Implementing Model Predictive Control on an Open Source Platform to Improve Autonomous Vehicle Research

Chase McNiel (ME), Hengyuan Yang (ME)

May 9, 2022

Contents

1 Executive Summary

This report and project were undertaken by a subgroup under the UC Berkeley ROAR initiative, where ROAR stands for Robot Open Autonomous Racing. The goal of ROAR is to develop 1/10 scale RC cars that can be used for autonomous vehicle research and education projects. This report focuses on the work of implementing Model Predictive Control (MPC) on the ROAR platform. The current ROAR platform uses PID controllers. PID controllers are widely used in controls engineering, but they do have some limitations. Model Predictive Control is an advanced optimal control technique that is popular in the self-driving industry. Since MPC is based on a dynamic model of the car itself, it can provide better control of the car in racing conditions when compared with a PID controller, and it can also be used for path planning as well. Creating an accurate dynamics model was the first task to implementing MPC on ROAR. The dynamic model we developed was then used in formulating a convex optimization program to create our base MPC algorithm. To quickly validate and iterate on our MPC algorithm, we used CARLA, an open-source simulator for autonomous driving research. We were successful in running an online MPC controller in CARLA which validated the accuracy of our dynamics model. We then moved to creating a global path planner for racing in CARLA. Future work of this project includes expanding the global and local MPC planners to create dynamic racing trajectories and testing MPC on the ROAR RC car platform.

2 The Autonomous Vehicle Industry is Progressing Fast, but Research is Still Needed

The autonomous vehicle industry is progressing fast. Analysis has indicated that Level 5 autonomous vehicles, which are capable of self-driving under all conditions, may be commercially available and legal in some jurisdictions by the late 2020s. These first Level 5 capable vehicles will initially have high costs and limited performance [1]. In the 2030s, benefits such as independent mobility for affluent non-drivers may begin. However, most broadly felt impacts, including reduced traffic and parking congestion, independent mobility for low-income groups (and therefore reduced need for public transportation), increased safety, energy conservation and pollution reductions, will only be significant when autonomous vehicles become common and affordable, most likely in the 2040s to 2060s.

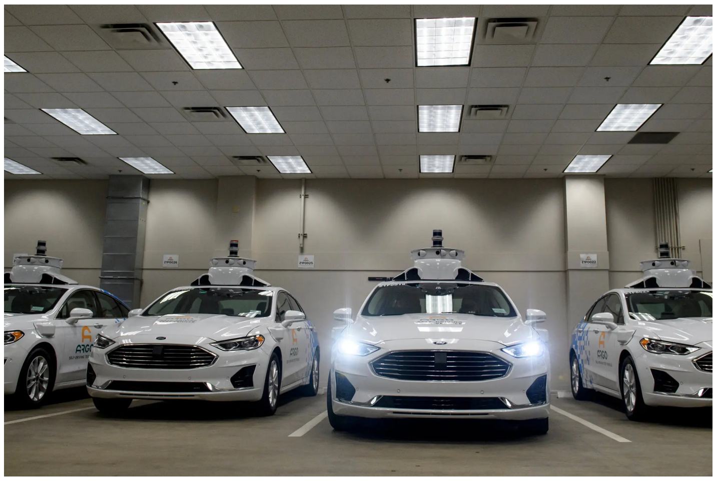

Figure 1: Full Size Autonomous Test Vehicles [2]

To reach those benefits that apply to a wide proportion of the population, research to develop higher levels of autonomous driving will be needed for decades to come. However, autonomous vehicle research is very expensive, and this is one factor that limits the pace of development. Leading companies in the industry like Waymo, Aurora, Uber and Cruise have already spent tens of billions of dollars on developing self-driving cars, and this is projected to continue [3]. According to Pitchbook, each of the major AV companies will spend between 6-10 billion dollars before the technology becomes commonplace, which is targeted to be at the end of the 2030s [4]. The investment of billions of dollars draws unrealistic expectations on the development of autonomous driving technology. As just one example, Elon Musk, the CEO of the electric car company Tesla, said in 2015 that fully functional self-driving cars were just two years away, and even now in 2022, Tesla still only offers lower levels of autonomy suitable only for specific driving environments. Using full sized vehicles is a major reason why research is so time-consuming and expensive, which leads to why ROAR can be such an important tool for researchers.

2.1 The ROAR Program Aims to Lower the Cost of Research

The Robot Open Autonomous Racing (ROAR) program at Berkeley is developing autonomous, 1/10 scale RC cars, with the goal to inspire more students to study the field of autonomous driving and significantly reduce the cost of research for companies. RC cars are a great substitute for full sized vehicles when conducting research on autonomous driving. The RC cars used in the ROAR program are bought from best selling RC car companies, and their mechanical design mirrors that of full sized vehicles, and therefore research done on the RC car platform can be transferred to real cars.

In addition to the physical RC platform, the simulator CARLA is used to validate and iterate upon the autonomy software written for ROAR. CARLA is an open-source simulator to support development and research of autonomous driving systems. CARLA provides open digital assets (urban layouts, buildings, vehicles) that were created for this purpose and can be used freely [5]. With the introduction of this simulation platform to the ROAR program, expenses and physical resources required for researching autonomous driving can be lowered even further.

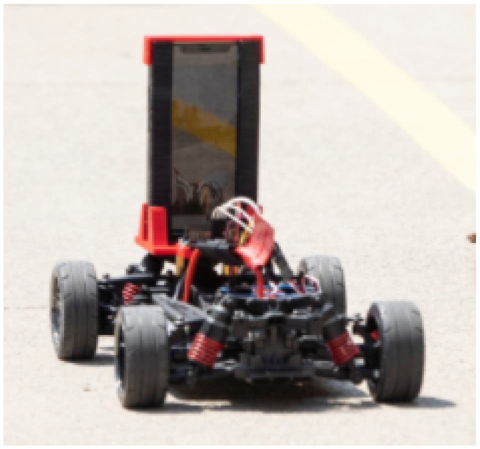

Figure 2: RC Car Setup Used in ROAR Program

2.2 Model Predictive Control is Used Widely in the Autonomous Vehicle Industry

The purpose of our subgroup is to apply Model Predictive Control to the ROAR platform. MPC was first developed in the 1970's as a way to control processes in chemical production facilities that were subject to process and input constraints [6]. MPC is an optimal control algorithm that is used to control a process while satisfying a set of constraints. Traditionally, the areas that MPC could be applied to were limited by the computational resources that it required. MPC relies on dynamic models of the process and solves for optimized results in a finite time-horizon, implements the output in the current time slot and then repeats the process, leading to high requirements for both software and hardware. With the huge increase in computational resources available today, MPC has now been applied as a control method in a large assortment of systems, including autonomous vehicles.

2.3 Adding Model Predictive Control to the ROAR Program Improves The Quality of Research

Prior to our work on the ROAR project, a rigorous examination of the dynamics of the vehicle platform had not been done. This was sufficient for a time as the vehicle system was being controlled with PID controllers. PID controllers do not require a dynamics model of the system to work, but they tend to work best on single input and single output (SISO) systems. An autonomous vehicle however, is an example of a multi-input and multi-output system (MIMO), which motivated our team to develop an MPC algorithm to control the vehicle. Additionally, the dynamics model we developed as part of our project can be used in other areas of the project, including in other control methods like feedforward control or in a linear-quadratic regulator (LQR), or in state estimation as part of a Kalman filter. The next section details the development of our dynamics model for the ROAR platform.

3 An Accurate Dynamics Model is a Crucial Part of Model Predictive Control

MPC is a method of dynamic optimization, meaning that a dynamic model of the system to be controlled is needed [9]. A general form of a discrete, dynamic model is listed below:

$$x(k+1) = g(x(k), u(k)), \;\; x(0) = x_0$$

Here the system state is represented as the vector $x(k)$ and can be of any length. The evolution of the system dynamics is captured by $g(k)$, which can be either a linear or non-linear function. The term $u(k)$ represents the inputs to the system at time k as a vector. The vectors $u(k)$ and $x(k)$ do not need to have the same dimensions.

Dynamic optimization problems are formulated in terms of an objective function that is optimized over a given time span. The objective function can be anything that the control system designer chooses, but generally takes a form similar to the one shown below:

$$\begin{aligned}\min_{U_N} \quad & \sum_{k=0}^{N-1} q(x(k),u(k)) + p(x(N)) \\ \text{s.t.} \quad & x(k+1) = g(x(k),u(k)), \\ & x(0) = x_0\end{aligned}$$

Here $q(x,u)$ is the stage cost and $p(x(N))$ is the terminal cost. Also note that the optimization problem is subject to the constraints of the evolution of the dynamic model and its initial conditions.

As can be seen from the problem formulations above, the effectiveness of any MPC algorithm is dependent on the accuracy of its underlying system dynamics model. If the dynamics model does not accurately predict how the system will evolve over time, then the optimization program will be unlikely to solve for input sequences that will adequately control the real system.

In the next two subsections, we will be constructing a dynamics model for the simulated CARLA vehicle. We need to construct a dynamics model ourselves because there is no documentation about the dynamics model that the CARLA simulator uses. However, this does provide a great opportunity for the ROAR program, as we took great pains to ensure that our dynamics model can be used as the basis to model the physical RC cars that are also used in the ROAR program. Future participants of ROAR can follow the steps we took with the CARLA simulated vehicle, to create an accurate dynamics model for the RC cars in the program.

3.1 The Kinematic Bicycle Model Describes Car Motion Well

For the next two subsections, we closely follow the process laid out in [8] for defining our dynamics model. Many of the steps we describe here are covered in much more detail there. The first model that we begin with is the kinematic bicycle model that has been used extensively in autonomous vehicle research [10]. This model is based on the motion of a bicycle and it requires us to model our vehicle as only having 2 tires. By modeling our system as having two wheels instead of four, we are making the assumption that the front tires travel roughly the same distance as each other, as well as assuming the rear tires travel the same distance as each other. During turns this is not true but the errors introduced by this assumption are small, and make modelling much simpler. The second major assumption we make when using this model is that only the front tires are steerable, which is the case for the ROAR vehicle.

The vehicle's state is modeled as $z = \begin{bmatrix} x & y & v & \psi \end{bmatrix}^T$. The states x and y describe the global position of the vehicle, v is the vehicle's velocity and ψ is the vehicle's yaw angle (which is determined with respect to the global x-axis). Meanwhile, the system input is modeled as $u = \begin{bmatrix} \delta & a \end{bmatrix}^T$, where δ_f is the steering angle and a is the vehicle's acceleration [8].

A diagram of the vehicle model, from [10], is shown in Figure 3. The kinematic

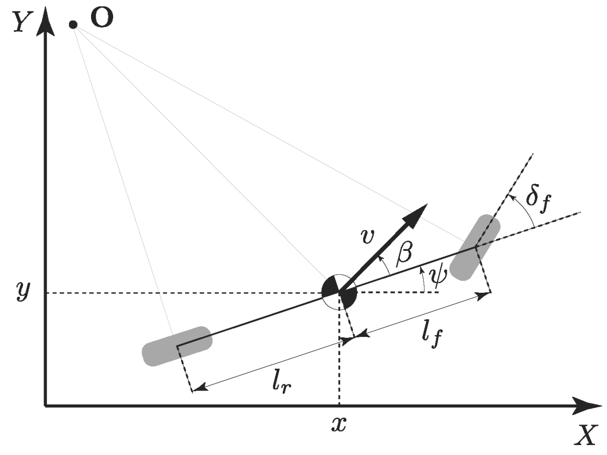

Figure 3: Kinematic Bicycle Model

bicycle model is used for autonomous vehicle research because the equations of motion are simple and easy to use, while being relatively accurate. The equations of motion can be formulated into the five equations listed below:

$$\dot{x} = v\cos(\psi + \beta)$$
$$\dot{y} = v\sin(\psi + \beta)$$
$$\dot{v} = a$$
$$\dot{\psi} = \frac{v}{L_r}\sin(\beta)$$
$$\beta = \tan^{-1}(\frac{L_r}{L_f + L_r}\tan\delta_f)$$

Where β is the angle between the center-line of the vehicle and the velocity vector direction, and is called the slip angle. L_r is the distance from the rear axle to the vehicle's center of mass and L_f is the distance from the front axle to the vehicle's center of mass. These equations do not take into account any of the forces acting on the vehicle. The next section will build upon the kinematic bicycle model and will cover the forces acting on the vehicle in more detail.

3.2 Dynamic Vehicle Model

The next step in developing our dynamics model, is to introduce the forces that compel the vehicle to move in different directions. For our system, we will define forces in terms of the vehicle body coordinate system, as shown in [8]. In this frame we define the longitudinal axis as starting at the vehicle's center of mass and pointing towards the front of the vehicle. Longitudinal forces are denoted with a subscript x, as in F_x, and are those forces that either propel or inhibit the forward motion of the vehicle.

The lateral axis is defined as starting at the vehicle's center of mass, points towards the vehicles left side, and is perpendicular to the longitudinal axis. Lateral forces are denoted with a subscript y, as in F_y. Lateral forces are a result of how the vehicle tires make contact with the surface, and vary based on the steering angle δ and vehicle slip angle β.

As a first step to adding forces to our model, we begin by defining a new state vector as $z = \begin{bmatrix} v_x & v_y & r \end{bmatrix}^T$, where v_x and v_y are the longitudinal and lateral speeds respectively and r is the yaw angle rate. This new state vector allows us to develop state equations based on the vehicle's acceleration, and relate acceleration back to forces that we will later develop models for. Since the simulated vehicle is rear wheel drive only, we denote the sum of the forces in the longitudinal direction as F_x^r. Both wheels in our dynamics model will be assumed to generate a lateral force under the right conditions, and the lateral force generated by the front tire will be F_y^f and the lateral force generated by the rear tire will be F_y^r. With these forces defined, and the geometry given by the kinematic bicycle model, we can define the state equations for our new state vector as follows:

$$\dot{v}_x = v_y r + \frac{1}{m}(F_x^r - F_y^f \sin\delta) \tag{1}$$

$$\dot{v}_y = -v_x r + \frac{1}{m}(F_y^r - F_y^f \cos\delta) \tag{2}$$

$$\dot{r} = \frac{1}{I_z}(L_f F_y^f \cos\delta - L_r F_y^r) \tag{3}$$

We will be covering the lateral and longitudinal forces separately in the next two subsections.

3.2.1 Longitudinal Dynamics Model

We begin developing our longitudinal dynamics model with a simple free body diagram of the forces acting on the vehicle in the longitudinal direction, as seen in Figure 4. The free body diagram shows the forces acting on the vehicle as it drives in a straight line. In other words, the steering angle input, δ, is equal to zero, and there are no lateral forces acting on the vehicle. The only force propelling the vehicle forward is the force from the vehicle's motor, denoted as F_{motor}. There are two forces that inhibit the vehicle's motion, the atmospheric drag force F_{drag} and drive-train friction F_f.

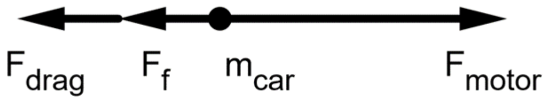

Figure 4: Longitudinal Free Body Diagram

From the free body diagram, we see that the sum of the forces in the longitudinal direction, F_x^r, can be rewritten as shown below:

$$F_x^r = F_{motor} - F_f - F_{drag}$$

Next, we decompose the atmospheric drag and motor forces into functions of system states and inputs. Doing this enables us to model these forces accurately as the system evolves over time. F_{drag} can be rewritten in terms of the vehicle's drag coefficient, C_D, multiplied by the square of the vehicle's longitudinal velocity, v_x. This is a standard method of modeling drag for objects moving through the atmosphere at a steady elevation.

The motor force, F_{motor}, is modeled as being a linear function of the motor input, u_{motor}, which is multiplied by a motor-input constant b. The DC motors used in the ROAR program motivated us to model the force from the motor in this way, because the torque from DC motors is proportional to the current supplied to them. Additionally, we later found that this motor-force model accurately captured the motor dynamics in the CARLA simulator as well.

The drive-train friction force, F_f, is assumed to be a constant, and the final equation for F_x^r can be written as shown below:

$$F_x^r = bu_{motor} - F_f - C_D v_x^2 \tag{4}$$

The next step for developing our longitudinal dynamics model is to identify values for the motor input constant, b, the drive-train friction force, F_f, and the coefficient of drag, C_D. This can be done by running a series of straight line, constant motor input experiments, where we record the car's velocity over time, and then fit parameter values to the recorded data. In our experimental scenario, there will be no lateral forces acting on the car, and the free body diagram shown in Figure 4 will apply.

For the straight line, constant motor input experiments, we recorded velocity data at different input levels from 0.4 to 1.0. The input command in CARLA is normalized so that a command of 1.0 is the maximum input possible. In our experience with CARLA, the vehicle is rarely given a motor command under 0.4, which is why we began our tests at that input level. Figure 5 show a subset of the velocity data that we recorded during these tests.

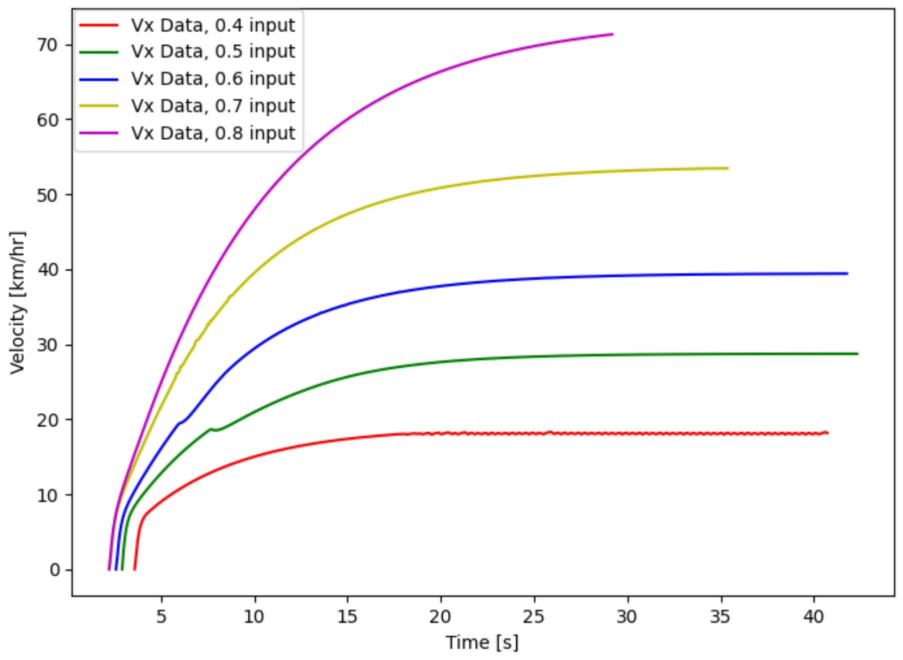

Figure 5: Recorded Velocity Data from Straight Line Experiments

To find the parameter values, we then set up an optimization problem to find the least squares fit between our velocity data and our proposed dynamics model. The full optimization problem can be found below:

$$\begin{aligned}
\min_{b, F_f, C_D} \quad & \Big|\Big|\sum_{k=0}^{T}(v_x[k+1] - v_x[k]) - \Delta t(\frac{1}{m}(F_{motor}[k] - F_f - C_D v_x[k]^2))\Big|\Big|_2^2 \\
\text{s.t.} \quad & F_{motor} = b u_{motor}[k], \\
& b, F_f, C_D \geq 0
\end{aligned}$$

The cost function is first taking the difference in velocity values from one time step to the next as a rough analogue for the vehicle's acceleration. This acceleration is then set to equal the acceleration that we can expect given our longitudinal dynamics model, where F_x^r is used through the use of Newton's 2nd Law. The Δt terms ensures our units are consistent between the two acceleration terms in the cost function. The optimization program than solves for the physical parameters b, F_f, and C_D that minimize the difference between our recorded data, and our proposed longitudinal dynamics model.

CARLA provides acceleration measurements as well as velocity measurements, so why did we record the simulated velocity data instead of acceleration data during our straight line experiments? This would have simplified the cost function in our optimization problem and may have produced more accurate results for our physical parameters as well. The reason we used velocity data instead of acceleration data, is because the physical ROAR RC platform has better sensors for recording velocity than it does for recording acceleration. Our aim is that this work can be extended to the ROAR RC platform in the future, and using velocity data ensures that this section, and optimization script can be used for the RC platform without any modifications to the sensors it already has.

In Table 1, we have tabulated the results from the optimization program for each experiment that we ran. In Figure 6 we plotted our model with the parameter values in Table 1 alongside the velocity data we recorded. The modeled dynamics for each input are shown with the dashed lines. These plots show that our longitudinal model closely matches how the simulated vehicle in CARLA behaves.

Motor Input u	b	F_f	C_D	Steady State v_x
0.4	10800	180	13.6	18
0.5	8875	170	5.4	28
0.6	8943	165	3.5	39
0.7	9530	114	2.4	52
0.8	9105	102	1.5	70
0.9	9429	206	0.86	97
1.0	9845	177	0.46	141

Table 1: Parameter Values Solved From Straight Line Experiments

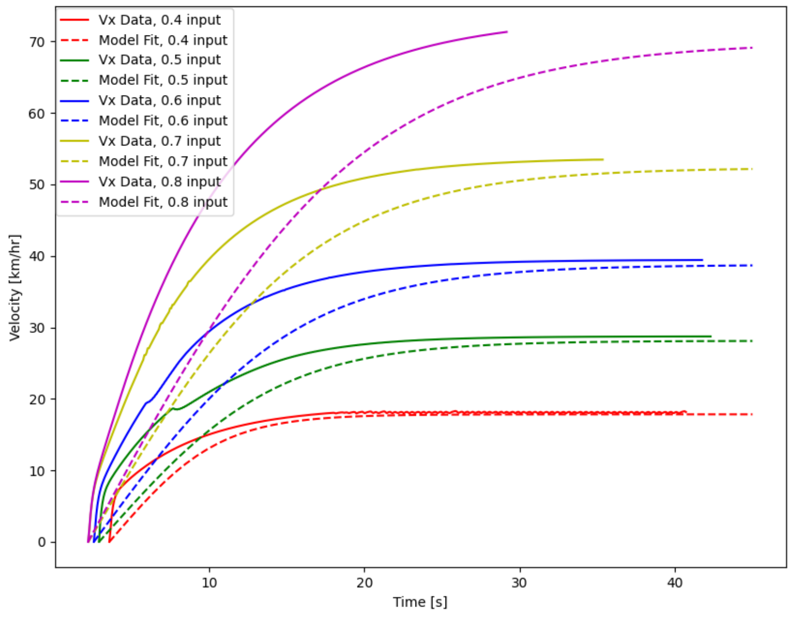

Figure 6: Velocity Data With Model Fits

3.2.2 Lateral Dynamics Model

For our lateral dynamics model, we begin by defining the tire side slip angles for both the front and the rear tires. We only need to define the side slip angle for two tires, since we are using the kinematic bicycle model as the basis for approximating our vehicle's motion. The tire side slip angles are defined as the angle between the tires' center-line and the vehicle's velocity vector. The tire side slip angle is very closely related to the vehicle's side slip angle β. Below, the tire side slip angles are defined completely in terms of the vehicle geometry as shown in Figure 3.

$$\alpha_f = \tan^{-1}(\beta + \frac{rL_f}{v_x}) - \delta_{steer} \tag{5}$$

$$\alpha_r = \tan^{-1}(\beta - \frac{rL_r}{v_x}) \tag{6}$$

The tire side slip angles form the basis for our lateral dynamics model, because many tire models commonly used in the field of vehicle dynamics depend on tire side slip angles to approximate lateral vehicle forces. For this project, we used a slightly simplified Pacejka tire model from [8], which also relies heavily on the tire side slip angle to model lateral forces. The Pacejka tire model that we used is defined below:

$$F_y = -\mu F_z \sin(C \tan^{-1}(B\alpha)) \tag{7}$$

F_y is the lateral force resulting from the tire side slip angle, while μ, C and B are parameters of the model that need to be determined experimentally. F_z is the normal force acting on the tire and is a result of the vehicle weight and vehicle geometry. Since the front and rear tires on the vehicle are identical, this model can be applied to either the front or the rear tire, and only F_z and α need to be changed based on which tire is being modelled. Note, that when α is equal to 0, the lateral force F_y is also 0, which matches our intuition.

To compute the Pacejka tire model parameters for our test vehicle, we conducted a series of constant steering input and constant motor input tests and collected yaw angle and velocity measurements of the vehicle during these experiments. At each motor input level, we ran six experiments with the steering angle δ in the range $[0.05, 0.2, 0.4, 0.6, 0.8, 1.0]$. The car would drive in a circle until it reached a steady state, and at that point we would start collecting data. In Figure 7, you can see the yaw data we collected for the trials we ran while the motor input u was set to 0.4.

We then used the data we collected in the optimization program found below to find the parameters B and C. The cost function in the optimization program attempts

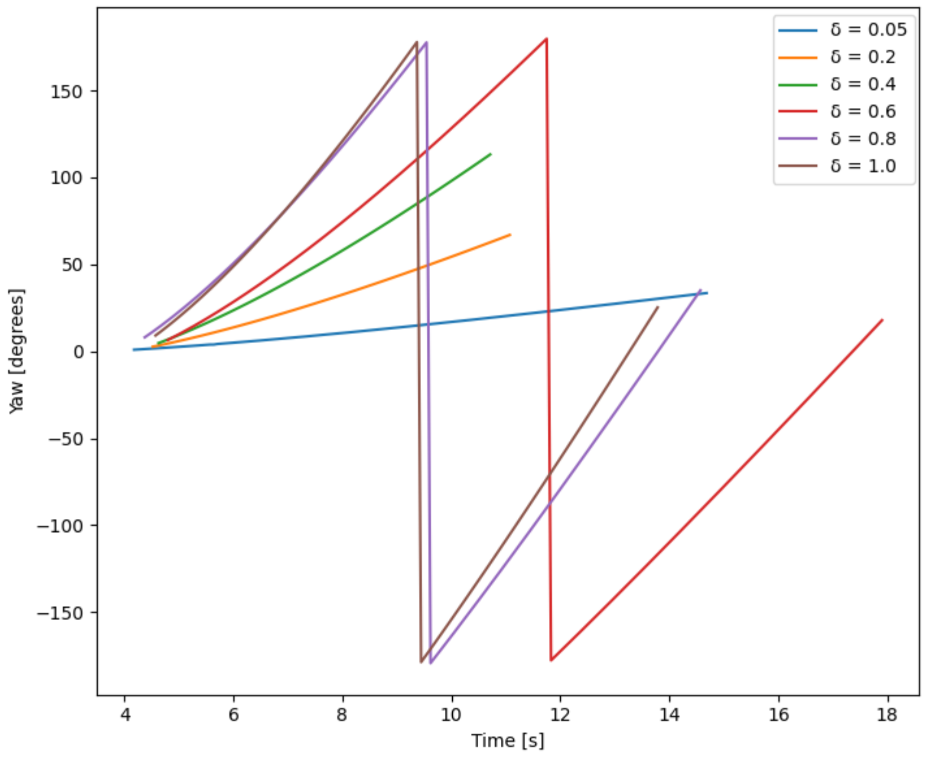

Figure 7: Yaw Data Collected for Constant Steering Input Trials While $u_{motor} = 0.4$

to minimize the sum of the vehicle side slip angle and angular velocity dynamics, using Equations 2 and 3, across all of the experiments that we ran. It is important to note that Equation 2 was originally the lateral velocity dynamics equation, and has been rewritten into a vehicle side slip angle dynamics equation by assuming $v_y \approx v_x\beta$, as shown in [8].

$$\dot{\beta} = -r + \frac{1}{mv_x}(F_y^r - F_y^f \cos\delta)$$

We use the vehicle side slip angle state equation instead of the lateral velocity state equation because there is not a straightforward way to record lateral velocity from CARLA or from the sensors on the physical RC platform.

Since we took measurements after the vehicle reached steady state, $\dot{v}_y$ and $\dot{r}$ should have been close to zero during our experiments. The optimization program took the yaw angle rate, speed and steering angles for each experiment as inputs. We approximated the steady state yaw angle rate r by taking the slope of the yaw angle measurements.

$$\min_{B, C, F_z^f, F_z^r} \quad \sum_{k=0}^{N} \left(\frac{F_y^r(k) + F_y^f(k)\cos\delta(k)}{m v_x(k)} - r(k)\right)^2 + \left(\frac{1}{I_z}(L_f F_y^f(k)\cos\delta(k) - L_r F_y^r(k))\right)^2 \tag{8a}$$

s.t.

$$F_y^f(k) = -\mu F_z^f \sin(C \tan^{-1}(B\alpha_f(k)), \tag{8b}$$

$$F_y^r(k) = -\mu F_z^r \sin(C \tan^{-1}(B\alpha_r(k)), \tag{8c}$$

$$F_z^f = mg - F_z^r, \tag{8d}$$

$$F_z^f \geq .4mg, \tag{8e}$$

$$F_z^r \geq .4mg, \tag{8f}$$

$$F_x^r(k) = b u_{motor} - F_f - C_D v_x(k)^2, \tag{8g}$$

$$F_y^r(k)^2 \leq F_z^{r2} - F_x^r(k)^2, \tag{8h}$$

$$\mu = 1.0 \tag{8i}$$

Constraints 8b and 8c come from the Pacejka tire model and define the lateral forces for the front and rear tires respectively. Constraints 8d - 8f define the normal forces for each tire. Constraint 8g is the force applied at the rear tires in the longitudinal direction, and comes from Section 3.2.1. Constraint 8h comes from the concept of the friction circle for tires and places an upper bound on the rear tire lateral force. In short, a tire is limited by how much force can be applied in different directions, based on the normal force that is being applied to it. And for constraint 8i, we set μ to equal 1.0 because we wanted to keep the modelled lateral forces to always be equal to or less than the normal force acting on the tires.

The results of the optimization program are shown in Table 2. We also used these parameters to plot the lateral forces predicted by our tire model at different tire side

Parameter	Value
B	4.52
C	2.16
F_z^f	7,239 N
F_z^r	10,859 N

Table 2: Parameters Solved for From Lateral Experiments

slip angles. A subset of our experimental data is plotted with the tire model in Figure 8.

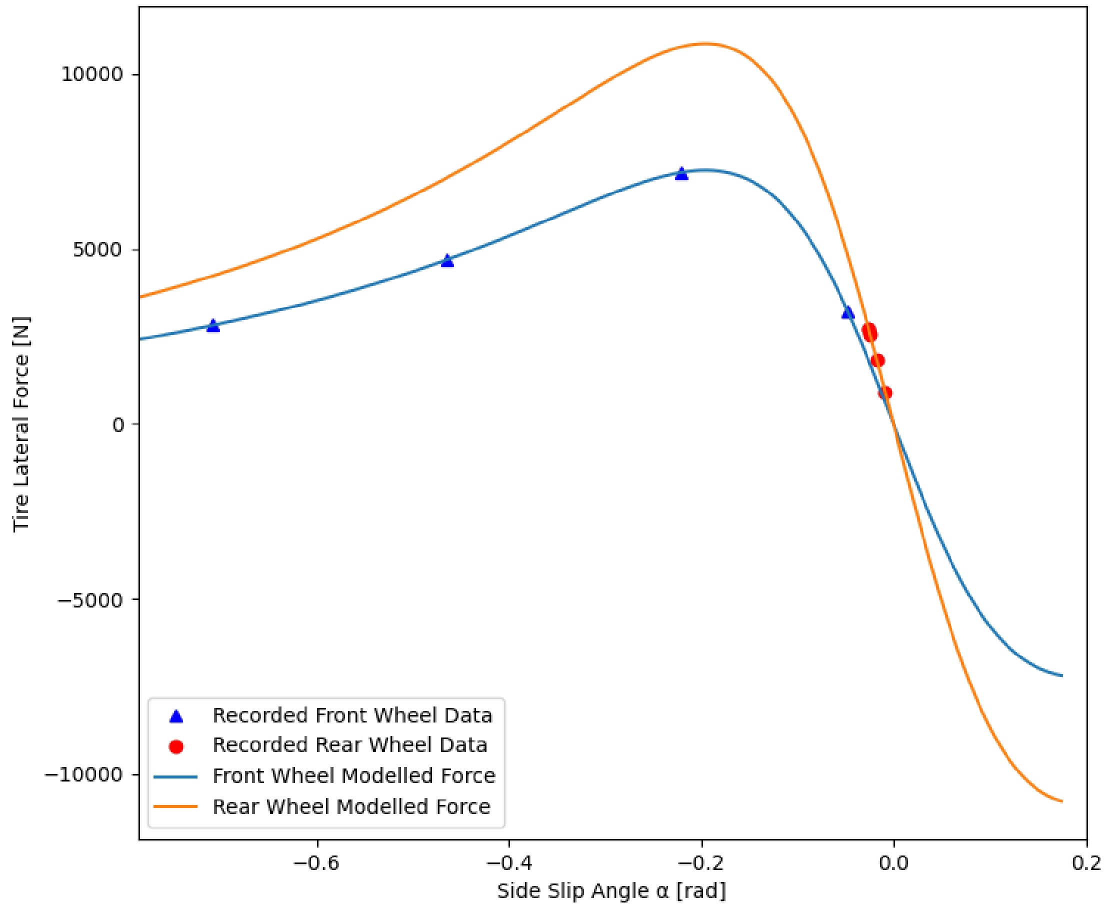

Figure 8: Lateral Force Data Plotted Alongside Modelled Fits

3.2.3 Fully Defined Dynamics Model

Finally, we can write out the full dynamics model we will use with all of our MPC algorithms:

$$\dot{\beta} = -r + \frac{1}{mv_x}(F_y^r - F_y^f \cos\delta) \tag{9}$$

$$\dot{r} = \frac{1}{I_{zz}}(L_f F_y^f \cos\delta - L_r F_y^r) \tag{10}$$

$$\dot{v}_x = v_x \beta r + \frac{1}{m}(F_x^r - F_y^f \sin\delta) \tag{11}$$

As you can see, we have chosen to continue using the vehicle's side slip angle instead of the vehicle's lateral velocity, as we can much more easily determine values for β than for v_y. Our state variables are now $\begin{bmatrix}\beta & r & v_x\end{bmatrix}^T$, while our inputs are $\begin{bmatrix}F_y^f & F_x^r\end{bmatrix}^T$. We can also define every term in Equations 9-11, by either using the kinematic bicycle model, our rear tire longitudinal force model (Eq. 4), our lateral force model (Eq. 7) or with the vehicle parameters given by CARLA, which we have listed in Table 3.

Parameter	Value
m [kg]	1845
I_{zz} [kg * m^2]	779
L_f [m]	1.62
L_r [m]	1.38
L [m]	3.0
h [m]	0.5

Table 3: Simulated Vehicle Properties

4 Our Simulation Results Pave the Way for Future Improvements to the ROAR Platform

In this section, we show the results of our online MPC controller developed for the CARLA simulator. From the data collected from our tests, we conclude that the MPC algorithm still shows areas where it can be improved. Additionally, we show results from a preliminary global path planner that provides a basis for optimizing the racing line based on the vehicle's dynamic model. With some modifications in code and the vehicle model, the MPC controller can be transferred to the RC car platform. Further applications such as dynamic path planning can also be expected based on the simulation results from our project.

4.1 Our Local MPC Controller Shows Some Improvement Over the Current PID Controllers

Before attempting to use our dynamics model in an online MPC controller for CARLA, we developed simulator scripts of the model being used to optimize vehicle inputs over a short time frame (0.5 - 1.0s). We timed how long it took to solve these scripts to get an idea for how long a full MPC control algorithm would take to find a solution at each time step during the CARLA simulation. We used the optimization package Pyomo for these scripts and found that it took between 20 - 40ms for a solution to be found using our full dynamics model. As we were aiming for CARLA to run at 10 fps, we knew that using our full dynamics model in an MPC controller would be too slow to run concurrently with CARLA.

This motivated us to linearize our dynamics model to reduce the solve time at each time step. We followed the linearization process as shown in [8], where our system can be described as:

$$\dot{z} = Az + Bu$$

Here, our state vector z is equal to $\begin{bmatrix}\beta & r & v_x\end{bmatrix}^T$ and our input vector u is equal to $\begin{bmatrix}F_y^f & F_x^r\end{bmatrix}^T$. A is a matrix of the partial derivatives of our state equations (Eq. 9 - 11), taken with respect to our state variables. B is again a matrix of the partial derivatives of our state equations, but this time taken with respect to our input variables. Below are the matrices A and B with their entries defined explicitly:

$$A = \begin{bmatrix} \frac{\partial\dot\beta}{\partial\beta} & \frac{\partial\dot\beta}{\partial r} & \frac{\partial\dot\beta}{\partial v_x} \\ \frac{\partial\dot r}{\partial\beta} & \frac{\partial\dot r}{\partial r} & \frac{\partial\dot r}{\partial v_x} \\ \frac{\partial\dot{v_x}}{\partial\beta} & \frac{\partial\dot{v_x}}{\partial r} & \frac{\partial\dot{v_x}}{\partial v_x} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{\partial\dot\beta}{\partial F_y^f} & \frac{\partial\dot\beta}{\partial F_x^r} \\ \frac{\partial\dot r}{\partial F_y^f} & \frac{\partial\dot r}{\partial F_x^r} \\ \frac{\partial\dot{v_x}}{\partial F_y^f} & \frac{\partial\dot{v_x}}{\partial F_x^r} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

Many of the entries in these matrices turn out to be 0, so we have defined the few remaining non-zero terms below. For matrix A:

$$a_{12} = -1$$
$$a_{13} = -\frac{F_{y,eq}^r + F_{y,eq}^f \cos\delta_{eq}}{m v_{x,eq}^2}$$
$$a_{31} = -v_{x,eq} r_{eq}$$

And for matrix B:

$$b_{11} = \frac{\cos\delta_{eq}}{m v_{x,eq}}$$
$$b_{21} = \frac{L_f \cos\delta_{eq}}{I_{zz}}$$
$$b_{31} = -\frac{\sin\delta_{eq}}{m}$$
$$b_{32} = \frac{1}{m}$$

To ensure our controller remains stable, we also need to linearize our system around equilibrium points. We find equilibrium points by first choosing the steering angle, δ_{eq}, and speed, v_{eq}, we would like to linearize around. From these two chosen values, we can find other values for the remaing variables in our state equations as follows:

$$\beta_{eq} = \tan^{-1}(\frac{L_r}{L}) \tan \delta_{eq}$$
$$v_{x,eq} = v_{eq} \cos \beta_{eq}$$
$$r_{eq} = \frac{v_{eq}}{L_r} \sin \beta_{eq}$$
$$\alpha_{f,eq} = \tan^{-1}(\beta_{eq} + \frac{r_{eq} L_f}{v_{x,eq}}) - \delta_{eq}$$
$$F^f_{y,eq} = -\mu F^f_z \sin(C \tan^{-1}(B\alpha_{f,eq})$$
$$F^r_{y,eq} = L_f F^f_{y,eq} \frac{\cos \delta_{eq}}{L_r}$$

And the final step before implementing our model into an MPC controller in CARLA, is to discretize our linearized system so that it takes the following form:

$$z(k+1) = Az(k) + Bu(k)$$

We do this by simply calling the `scipy.signal.cont2discrete` function on our linearized matrices A and B, which returns discretized matrices A_d and B_d. This function is from the Scipy Python package and uses the Zero-Order Hold method to discretize our system.

We chose to use the CVXPY optimization package in our online MPC controller to reduce solve times even further, as we found CVXPY to be about twice as fast as Pyomo in our simple script experiments. The speed of CVXPY does come at a cost, as it is more difficult to use than Pyomo in our experience. The optimization program we used in our online MPC controller is shown below:

$$
\begin{aligned}
\min_{z_k,\, u_k} \quad & \sum_{k=0}^{N} (z_k - z_r)^T Q (z_k - z_r) \\
\text{s.t.} \quad & z_{k+1} = A z_k + B u_k, \\
& z_{min} \leq z_k \leq x_{max}, \\
& u_{min} \leq u_k \leq u_{max}, \\
& z_0 = \bar{z}
\end{aligned}
$$

The cost function is a simple sum of the squared difference between the system state, z_k, and a reference state, z_r, at each time step k, from 0 to N. Prior groups of students in the ROAR program have implemented waypoints on the CARLA track that become updated as the simulated vehicle drives. The lateral PID controller uses this waypoint system to calculate the cross track angle between the vehicle and the next waypoint position, and the PID controller tries to reduce the cross track angle to 0.

We use the waypoints in a similar way, and set the reference state value for the vehicle side slip angle, β_r, to this cross track angle given by the waypoint system. The cost weight for the vehicle's angular velocity, r_k, is kept to 0 at all time steps, as it gives the solver flexibility to find the optimal solution regarding the other two states, which we care more about. We keep the reference state value for the longitudinal velocity, $v_{x,r}$, set to 100 km/hr at every time step. Our reference state z_r is then equal to $\begin{bmatrix} \beta_r & 0 & 100 \end{bmatrix}^T$ and β_r must be found from the waypoint system each time before running the controller.

The solver finds the optimal inputs for our system, which are F_y^f and F_x^r, for the time steps k from 0 to N. We only care about the optimal inputs at $k = 0$ and only keep those input values while discarding the rest. Since the inputs are given in terms of forces, we convert the optimal inputs at $k = 0$ into terms that are can be commanded to the vehicle. The steering actuator takes the desired steering angle, δ_{des}, and the motor actuator takes the desired motor input, u_{des}. The below equations are used to convert from the optimal forces, to the desired vehicle inputs.

$$F_y^f = -\mu F_z^f \sin(C \tan^{-1}(\beta\alpha_f)$$
$$\alpha_f = \tan^{-1}(\beta + \frac{rL_f}{v_x}) - \delta_{des}$$
$$F_x^r = bu_{des} - F_f - C_D v_x^2$$

In Figure 9 we show how the MPC controller follows the waypoints on the first section of the ROAR track compared to the PID controller setup. As can been seen, the MPC controller does a slightly worse job at tracking towards the given waypoints. In Figure 10 however, the MPC controller appears to control the vehicle to a desired speed, which is 100 km/hr in this case, more effectively than the PID controller.

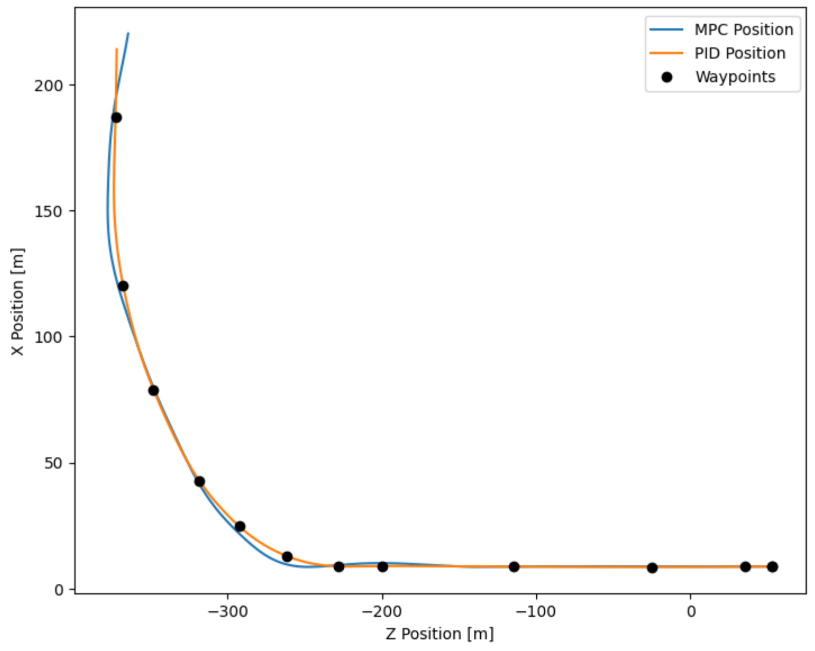

Figure 9: Online MPC vs PID at Tracking Waypoints

There are many avenues to improve the performance of the online MPC controller. The cost function we had in the optimization program was very simple, and did not attempt to minimize the race time of the vehicle at all. Additionally, the way we

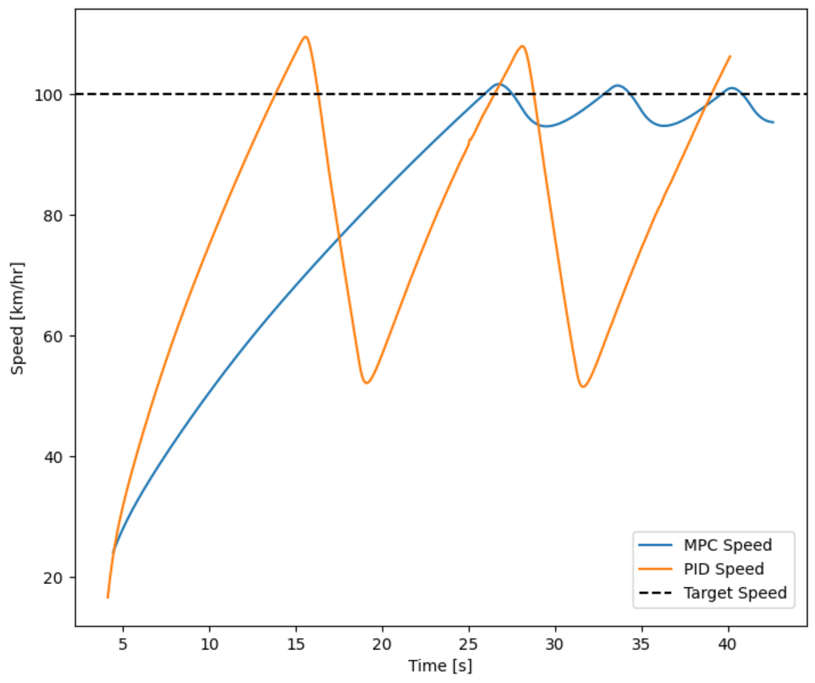

Figure 10: Online MPC vs PID at Speed Control

calculated the reference state for the vehicle's side slip angle, β_r, was probably not correct, as we chose to use the same set up as the PID controller for quickly validating the controller design instead of determining what the reference side slip angle actually needed to be set to.

4.2 Our MPC Global Path Planner Provides A Basis For Improving Raceline Optimization in ROAR

A global path planner is another application of the vehicle dynamic model we developed. The online MPC controller can be used for predictions for the close vicinity when the car is running, while the global planner sees the track as a whole and computes the optimized racing line for the car based on the dynamic model developed. The global planner can be run offline from the simulator, as long as the track data and the vehicle dynamic model are precise, which means limiting the overall computation time is not required. Therefore, we chose to use the nonlinear version of the vehicle dynamics model for the global planner.

In order to construct the track model, a model of the track used in CARLA is required. The manual driving script in CARLA was modified to record position data when running. Multiple tests were run and the best ones were selected for building the track. The reference trajectory is then obtained by driving in the middle of the track for optimizing the racing line. Then the position data was used to generate a track model as shown in Figure 11.

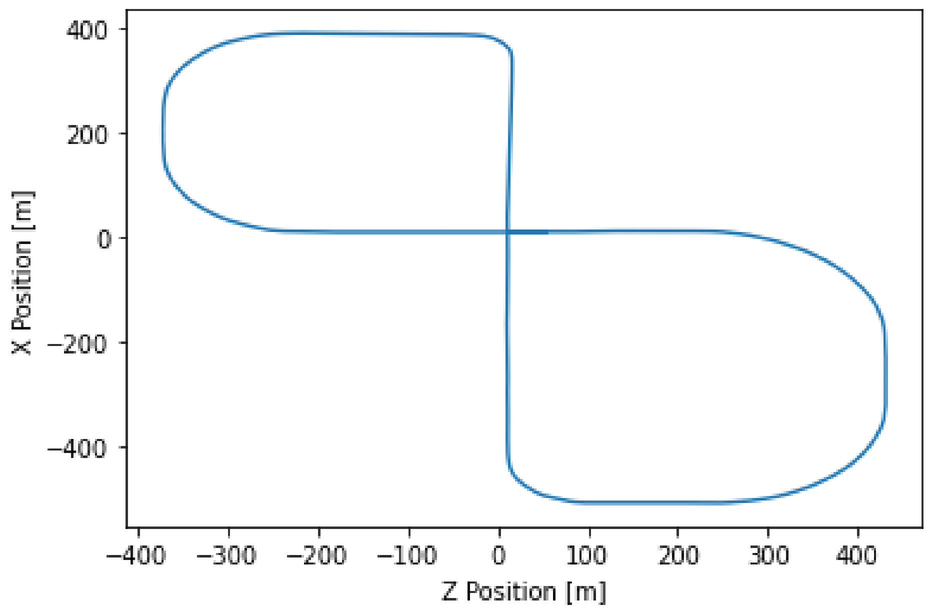

Figure 11: Reference Trajectory

There were issues with the intersection in the middle of the track in our global planner program. Therefore, we chose to only use the first half of the track. Results from the global planner are shown in Figure 12. The optimized trajectory does not look smooth because of the setup of the optimization program and the cost function used. The nonlinear global planner segmented the track into small sections and the cost function was looking at reducing how long it took to travel from the start to the end of that individual section. A more sophisticated cost function will be required to solve the problem by treating the track as a whole.

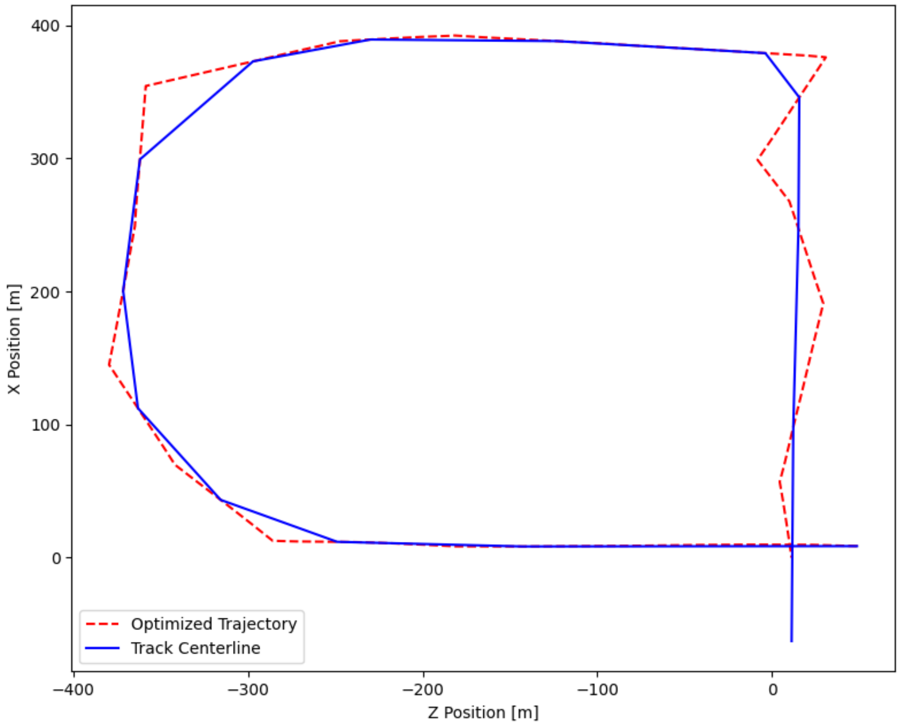

Figure 12: Optimized Trajectory from Non-Linear, Global Planner Program

4.3 Possible Avenues for Future Work

As mentioned at the beginning of this paper, Model Predictive Control is a widely used technique in control of vehicles, and the MPC developed by our team in simulation can have more applications on other platforms. For instance, it can be transferred to the RC car platform with slight modifications, and it also paves the way to more functions such as dynamic path planning. More details are discussed in the following sections.

4.3.1 Running MPC On RC Car Hardware

Due to the inconvenience caused by the pandemic and the limited hardware resources, most of the current works are based on simulation in software. The final objective of this project, which is to apply MPC on the RC car platform, is hard to achieve in the current situation. Therefore, one of the most important avenues of future work for this project will be transferring all the existing work to the RC car platform.

Fortunately, the simulation software, CARLA, used in this project is a highly developed software aimed for validation of autonomous driving systems. Physical environment can be well simulated in CARLA, and it makes the results of this project reliable and applicable to real-world systems. There are still some barriers for the transfer process. The MPC module is written to be workable in CARLA, and the vehicle parameters used at present are for the simulated vehicle. The MPC module will need to be modified to be able to work on the RC car hardware. Additional experimental tests will also need to be run in order to find out any other underlying problems when running MPC on the physical RC car.

4.3.2 Dynamic Path Planning Capabilities

Another possible future application for this project is in the development of dynamic path planning capability. Dynamic path planning is when the prediction of the path is continuously computed based on the physical parameters of the vehicle and the environment. Therefore it is a perfect application for the MPC dynamic model developed in this project. In order to modify the MPC to have the dynamic path planning capability, more environmental information needs to be introduced into the current program, including the track environment and information of other vehicles and obstacles in the track. More hardware computational resources will be needed to achieve expected functions such as passing other vehicles and dodging obstacles. This will be a longer process than transferring the MPC controller to the RC car platform because it will be a further development based on the current work.

References

[1] T. Litman, "Autonomous Vehicle Implementation Predictions", Victoria Transport Policy Institute, Victoria, British Columbia, rep., 2022.

[2] E. Boudette, "Despite High Hopes, Self-Driving Cars Are 'Way in the Future'" The New York Times, 17-Jul-2021. [Online]. Available: https://www.nytimes.com/2019/07/17/business/self-driving-autonomous-cars.html [Accessed: 13-Apr-2022].

[3] C. Metz, "The costly pursuit of self-driving cars continues on. and on. and on.," The New York Times, 24-May-2021. [Online]. Available: https://www.nytimes.com/2021/05/24/technology/self-driving-cars-wait.html. [Accessed: 13-Feb-2022].

[4] J. Thorne, "Cruise, Rivian raise billions to ready technology for the road," PitchBook, 19-Jan-2021. [Online]. Available: https://pitchbook.com/news/articles/cruise-rivian-raise-billions-to-ready-technology-for-the-road. [Accessed: 8-Apr-2022].

[5] A. Dosovitskiy, "CARLA: An Open Urban Driving Simulator," 1st Conference on Robot Learning, 2017. [Online]. Available: https://carla.org/. [Accessed: 8-Apr-2022].

[6] N. R. Ruchika, "Model Predictive Control: History and Development," International Journal of Engineering Trends and Technology, vol. 4, no. 6, pp. 2600–2602, Jun. 2013.

[7] F. Pucher, "Model predictive control," fjp.github.io, 07-Dec-2017. [Online]. Available: https://fjp.at/control/model/predictive/model-predictive-control/. [Accessed: 24-Jan-2022].

[8] J. M. Gonzales, "Planning and Control of Drift Maneuvers with the Berkeley Autonomous Race Car," dissertation, 2018.

[9] F. Borrelli, Predictive control for linear and hybrid systems. Cambridge: Cambridge University Press, 2017.

[10] Jason Kong et al. "Kinematic and dynamic vehicle models for autonomous driving control design". In: Intelligent Vehicles Symposium (IV), 2015 IEEE. IEEE. 2015, pp. 1094-1099.