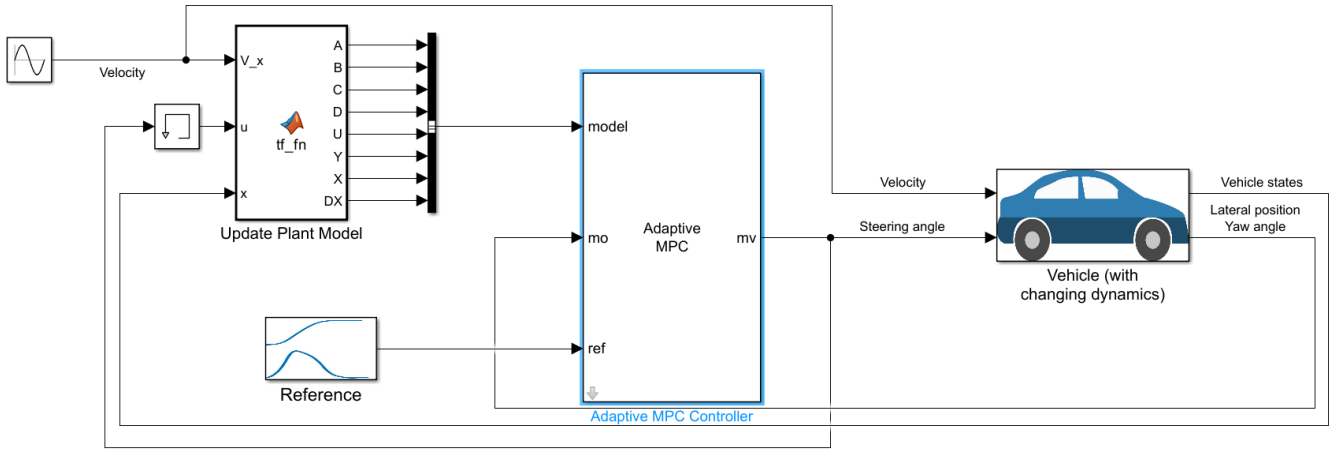

Figure 4.4: Adaptive MPC Control Loop

As expected, the output response in case of a constant velocity is identical to that of the basic MPC controller (refer to Figure 4.2 for output response plots). Figure 4.5 illustrates the response of the adaptive MPC control system to a time-varying velocity input. For simulation purposes, we have employed a sample-based sinusoidal velocity input, but the controller performs just as well for any other varying velocity function. While the traditional MPC controller discussed in the previous section failed to track a time-varying velocity input, the adaptive MPC model is able to satisfactorily track a sinusoidal velocity input.

This example illustrates the effectiveness of an adaptive MPC controller to take into account changing plant dynamics as part of the controller feedback loop. Note that the output response for the yaw angle does deviate slightly compared to the reference signal - this is because we have employed constrained on the maximum and minimum values of the yaw angle as well as yaw rate in order to avoid sudden jerks.

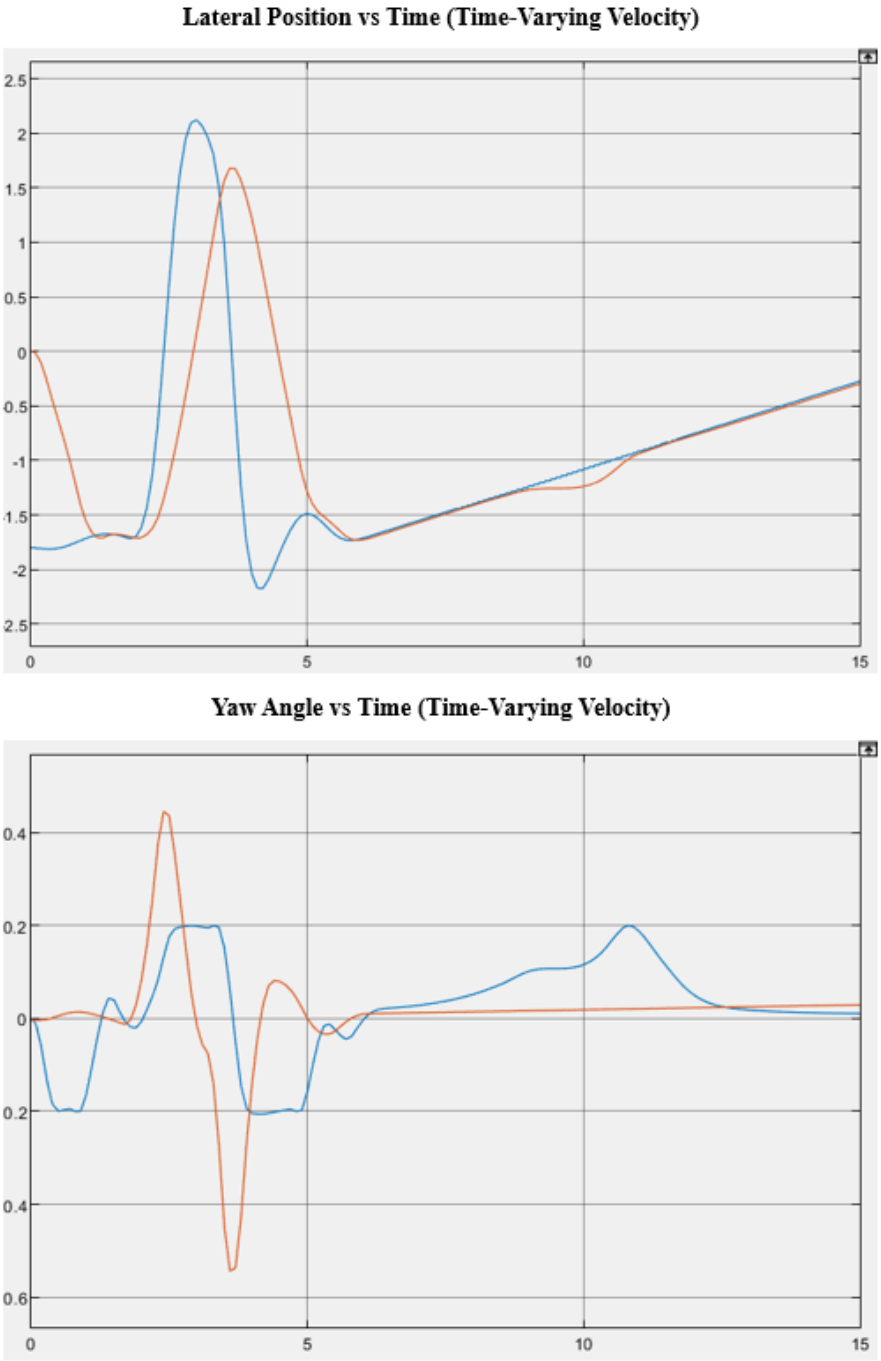

Figure 4.5: Adaptive MPC Controller Response to Sinusoidal Velocity Input

4.1.3 MPC controller model for path tracking across a racetrack

This section discusses a the implementation of an adaptive MPC controller that has been designed to traverse a given racetrack. Because performing steering maneuvers along sharp turns is one of the major challenges faced in autonomous steering control, we select a track comprising two circular loops placed adjacent to each other, mimicking a figure-eight shape. The driving task is to traverse the path with reasonable accuracy.

The simulation of a vehicle traversing a figure-eight loop involves the following steps:

(1) defining the reference points for the figure-eight track by specifying the radius of the circular loops and step size of angles to be traversed

(2) calculating the reference pose and curvature vectors using the linearized distance vectors

(3) transforming the vehicle parameters and state variables (velocity, torque, etc) to compute the state-space matrices for the vehicle dynamics model

(4) compute the curvature function to maintain the vehicle along the curved path

The control for this driving scenario is illustrated in Figure 4.6. Note that this model employs the use of 3-degree of freedom vehicle dynamics instead of the simple 2-degree of freedom bicycle model discussed earlier in Section 2. At the current stage, this lies outside the scope of the project and hence, the code for the same been adapted from an online Mathworks repository[6]. This includes the parts of the simulation involving the powertrain & driveline system and the transformation functions for mapping vehicle states to the 3-DOF model parameters.

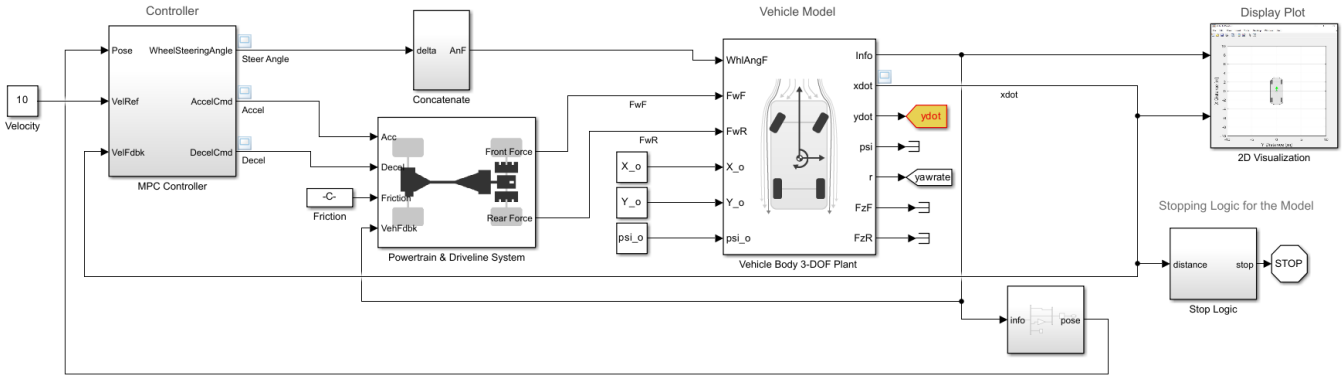

Figure 4.6: Control Loop for Path Tracking Along a Racetrack

Figures 4.7 and 4.8 illustrate the response of the above control system to various operating conditions (red plot represents reference trajectory, black plot represents model response). As observed, the performance of the MPC controller is near perfect for tracking the reference trajectory at reasonably low values of desired vehicle velocity. As we increase the desired velocity, the controller displays a tardy response and the trajectory tracking output is significantly worse. This is an expected behaviour, since the task of turning corners or performing roundabout maneuvers is significantly more difficult at higher speeds even with current manual control vehicles. As is seen in Figure 4.8, the model struggles to keep up with the reference trajectory at high velocities. In fact, setting the vehicle speed to 30 m/s causes the MPC algorithm to reach an infeasible solution while tracking the loop.

Note that the model discussed above only allows for a fixed vehicle speed. However, in real life, we rarely wish to perform sharp turning maneuvers at fixed speeds. To counter this problem, we can instead employ an adaptive MPC controller in order to include the vehicle states as part of the feedback control loop. This potential improvement to the model is discussed in greater detail in Section 5 of the report.

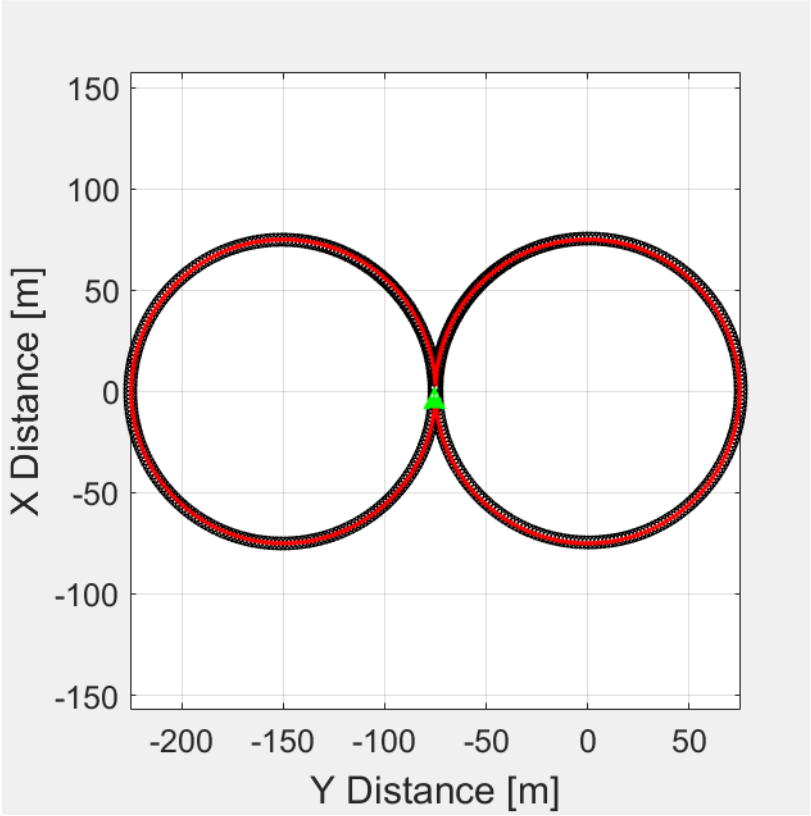

Figure 4.7: Vehicle Path Tracking Response for Low Reference Velocity (10 m/s)

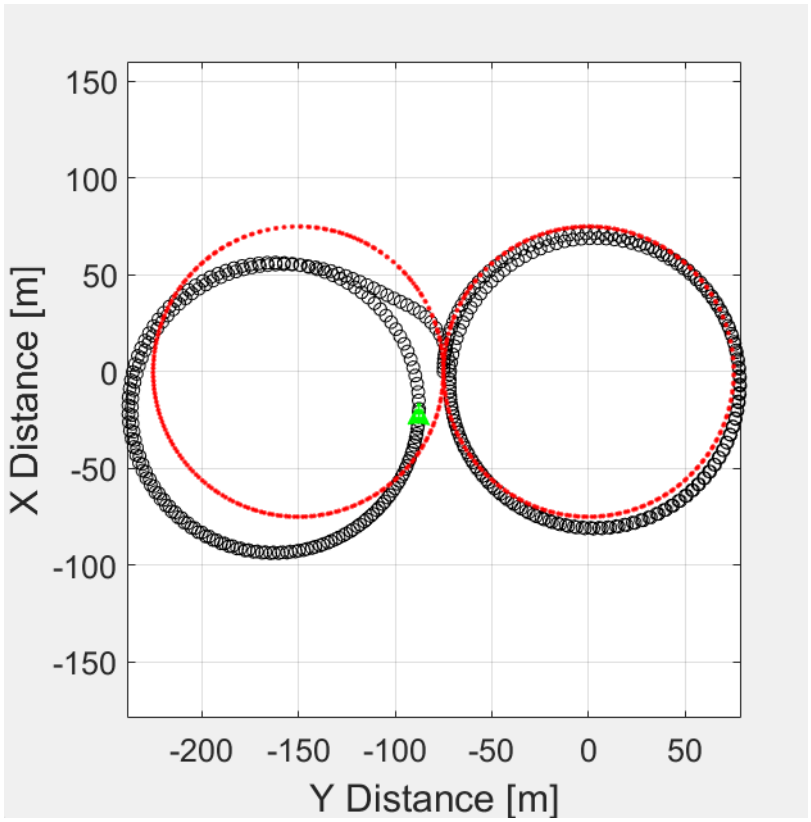

Figure 4.8: Vehicle Path Tracking Response for High Reference Velocity (25 m/s)

4.1.4 MPC controller model for generation of obstacle constraints

This section discusses the implementation of an adaptive MPC controller for dynamic generation of obstacle constraints. The simulation initially starts out with an empty set of constraints (barring the constraints on velocity and steering angle) and updates them upon detection of an obstacle based on several parameters such as obstacle & safe zone dimensions, detection distance and distance of the vehicle from the obstacle. The driving task is straightforward - the vehicle must avoid the obstacle (and preferably also avoid entering the safe zone) by switching to the adjacent lane and then coming back to it's original lane. Without loss of generality, we assume that the vehicle always shifts into the left lane to avoid the obstacle. The control loop for this driving scenario is illustrated in Figure 4.9.

Some aspects of the code for this simulation have been adapted from a web-based Mathwork's tutorial[7], including code for plotting the obstacle avoidance maneuver and relationships between some of the simulation blocks used to store vehicle states and for passing the generated constraints to the adaptive MPC controller.

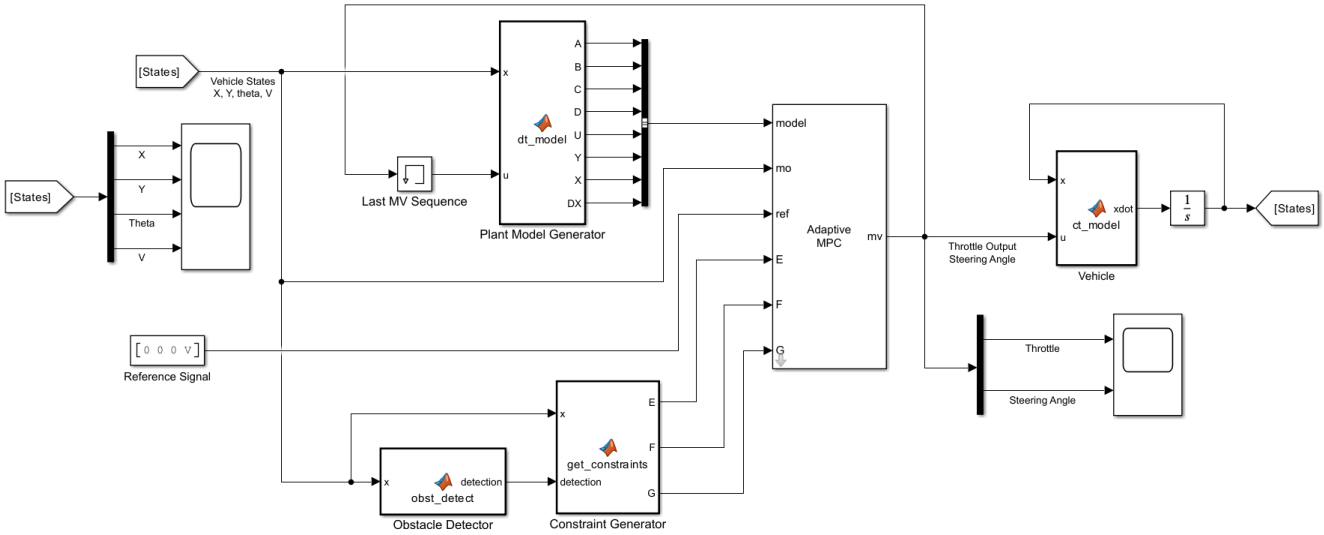

Figure 4.9: Adaptive MPC Control Loop for Obstacle Avoidance

Apart from the velocity and steering angle constraints, we dynamically compute the obstacle constraints at each time step. This is done as follows:

(1) As the vehicle travels along its path, the distance between the vehicle and the obstacle decreases until it hits the obstacle detection distance

(2) Once the vehicle detects the obstacle, it computes a safe zone around the obstacle

(3) For every time step, we impose the following constraints:

  (i) vehicle must not collide with the obstacle (hard constraint)

  (ii) vehicle must not enter the safe zone as far as possible (soft constraint)

  (iii) draw a hypothetical line passing through the nearest corner of the safe zone and the current position - the vehicle must not cross the region to the right of this line (soft constraint)

Depending on the vehicle's starting velocity, it may or may not end up violating the soft constraints. For example, if the starting velocity of the vehicle is very high, it will be unable to decelerate in time - this is because we have also imposed constraints on the rate of acceleration to avoid sudden jerks in the response. In such a case, the vehicle may violate the safe constraints and cross into the obstacle's safe zone and/or the partition defined by the third constraint.

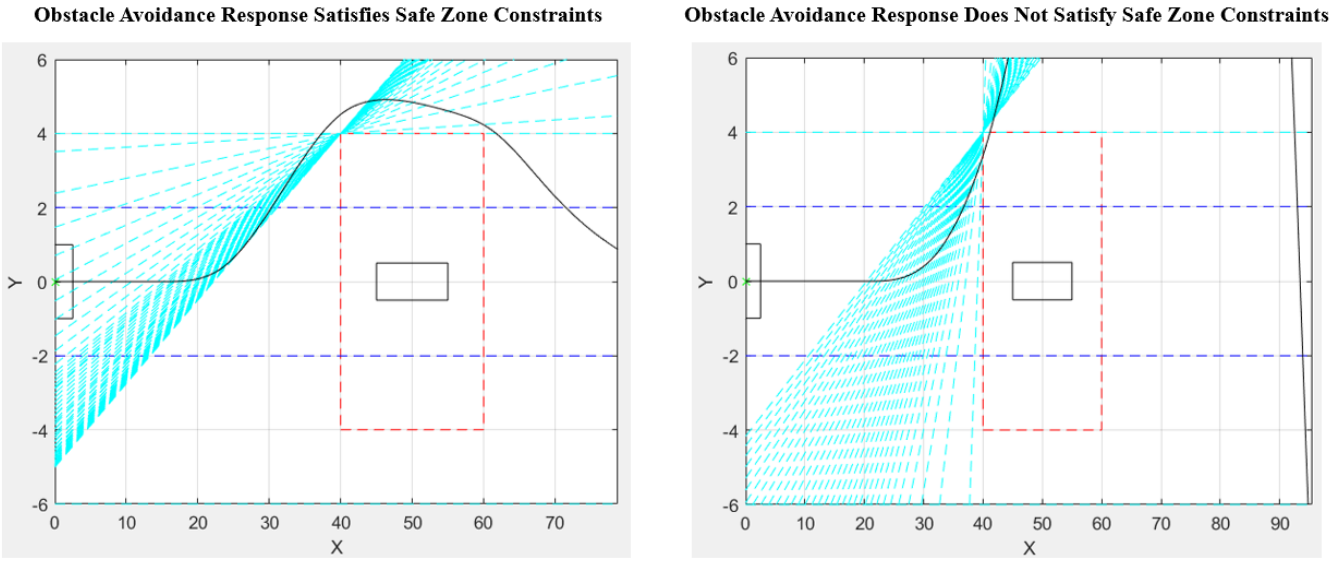

Figure 4.10: Variation of Obstacle Avoidance Response with Initial Velocity

The prediction horizon also has a considerable effect on the output response. In general, the longer the prediction horizon, the smoother the output response. This is because the optimization is solved for a larger number of future time steps. As illustrated in Figure 4.11, the simulation with the smaller prediction horizon displays a much tighter output response. It also crosses into the safe zone while returning back into its original lane. On the other hand, the simulation with the larger prediction horizon provides a much smoother and generally safer obstacle avoidance maneuver. However, a large prediction horizon may cause computational difficulties in some cases.

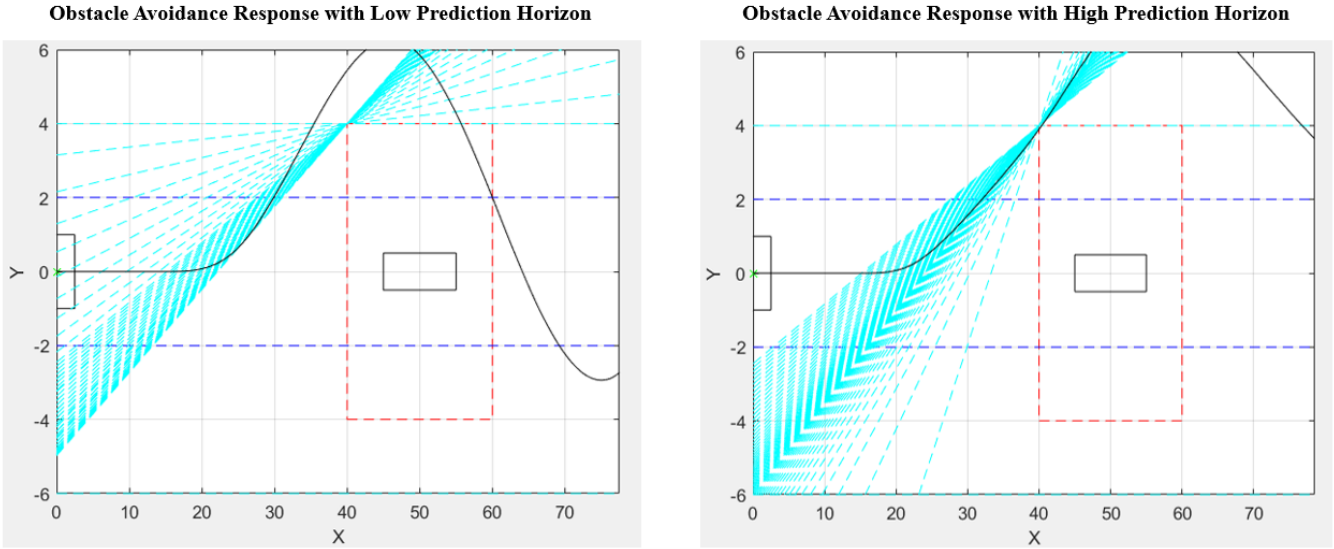

Figure 4.11: Obstacle Avoidance Response for Low and High Prediction Horizons

4.2 Model predictive controller for trajectory tracking and obstacle avoidance

This section discusses a model that performs the driving tasks of tracking a reference trajectory, similar to the model outlined in section 4.1.3, as well as avoids obstacles in its path, much like the model discussed in section 4.1.4. This model has been implemented in Python using the `ipopt` library for solving the optimization problem. The model comprises several files which perform the following functions:

(i) `main.py`: the main model file, which loads the map & simulation environment, sets up and solves the MPC problem, and plots the vehicle's position at each time step

(ii) `MPC.py`: defines a class to store the characteristics of the MPC controller, get data regarding time-varying parameters (waypoints and lateral position errors), and generate obstacle constraints

(iii) `model.py`: defines a class for the simple bicycle model and defines functions for model setup and getting the current waypoint from the reference path

(iv) `reference_path.py`: (adapted from an online GitHub repository[8]) defines the `ReferencePath` class accessed in `main.py` and `model.py`

(v) `simulator.py`: (adapted from an online GitHub repository[8]) defines the `Simulator` class used in `main.py`

(vi) `map.py`: (adapted from an online GitHub repository[8]) defines the `Map` and `Obstacle` classes use in `main.py`

(vii) `globals.py`: defines global variables (distance covered by the vehicle and the prediction horizon) referenced by multiple files in the simulation

Figure 4.12 illustrates the output response of the model at a particular time step. With each time step, the vehicle position is updated on the plot. The obstacle constraints and optimal control outputs are predicted by the MPC algorithm and updated on plot as well. As expected, the model results in a response that tracks the reference trajectory with a high accuracy, except for certain parts where it is forced to go off-track due to the presence of obstacles.

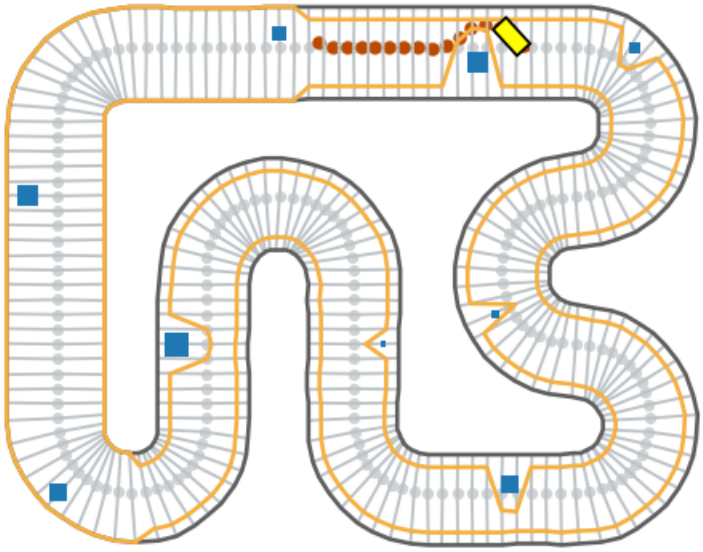

Figure 4.12: Trajectory Tracking and Obstacle Avoidance Response of MPC Controller (Python Simulation)

Chapter 5

Future Work

The project work completed as part of Stage 1 of the project has resulted in significant insights into the behaviour of model predictive controllers. We have successfully implemented basic and adaptive controllers for two different driving tasks, namely trajectory tracking and obstacle avoidance. However, a significant amount of work remains to be done, which will be tackled in the next stage of the project. This includes, but is not limited to:

- **Use of adaptive MPC for trajectory tracking around a racetrack**

  The model discussed in section 4.1.3 was locked to a specific pre-set value of vehicle velocity. This significantly hampered the response of the controller, since we were unable to adjust the vehicle's speed. A possible improvement to this model would be the inclusion of an adaptive MPC controller instead of a basic MPC controller, so that the vehicle states can be fed back to the controller to allow for adjustment of velocity as the vehicle turns along the track.

- **Employing non-linear MPC models**

  The continuous-time vehicle dynamics model is a non-linear one. Up till now, we have only looked at implementations of linear model predictive controllers, which required discretization to convert the non-linear model into a linear one that can be passed as an input to the controller. Though the output responses observed so far have been satisfactory, employing a non-linear MPC controller to be able to handle the continuous-time model without any need for discretization. This could potentially lead to performance improvements and better output responses.

  A major part of the future plan involves the exploration of non-linear MPC to incorporate a spatial bicycle model instead of the simple bicycle model we have been using so far. This will preclude the requirement of discretization, providing a more accurate tracking response. It will also allow for use of more complex tire models such as the Dugoff and Pacejka tire models for more precise computation of vehicle dynamics.

- **Inclusion of image processing algorithms for obstacle detection**

  The obstacle avoidance algorithms discussed in sections 4.1.4 and 4.2 have relied on user-instantiated obstacles. While this is a convenient and less computationally expensive way of assessing the accuracy of a model, obstacles encountered by cars in the real world may not necessarily be pre-determined (e.g., landslides, construction work, etc). To account for this aspect of real-life obstacles, it is important to incorporate input from

image sensors, such as cameras, LIDAR and SONAR sensors, in order to effectively locate any obstacles in the operational design domain of the ego vehicle.

With some additional processing, the sensor outputs can be converted into the obstacle's relative coordinates that can be used to generate obstacle constraints dynamically. The relevant models discussed section 4 can then be employed for the path planning task.

- **Inclusion of other actors in the self-driving environment**

  The simulations discussed in this report have included only the ego vehicle and obstacles, if any. We have not accounted for the presence of other actors in the self-driving environment, such as other vehicles and pedestrians. Hence, the models have been much less complex relative to an actual self-driving task. The inclusion of other actors in the surrounding environment would pose additional challenges for the optimal control problem. However, this would also provide a more accurate measure of how the controller would behave in the real world.

  This will be another major point of focus in the next stage of the project. 2-dimensional top-down simulations or 3-dimensional simulation will be carried out in open-source simulators, such as CARLA or Udacity's self-driving simulator, to demonstrate the effects of real-world driving aspects on MPC controller performance, including but not limited to traffic lights, pedestrian crossings and other vehicle actors in the operational design domain of the ego vehicle.

- **Implementation of reinforcement learning models for the path planning task**

  An alternative approach instead of employing a traditional feedback control system is the use of reinforcement learning algorithms for self-driving tasks. This would mean that the current system framework would be retired, in favour of a neural network that runs a reward-based optimization algorithm for controlling the vehicle. The training of such a type of self-driving algorithm is likely to take a significant amount of time, but it will reduce the number of post-training computations required. A reinforcement learning framework also means that the model learns from its earlier actions, which, in turn, would result in a generally superior ability to predict, follow and respond faster to previously encountered environmental changes as well as drastically reduce the number of repeated errors.

The possibilities mentioned above will be tackled as part of the project's next stage. Of course, this list is not exhaustive, and there may be other methods/improvements that may be discovered over the course of the project.

Chapter 6

References

6.1 Research Papers

[1] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng. Predictive Active Steering Control for Autonomous Vehicle Systems. *IEEE Transactions on Control Systems Technology*, 15(3), 2007.

Link: https://ieeexplore.ieee.org/document/4162483

[2] Daisuke Akasaka, Taku Takahama. Model Predictive Control Approach to Design Practical Adaptive Cruise Control for Traffic Jam. *International Journal of Automotive Engineering*, 9(3), 2018.

Link: https://www.jstage.jst.go.jp/article/jsaeijae/9/3/9_20184095/_article

[3] Trieu Minh Vu, Reza Moezzi, Jindrich Cyrus, Jaroslav Hlava. Model Predictive Control for Autonomous Driving Vehicles. *Electronics, MDPI Open Access Journals*, 10, 2021.

Link: https://doi.org/10.3390/electronics10212593

6.2 Other Resources

[4] Melda Ulusoy (2023). Designing an MPC controller with Simulink, MATLAB Central File Exchange. Retrieved October 11, 2023.

Link: https://www.mathworks.com/matlabcentral/fileexchange/68992

[5] Melda Ulusoy (2023). Adaptive MPC Design with Simulink, MATLAB Central File Exchange. Retrieved October 11, 2023.

Link: https://www.mathworks.com/matlabcentral/fileexchange/68939

[6] Vehicle Path Tracking using Model Predictive Control (Improving Your Racecar Development), Mathworks GitHub Repository, 2022.

Link: https://in.mathworks.com/videos/vehicle-path-tracking-using-model-predictive-control

[7] Obstacle Avoidance Using Adaptive Model Predictive Control, Mathworks Help Center (Automated Driving Applications, Model Predictive Control).

Link: https://in.mathworks.com/help/mpc/ug/obstacle-avoidance-using-adaptive-model-predictive-control

[8] Multi-Purpose-MPC, Online GitHub Repository owned by `matssteinweg`, 2019.

Link: https://github.com/matssteinweg/Multi-Purpose-MPC