Implementing Model Predictive Controller for Autonomous Vehicles

Ran Jing
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
rjing@wpi.edu

Abhishek Jain
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
ajain4@wpi.edu

Kavit Nilesh Shah
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA
kshah@wpi.edu

***Abstract*—Autonomous Vehicles have become one of the hottest topics in the robotics field in the past decade. Control of the vehicle is never a trivial problem. Traditional control methods have limitations on constraints formulation or trajectory tracking. Model Predictive Control (MPC) is a technique that can accurately track a given trajectory and easily add constraints. In this project, we focus on simulation about adaptive Model Predictive Control for autonomous vehicles. We design the control system, set up a simulation environment for multiple driving scenarios, implement adaptive MPC, and give an analysis of the experimental results.**

***Index Terms*—Model Predictive Control, Autonomous vehicle, Lane changing, Advanced control**

I. Introduction and Motivation

There have been rapid advancements in the field of autonomous vehicles and navigation over the past few years. Autonomous vehicles have their application is a wide variety of fields like scientific and planetary exploration, health technology, security, nuclear power industry, transportation and logistics, military operations like mine deactivation, surveillance, education, excavation, entertainment, agriculture (farm vehicles), housekeeping, mining and exploration, inspection and maintenance, completion of complex, dangerous and remote environment tasks, reduction of vehicle accidents on highways, etc.

The main components of a modern autonomous vehicle are localization, perception, and control. Autonomous Control is a challenging and vital component of autonomous navigation. Most of the applications of autonomous vehicles involve a fast dynamically changing environment. Hence, it is pertinent to have a robust and efficient control mechanism to respond to the fast-changing environment. Model Predictive Control(MPC) is one such technique that can be used to efficiently track a reference trajectory and navigate through cluttered environments. The basic idea of MPC is the use of a model to predict the output of a process over a future time horizon by obtaining a control sequence that minimizes an objective function [1]. An analogy can be made between the MPC and the act of driving a car [2]. The driver knows the desired reference trajectory for a finite horizon: his field of view of the road. Taking into account the characteristics of the car (a mental model of the car and speed limits, acceleration and maneuverability) and possible roadblocks (like holes, intersections and other cars), he will decide what action to take (increase or decrease speed, turning direction to one side or the other) so that the desired trajectory is traversed. This control action is then applied for a short time and the procedure is repeated for the next control action now with the field of view updated.

We aim to implement the MPC technique for trajectory tracking on a simulated autonomous vehicle model based on Dubin's car model in MATLAB. We simulate an environment with mulitple obstacles for our vehicle model and then design an MPC controller to navigate our vehicle in the environment.

II. Preliminaries

The simulation of an autonomous vehicle requires modeling of the vehicle dynamics using a mathematical system of equations. In the initial work, kinematic bicycle models were used to simulate the dynamics of vehicles. In [4], a comparison has been made between kinematic and dynamic system model to be simulated with an MPC controller. For this project, we plan on using a Dubin's car model for simulating the vehicle dynamics [5]. The equations (1),(2), (3) & (4) represent Dubin's car model of a vehicle.

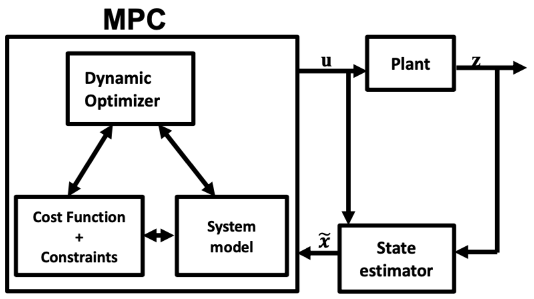

Fig. 1: MPC Architecture [3].

$$\dot{x} = v \cos\theta \tag{1}$$

$$\dot{y} = v \sin\theta \tag{2}$$

$$\theta = \frac{v}{L} \tan\phi \tag{3}$$

$$\dot{v} = a \tag{4}$$

The system of equations describing vehicle dynamics as described above are non-linear in nature. To simulate this system we plan to use an adaptive Model Predictive Control for controlling the vehicle velocity and steering angle because of its ability to handle nonlinearities and disturbances in the system and ability to optimize the next immediate input while considering future trajectories and future inputs.

A Model Predictive Controller predicts future positions of the system based on the system model and generates an optimized sequence of control input over the horizon (a definite time span) from the current state. From this sequence, input at the next immediate time-step is fed to the controller and the controller, based on this input, drives the system from the current state to another state. Based on this new state, the controller again generates a new optimized sequence of control inputs over the horizon and feeds the next immediate input to the controller. By application of this repeated process, an MPC controller can drive any linear or non-linear system, facing multiple constraints, from any initial state to the desired state (a set point or a trajectory), even in the presence of disturbances in the system. This renders the use of Model Predictive Controller suitable for self-driving cars.

The Fig 1 shows the basic architecture of the MPC Controller. The MPC block has 3 components: The System Model, Cost-function with Constraints, and Dynamic Optimizer. The system model represents the kinematic and dynamic model of the car system. The Cost-function and constraints enlist the objective function which needs to be minimized subject to various constraints. The dynamic optimizer integrated both system model and objective function under the constraints to generate the control input which is fed to the plant to generate the system output.

The Controller will be used to track a reference trajectory that will be generated using a trajectory generator. Error based on the current trajectory and the reference trajectory will be fed to the MPC block which will further be processed by its components to generate the next control input. The challenge is to change the reference trajectory by introducing random disturbances so as to adapt the controller to track the changing trajectories and quickly converge to the optima.

MATLAB and its various toolboxes are used to design, define, and implement the controller.

III. Problem Statement

In order to control autonomous vehicles, we need to have a test environment, dynamic model of the vehicle, trajectory generation algorithm, controller, and actuator. For our project, we focus on the control problem and try to simplify other aspects of the system.

First, since the school is temporarily closed, we have no chance to test our methods on real robots. We will test the controller in a simulation environment based on Automated Driving Toolbox (for lane changing task) or Vehicle Dynamic Blockset (for open space racing task through traffic cones).

Second, we will use a bicycle model or car-like model for the system dynamics. A bicycle model can be used to represent a four-wheel vehicle. Two front wheels are lumped into one single wheel at the midpoint of two wheels and two back wheels are lumped in the same way. In the rear-wheel model, the rear wheels handle longitudinal driving forces and front wheels handle steering forces.

Third, since we are trying to focus on a control problem, trajectory optimization is not considered in the first place. Our main goal is to minimize the error when the controller tries to follow a specific trajectory. We first set the vehicle a fixed trajectory(e.g. a trajectory for lane changing) to test the controller and improve the performance. Also, We will look to implement advanced trajectory generation to help the vehicle decide when and how to switch to a different lane or avoid an obstacle after solving the control problem.

For MPC itself, the main problem is to choose a suitable dynamic optimizer so that we can get a precise prediction for future states and find a global optimal control sequence for each time step. We plan to test both linear and non-linear MPC to find a better solution. Non-linear MPC is more precise on modeling but needs more calculation and is more likely to fall into a local minimum. We also build a dynamic model updating function for adaptive MPC that can involve the influence of speeding changing of the vehicle.

In general, in this project, we implement a non-linear adaptive MPC controller to minimize the error in following a given trajectory on autonomous vehicles. As our desired goal we implement a simple trajectory generator for tasks like lane changing and obstacle avoidance on the road to achieve autonomous driving in specialized scenarios.

IV. Main Methods

The Model Predictive Control is the most prominent control technique for autonomous vehicles and self-driving cars. This possibly can be attributed to a lot of factors including compliance with linear as well as non-linear systems, stability, and robustness to disturbances and offsets in the system and can track set points as well as trajectories accurately. The MPC controller proposed in this project works on trajectory tracking with combined control of velocity as well as the steering angle. Equation (5) shows the system state vector represented by X and control inputs represented by U. The system state consists of x, y, θ, v and the control inputs consists of acceleration or throttle and δ.

$$\vec{X} = \begin{bmatrix} x \\ y \\ \theta \\ v \end{bmatrix} \vec{U} = \begin{bmatrix} a \\ \delta \end{bmatrix} \tag{5}$$

In this project, we build our own implementation of the MPC controller following the architecture described in [10]. The following sections explain the implementation methodology in detail.

A. Linearization and Discretization

As described by equations (1), (2), (3) & (4), the system model taken into consideration is non-linear in nature. There are many different MPC architectures that can work with non-linear plant models, but the MPC architecture followed by us works on a linear system. Thus, in order to convert the non-linear plant model into a linear system, Jacobian linearization was employed which computes the partial derivatives of the system dynamics model w.r.t. state vector to obtain the A matrix and partial derivatives w.r.t. the input vector to obtain the B matrix as shown in equation (6). The linearized system model is then discretized to obtain new system states as a linear function of the current system state [10]. Once the system model has been linearized as well as discretized, the next step marks the start of the iterative process for computing the controlled inputs for the MPC controller. All the steps described below are computed online in each iteration for the controller.

$$\dot{x} = \frac{\partial f(x,u)}{\partial x} x + \frac{\partial f(x,u)}{\partial u} u \tag{6}$$

B. State Estimator

At each point, first a set of control inputs is computed using which, the state estimator computes the new state based on which new control inputs are computed and the process repeats iteratively for the prediction horizon. An important point here being, that the control horizon being a subset of the prediction horizon, could be of same or lesser length that the prediction horizon. Thus the control inputs are being computed at each iteration till the iterator reaches the control horizon, after which the control inputs are set to some constant value till the end of prediction horizon. However the state estimator estimates the state of the system till the end of prediction horizon. This estimated of system states and control inputs over prediction horizon are used for computing the A and B matrices for the cost function of the optimizer which will be discussed in the next step.

C. State Optimization

Instead of considering and trying to track the entire trajectory at one computation like LQR, the MPC controller achieves the trajectory tracking control in moving window segments of the entire trajectory. At each time step, the controller considers the trajectories of a finite prediction horizon of the future time steps from current time instance and tries to optimize the set of control inputs obtained from previous step which could help the system to track the desired trajectory while satisfying all the constraints. For realistic trajectory tracking, constraints on x and y positions, heading directions, velocity, steering angle, and rate of change of velocity have been imposed. Additional constraints have been imposed on the system to minimize the rate of change of control inputs to avoid abrupt and sudden changes in the system inputs. The equation (7) is the cost function that is optimized at every time step along with all the constraints being imposed on the same. As seen from the equation, the Q matrix penalize the errors in system states and the R matrix penalize the system inputs. The optimizer minimizes the cost function J to generate an optimized set of control inputs which would drive the system from the current state to track the trajectory accurately. In MATLAB™, the mpcActiveSolver has been used for the purpose of optimization.

$$\min_{U_{\mathrm{i}}} \quad J = U^T H U + f^T U \tag{7}$$

$$\begin{aligned}
&\text{s.t.}\\
&x_{\mathrm{k+1}} - (x_{\mathrm{k}} + v\cos\theta \mathrm{d}t) = 1 \qquad (a)\\
&y_{\mathrm{k+1}} - (y_{\mathrm{k}} + v\sin\theta \mathrm{d}t) = 1 \qquad (b)\\
&\theta_{\mathrm{k+1}} - (\theta_{\mathrm{k}} + \tfrac{v}{w_{\mathrm{b}}}\tan\delta \mathrm{d}t) = 1 \qquad (c)\\
&v_{\mathrm{k+1}} - (v_{\mathrm{k}} + a\mathrm{d}t) = 1 \qquad (d)\\
\\
&0m \le x \le 500m \qquad (e)\\
&10m \le y \le 160m \qquad (f)\\
&-\pi \le \theta \le \pi \qquad (g)\\
&-20m/s \le v \le 40m/s \qquad (h)\\
&-2m/s^2 \le a \le 2m/s^2 \qquad (i)\\
&-\pi/4 \le \delta \le \pi/4 \qquad (j)\\
&where\\
&H = (B^T Q B + R)\\
&f = 2B^T Q A x
\end{aligned}$$

D. System State Evolution

On obtaining the optimized set of control inputs from the optimizer, the next step involves feeding the control input from the next immediate time step into this module. It uses a model of the system to simulate the system dynamics and evaluates the new system state to which the system evolves from the current state given the control input.

On reaching the new state, the iteration for next time instance begins from step B of the methodology and system again goes through the same process iterating and optimizing the control inputs on the fly. This is one of the reasons for the MPC controller being able to overcome non-linearities and disturbances very smoothly resulting in a robust and an accurate trajectory tracking.

V. Experiments and Results

In this work, in order to simulate the controller performance with visualization, we set up a road system model in Automated Driving Toolbox. It provides us a good real-time display simulation environment. We can easily add actors like vehicles and obstacles, roads and way-points. As shown in Fig.3 (a), the vehicle is represented by a blue box and the obstacle is represented by the red box located on the lane of the vehicle. For the perception of the environment, you can add a camera or LiDAR to your vehicles. For our task, we add one camera in front of the vehicle as shown in Fig.3(b). The camera holds a visible distance of 50 meters.

For tracking trajectory task, we first set a trajectory that achieves obstacle avoidance in the Automated Driving Toolbox scenario designer by adding a series of way-points. Then we manage to output the x-y trajectory to a MATLAB file(as shown in Fig.4) and get θ and v trajectory based on that.

After that, we test our own MPC controller from scratch through output trajectory files. We also test the controller we build in SIMULINK with scenario reader to Automated Driving Toolbox to get a real-time visualization.

In Fig.5 and Fig.6, we can see that experimental results tracking the given trajectory as Fig.4 with our own MPC controller. We simulate the controller for 350 time steps. From sub-figure (a),(b), and (c), you can find the controller performs fast reaction speed and small tracking error in this specific task. We set a constant velocity of 15 m/s for the vehicle and it takes the controller around 50 time steps to reach the desired speed smoothly as recorded in sub-figure(d). Also, it takes the vehicle around 15 time steps to diminish a relatively larger initial orientation error of 0.8 rads.

In Fig.8 and Fig.9, the experimental results tracking the circular trajectory with our own MPC controller are shown. We simulate the controller for 100 time steps. From sub-figure (a), (b), and (c), you can find the controller performance in this specific task. The constant velocity is set to be 15 m/s for the vehicle. It still takes the controller around 50 time steps to reach the desired speed smoothly as recorded in sub-figure(d). For this task, the tracking error is larger than the previous task. I think this might because the steering angle for controlling this specific task is always large during the whole process. To be mentioned, we observe that for both tasks, the error of tracking along the x-axis is relatively larger than the error along the y-axis and θ angle. We haven't figure out the reason why this phenomenon appears. It might cause by the weight of states in the design of our MPC model.

Here we would like to have a brief analysis of the influence of the prediction horizon. In Fig.2, we run the simulation multiple times with a fixed control horizon of 5 and prediction horizon to be 5, 10, and

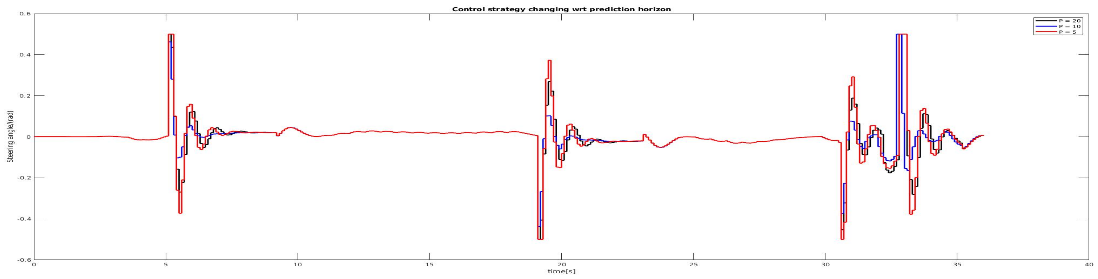

Fig. 2: The steering angles with respect to time steps for different prediction horizons.

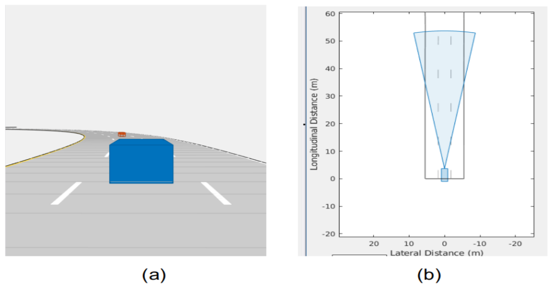

Fig. 3: Building scenarios in Automated Driving Toolbox.

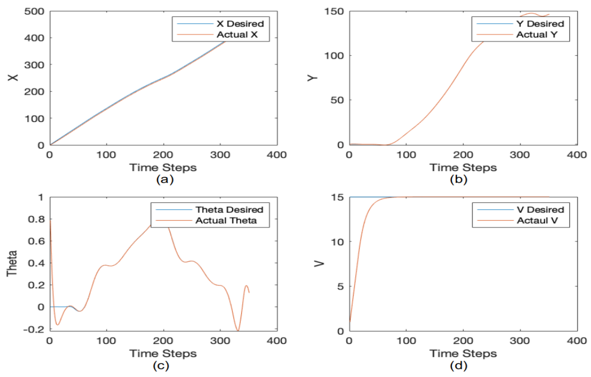

Fig. 5: Autual and reference trajectory for system state x in (a), y in (b), θ in (c), v in (d).

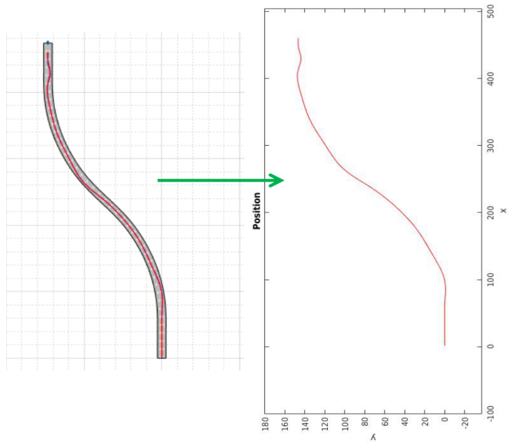

Fig. 4: Reference trajectory generation for obstacle avoidance.

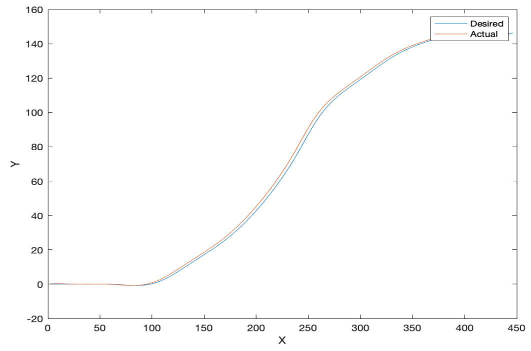

Fig. 6: Autual and reference trajectory tracking an obstacle avoidance task

20 correspondingly. From the legend, P is set to be the prediction horizon, where red, blue and black line represents P = 5, 10, 20 respectively. As the prediction horizon changes from 5 to 20, The control strategy becomes less and less aggressive. The scenario where P = 5 gives a much higher control effort than other prediction horizons. Because the less information the robots can get, it less prediction about the future state can be made and the controller needs to hold a relatively larger control effort to diminish the error.

We also test the MPC controller from MPC toolbox with real-time visualization in the simulation environment. It requires you to read all information from Automated Driving Toolbox using Driving Scenario Reader

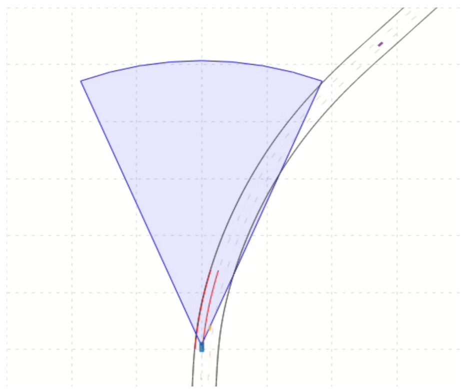

Fig. 7: Real-time simulation GUI with Automated Driving Toolbox and SIMULINK.

in SIMULINK and extra calculation for data format transitions in SIMULINK. An example result of the GUI to the visualization of real-time running result is shown in Fig.7. More details can be found in the video here.

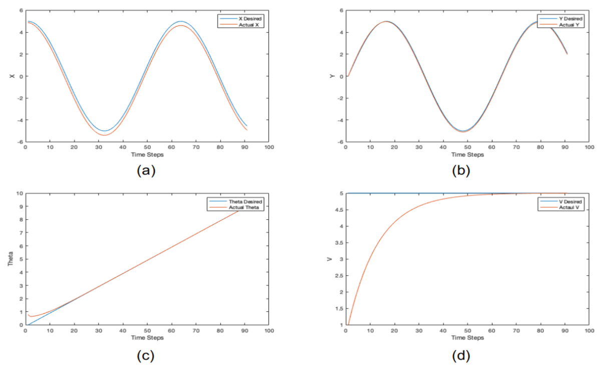

Fig. 8: Actual and reference trajectory for system state x in (a), y in (b), θ in (c), v in (d).

VI. Conclusion and Discussion

In this work, we study the principle of Model Predictive Control, set up a simulation environment by using Automated Driving Toolbox and SIMULINK, learn the structure of MPC Toolbox in MATLAB and build our own MPC controller from scratch. For experiments, we test the controller from MPC Toolbox with real-time visualization in a simulation environment. Also, we generate multiple trajectories and implement them on our own MPC controller. The analysis of experimental results and system parameters such as the prediction horizon is given.

A lot of work can be expanded from current results. The tracking error might be decreased with better architecture design. We should also improve the computational efficiency. Detailed analysis of the influence and relationship between the prediction horizon and the control horizon can be done. After that, automatically selecting the prediction horizon and the control horizon for different driving scenarios can be a good topic. As potential future work, we might be able to improve this algorithm from the following perspectives:

1) Build our own optimization function for our MPC controller.
2) Improve the computational efficiency and align our own MPC controller with Automated Driving Toolbox to check the 3-D visualization result in real-time.
3) Achieve automatic parameter learning and selecting for different driving scenarios.

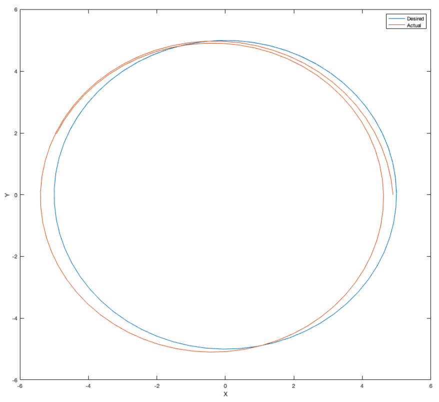

Fig. 9: Autual and reference trajectory tracking a circular road.

References

[1] E. F. Camacho and C. Bordons. Model Predictive Control (Springer-Verlag, 2nded. 2004).
[2] C. Bordons, E. F. Camacho. Model Predictive Control. Advanced Textbooks in Control and Signal Processing (Springer-Verlag,London, 1999).
[3] Oyelere, Solomon Sunday. ”The application of model predictive control (MPC) to fast systems such as autonomous ground vehicles (AGV).” IOSR J. Comput. Eng.(IOSR-JCE) 16.3 (2014): 27-37.
[4] J. Kong, M. Pfeiffer, G. Schildbach and F. Borrelli, ”Kinematic and dynamic vehicle models for autonomous driving control design,” 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, 2015, pp. 1094-1099.
[5] Du, Xinxin, Kyaw Ko Ko Htet, and Kok Kiong Tan. ”Development of a genetic-algorithm-based nonlinear model predictive control scheme on velocity and steering of autonomous vehicles.” IEEE Transactions on Industrial Electronics 63.11 (2016): 6970-6977.

[6] L. Cunjia, C. Wen-Hua, and J. Andrews. Optimisation based control framework for autonomous vehicles: Algorithm and experiment. In Mechatronics and Automation, pp. 1030-1035, 2010.

[7] T. Keviczky, P. Falcone, F. Borrelli, J. Asgari, and D. Hrovat. Predictive control approach to autonomous vehicle steering. In American Control Conference, 2006.

[8] Falcone, Paolo, et al. "A model predictive control approach for combined braking and steering in autonomous vehicles." 2007 Mediterranean Conference on Control & Automation. IEEE, 2007.

[9] Belvén, Pontus. "Implementation of Model PredictiveControl for Path Following with the KTH Research Concept Vehicle." (2015).

[10] https://en.wikipedia.org/wiki/Discretization