Adaptive Cruise Control with Lane Change Algorithm

Yizhou Lu, Arpita Petkar and Snehal Prabhudesai

***Abstract*— This paper attempts to enhance the adaptive cruise control (ACC) algorithm with the additional functionality of change of trajectory impelmented using model predictive control(MPC). The non-linear bicycle model is chosen as a candidate model for the implementation of the proposed algorithm. The algorithm operates in four distinct stages- detection of a lead car, change of trajectory, passing the lead car and reverting to the original trajectory. The validation of the proposed algorithm is done using MATLAB.The results indicate a seamless transition between the two distinct algorithms defined previously, and thus proves to be ideal for real-life scenarios as well.**

Keywords - Model Predictive Control, Adaptive Cruise Control, autonomous vehicles, trajectory change algorithm

I. INTRODUCTION

In autonomous driving, adaptive cruise control is a broadly accepted and highly implemented idea.In the past, adaptive cruise control has been implemented for road vehicles by automatically adjusting the vehicle speed to maintain a safe distance from vehicles ahead. Another feature being worked on right now in self-driving cars is the lane change when a lead car is detected.The lane change can either mean driving around a parked vehicle or bypassing a slower car on highways or even avoiding collisions. Hence the idea behind the work presented is to merge these two studies and develop an algorithm for a car which detects a lead car based on a pre-determined distance, tries to move around the lead car by accelerating, gets back on the lane and maintains a certain constant velocity on the road until another car is detected.

The remainder of this report is organized as follows: Sec II details the problem statement and the assumptions made regarding this problem to arrive at a solution.Sec III defines the system with the details regarding vehicle dynamics model for the car, determining the conditions when to make the switch to the other lane, the trajectory change definition and also the model predictive control method used to optimize the cost function according to certain constraints described in detail.Sec IV lists the observations and the results achieved after the successful run of the algorithm on MATLAB 2018b software and finally Sec V details the conclusions drawn from this work followed by Appendix and references.

II. PROBLEM DESCRIPTION

A. Problem Statement

This problem environment basically has a ego car which starts at a certain constant velocity and is running along a pre-determined path.There are two other vehicles, car A and car B, on the path which are travelling in the same lane as the ego car and are travelling at lower but constant velocities.The ego car is expected to detect those cars and move around those cars by entering the other lane on the side and then get back to the original lane after passing the car.The problem would be termed as a success after the ego car is able to do the following: (1) Detect a lead car based on the distance between the cars A and B and the ego car. (2) Change the lane with minimum overshoot and stay in the lane after change. (3) Detect the car in order to ensure that the ego car has passed the lead car (4) Change back to the previous lane again and passing the lead car

B. Design Specification

The Design specification are listed below: (1) Overshoot has to be less than 5% when changing the lane (2) The oscillation has to be less than 1% during cruising (3) Rise time for the velocity increase or decrease has to be less than 5 seconds (4) Settling time for the velocity to reach steady state value has to be less than 10 seconds.

C. Assumptions

For a successful demonstration of this problem, numerous assumptions were made for this problem statement as follows:

(1) Car Dynamics : The car model was considered point mass. Its mass was used mainly to convert force to acceleration. The ego car always started with an initial velocity of 10m/s. The ego car speed was always considered to be greater than the speed of the other two cars.The cars A and B were travelling on the same path,in the same direction and in the same lane with different but constant speeds and their locations were always known with respect to the path. The cars A and B were always considered to be the lead cars, or in other words, the distance between the ego car and these two cars was always monitored irrespective of whether these cars were too close to the ego car or not.

(2) Path and reference body frame : The path was basically composed of a set of 1230 points spanned over a distance of 3780m. Since the car was considered as a point, there existed just one reference frame which is described by the x-y co-ordinate frame.The car was always moving on the right side lane and it moved into the left side lane only when it had to bypass the lead car.

(3) Lane and Trajectory change : In order to avoid overshoot, the lane change to another lane was achieved in 9 points with manually-defining points for the ego car to follow.

III. SYSTEM DESCRIPTION

A. Track Description

The track given had a center lane, left boundary, right boundary and head angle along the path. Each data set was

consisted of 246 points. In order to make the track more precise, it was poly-fitted with splines to add more points and construct more lines. The end result of the track includes left boundary, right boundary, center line, left lane, right lane, and head angle. Each data set was consisted of 1230 points. The path that the ego car as well as the other cars A and B follow was basically a set of points in the middle of the right side of the lane.

B. Vehicle Dynamics

The vehicle dynamics were obtained from the non-linear bicycle model as described:

$$\dot{u} = \frac{(F_d - bu)}{m} \tag{1}$$

$$\dot{x} = \left(\frac{-L_1}{L_2}\sin\delta\sin\psi + \cos\delta\cos\psi\right)u \tag{2}$$

$$\dot{y} = \left(\frac{L_2}{L_1}\sin\delta\cos\psi + \cos\delta\sin\psi\right)u \tag{3}$$

$$\dot{\psi} = \left(\frac{1}{L_1}\sin\delta\right)u \tag{4}$$

The above described vehicles dynamics system was solved as a discretized system. The above equations were solved by sampling at a time interval dt=0.01 seconds. The differential equations were converted into difference equations and solved in MATLAB by using a discrete-time integrator. The input values were held constant for a given time interval i.e. u(k) value was held constant till the next sample arrives.

C. Switch System

The switch system was activated when the vehicle needed to change to the left lane to bypass the lead car at front. When the distance between the ego car and the lead car was less than 50 meters, a flag would be turned on to indicate that the trajectory needed to be changed at next iteration. The system was also activated after the ego car had done passing and needed to move back to the right lane. When the distance was larger than 5 meters, another flag would be turned on for the similar purpose. Due to the approximation of the car as a point mass, the distance here only served as a guideline, the real values should be modified based on the car length. In order to keep track of the distance between cars, a look-up table was generated beforehand for each point on the track. Only for the purpose of this section, the location of each cars was approximated by the nearest point on the track. The difference between two points served as a good prediction for the difference between two cars along the path.

D. Reference Trajectory Modification

Once a flag was turned on, the reference trajectory would be changed for the rest of the path. In order to reduce overshoot of the car position during lane changing, the next 9 points on the reference trajectory in front of the ego car were modified to fit a smooth inverse tangent function. The number of points was chosen based on trials of simulations. This was done by pushing the points from the current lane to the other lane by a sequence of percentage. After that, all the points were pushed to the other lane for the rest the trajectory.

E. Model Predictive Control

The model predictive control was used to predict how the system was going to behave in the next couple time steps. In this project, the time step window was chosen to be 10. Within those 10 time steps, 44 system states and 20 inputs were required to be calculated and that made a total of 64 decision variables. In order to further explain the procedure of MPC used in this project, an example of matrices used in calculation was given in the next couple sections. The example given used a time step window of 2, which included 16 decision variables.

1) Cost Function: The cost function was used to penalize the states and input deviation from 0. Here 0 stood for the linearization point. Every iteration the system was linearized, the origin of the state vector was moved to that linearized point and became 0.

$$Q \rightarrow x\epsilon\mathbb{R}^{4\times4} \tag{5}$$

$$R \rightarrow u\epsilon\mathbb{R}^{2\times2} \tag{6}$$

$$H = \begin{bmatrix} Q & & & & \\ & Q & & & \\ & & Q & & \\ & & & R & \\ & & & & R \end{bmatrix} \tag{7}$$

2) Reference State and Reference Input: The reference trajectory, which includes position x, position y and heading angle was choosing to be the right lane since it was the default line for the ego car. The reference speed was chosen to be 10 m/s. The reference steering angle was chosen to be 0°, considering each point on the reference trajectory was very close to each other and the steering angle can be assumed to be small. The reference force was chosen to 1000 N, calculated based on the reference speed and the first equation of car dynamics, which described acceleration.

3) Inequality Constraint: In this MPC problem, the inequality constraint was implemented in terms of input constraints.The input constraints comprised of limits on the steering angle(δ) and the thrust force Fd.The limits considered for the steering angle are -0.5 to +0.5 radians and the limits for the thrust are -5000N to +2500N. The negative thrust indicates the braking force and this negative region of thrust was never used in this problem.

The MPC algorithm solved the problem at every instant considering the first control input as u(k) for the iteration and calculated where this input lied as far as the inequality constraints were considered. Suppose the u(k)=1000N , then the algorithm calculated the buffer limit the u(k) could have according to the limits. Like for 1000N, the buffer would be (-5000-1000)= -6000N on the lower side and (2500-1000) = +1500N on the upper side. If in any case the u(k) calculated in order to solve the cost function exceeded the bounds on thrust, then the MPC would break down the thrust in parts

and would achieve the constraints over next iterations.For eg, if the thrust calculated was 3000N , then the MPC would make u(k)=2000N for the first iteration and then would try to reach 3000N in steps over the next few iterations.Hence, MPC is considered as a better method to use to solve cost functions because of it's ability to deal with hard constraints.

4) Equality Constraint:

$$n_{pred} = 2 \tag{8}$$

$$n_{dec} = 4\,(2+1) + 2\,(2) = 16 \tag{9}$$

$$\dot{x} = f\,(x, u) \tag{10}$$

$$A_k = \frac{\partial f}{\partial x}\Big|_k \qquad B_k = \frac{\partial f}{\partial u}\Big|_k \tag{11}$$

$$x_{k+1} = A_k x_k + B_k u_k \tag{12}$$

$$\begin{bmatrix} I_{4\times4} & 0 & 0 & 0 & 0 \\ A_k & -I_{4\times4} & 0 & B_k & 0 \\ 0 & A_{k+1} & -I_{4\times4} & 0 & B_{k+1} \end{bmatrix} \times \begin{bmatrix} x_k \\ x_{k+1} \\ x_{k+2} \\ u_k \\ u_{k+1} \end{bmatrix} = \begin{bmatrix} Y_{error} \\ 0_{4\times1} \\ 0_{4\times1} \end{bmatrix} \tag{13}$$

$$A_k x_k - x_{k+1} + B_k u_k = 0 \tag{14}$$

$$A_{k+1} x_{k+1} - x_{k+2} B_{k+1} u_{k+1} = 0 \tag{15}$$

The example of equality constraint again had a time step window of 2, and a total of 16 decision variables to be calculated, with reference specified in part 2, and constraints specified in part 3.

The equality constraints were derived to describe the linearized dynamics. The linearized A and B matrix at each iteration were concatenated into matrix Aeq. The decision variables were augmented into a vector. Those two matrices were designed in such way that the their product equaled to a zero vector, beq. This forced MPC to satisfy the dynamics of vehicle when it was calculating states and inputs in the next time step window. In additional to that, the difference between the actual state and reference state, denoted as Yerror, was put into the top of beq as an initial condition for the equality constraint.

Yerror was calculated at the beginning of MPC algorithm. After the decision variables were calculated, only the first set of inputs were used to simulate the car dynamics for 0.01 second, specified above. The new state of the vehicle was used to calculate Yerror for the next iteration.

IV. OBSERVATIONS AND RESULTS

The following results were observed on running the algorithm on MATLAB:

A. Detection Mode

When the lead car was detected, the trajectory changing algorithm was initialized.

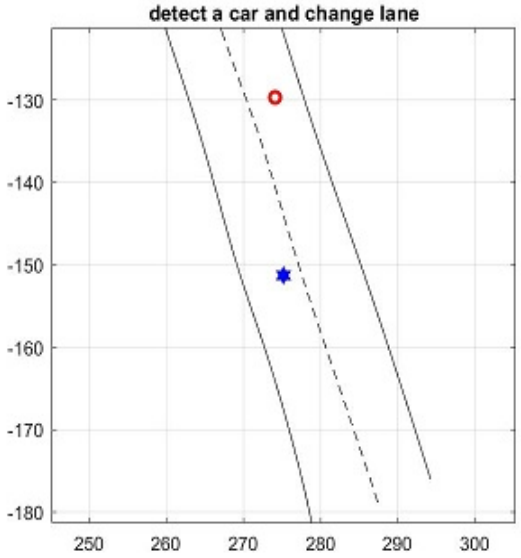


Fig. 1. Detection of lead car by ego car.

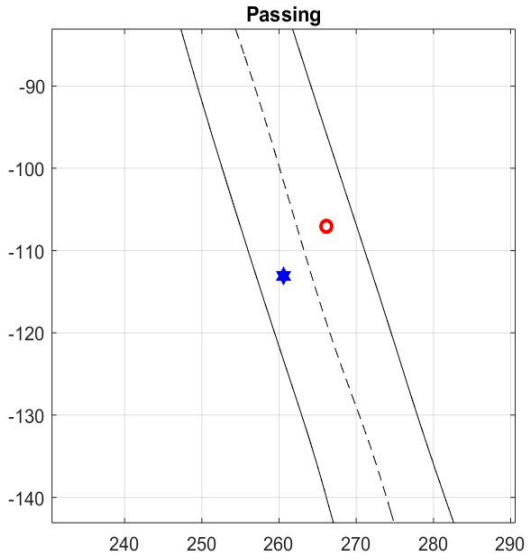


Fig. 2. Ego car passing Lead Car A in left lane.

B. Passing Lead Car Mode

When the ego car was passing the lead car,the speed of the ego car increased to minimize the time consumption on the other lane.

C. Go Back Mode

After the lead car was passed by at least 5 meters, the ego car changed its trajectory to its right lane. This was done in order to ensure the go car to have a safe distance in front of the lead car when it gets back to the original right lane.

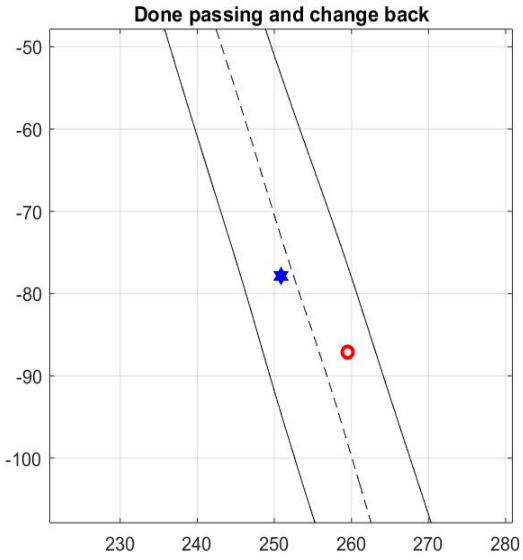


Fig. 3. Ego car changes lane to original one.

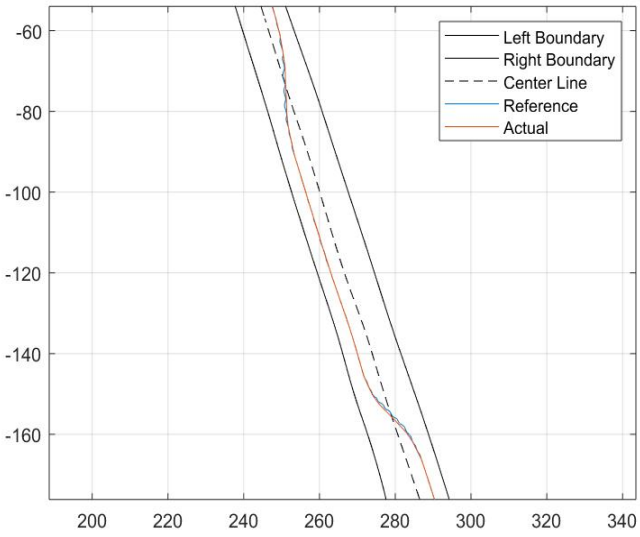


Fig. 5. Position profile of car interacting with Lead Car A.

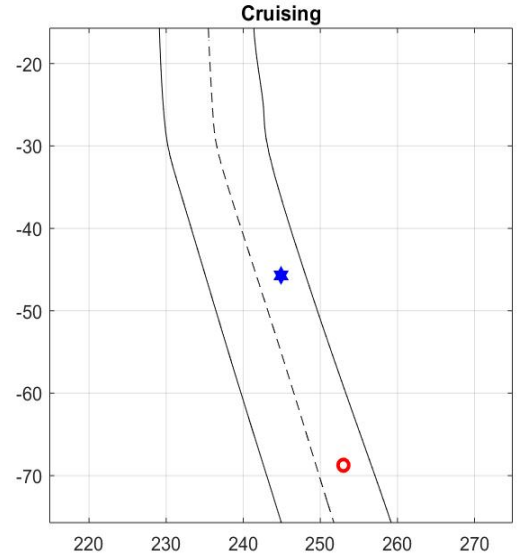


Fig. 4. Ego car in cruise control mode.

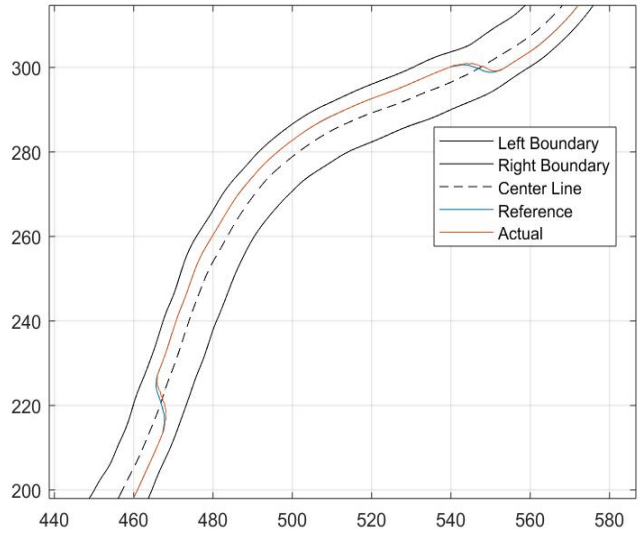


Fig. 6. Position profile of car interacting with Lead Car B.

D. Cruise Control Mode

This mode was entered into by the ego car under two circumstances. First, when the ego car had passed Car A and the Car B was very far away from the ego car. Secondly, when there was no other car in front of the ego car, and the ego car was the new lead car.

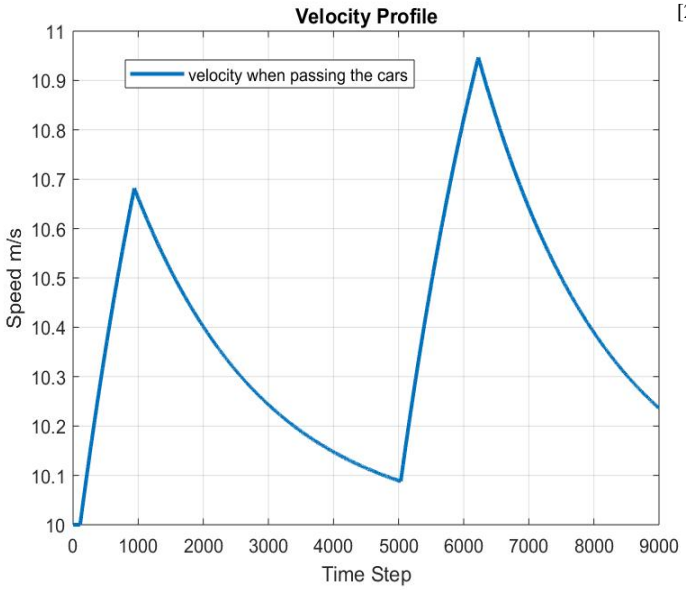
E. Position Profile

In the position profile, the trajectory followed by the ego car when it passes the lead car can be seen. The ego car initially moved into the lane on the left, and increased it's velocity so that it passes the lead car that was in the other lane. Once the position of the ego car was sufficiently far away from the lead car, it changed the lane back and then was in front of the lead car.

The position profile met the required design specification listed in Section II Part B. There were minimum oscillations occurrence during cruising stage. The occurred overshoot was less than 0.1% when the car was changing the lane.The results are shown in Fig 5 and Fig 6.

F. Velocity Profile

The velocity profile was obtained for both cars A and B. The velocity increased when the passing was initialized, and it decreased when the ego car was done with passing and changed the lane back to the original lane. The velocity



had a huge rise time to reach the desired velocity. We have presented this as a proof of concept and were thus not highly concerned about the high rise time.

The velocity profile did not reach the design specification due to MPC design constraints. MPC ensured velocity would reach the steady state value at the end of the algorithm, therefore, the rising time was huge. However, the rising time and settling time of the velocity came at a lower priority than the position requirement.

V. CONCLUSIONS

In this work, adaptive cruise control with Lane Change algorithm was formulated as a model predictive control problem to detect lead vehicles and change the lane to move around the lead car and get back on the lane resuming the velocity mode for adaptive cruise control. This work incorporated specific constraints on the inputs as well as on the linearized dynamics to ensure an optimal solution which avoided running into the lead vehicle and avoided crossing the outer boundaries.

Future work can be focused on expanding the context to the environment with roads having the traffic flow in both the directions and investigating alternative problem formulations such as trying to generate a universal algorithm for the lane change trajectory,considering a different body frame for the car and improving the MPC algorithm to reach the design specifications for velocity profile by dividing the trajectory into segments to ensure the velocity will reach designed value at the end of one segment.

APPENDIX

Github Repository: `https://github.com/YizhouLu/ME561_Project`

REFERENCES

[1] EECS 461 Final Project Fall 2018 Adaptive Cruise Control by James Freudenberg, University of Michigan.

[2] ME599 Self Driving Cars Final Project by Ram Vasudevan, University of Michigan.