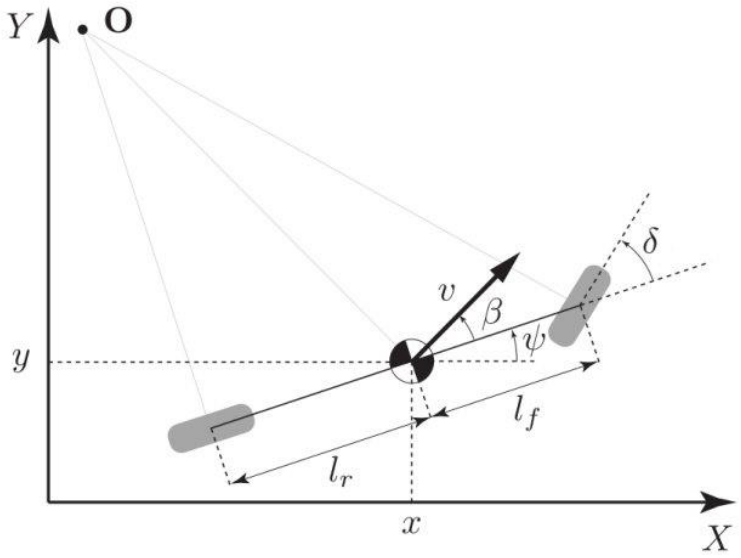
Project: 5 - Model Predictive Control (MPC) Implementation

This is the last project of the term 2, where the model predictive controller is implemented and tested with the Udacity simulator. The MPC controller is more efficient than the PID controller due to its ability to deal with the latency or timing issues by looking into the future. The purpose of this report is to describe various rubric points for this project.

The Model

Objective: Student describes the model in detail. This includes the state, actuator and the update equations.

The MPC controller works by estimating the states or state variables as a result of actuation and minimizing the errors by using the update equations as a source for future commands for the actuators. Sometimes, the errors are also included in the state vector as in the current implementation. The entities that are included in the state are position (x and y components as a flying car is not assumed at the moment), velocity of the car (v), orientation of the car(ψ), cross track error (cte) which is the displacement of car from the center of the road and orientation error(eψ or epsi). Utilizing the bicycle model introduced in the lesson, these states are illustrated in the following figure where the first four states are shown.



MPC controller algorithm is based on numerically solving an optimization problem at each step to predict the future state. In this project, we are using the ***Ipopt*** library that is a widely used/acknowledged C++ library for numerically solving the optimization problems. The optimizer's output is used to compute the error or cost function. The close loop operation continues until the cost reaches the minimum desired levels.

The state, actuator and the update equations that are employed in the project are mentioned below.

State	[x, y, v, psi, cte, epsi]
Actuator	The two actuators utilized for controlling the car are throttle/brakes and steering wheel that control the acceleration (**a**) and steering angle (**δ**) of the car respectively. [δ,a]
Update Equations	The following equations are used at each time step to update the above states and estimate the errors.
x	$x_{t+1} = x_t + v_t * \cos(\psi) * dt$
y	$x_{t+1} = x_t + v_t * \sin(\psi) * dt$
v	$v_{t+1} = v_t + a_t * dt$
psi	$\psi_{t+1} = \psi_t + (\left({}^{v_t}/_{L_f}\right) * \delta_t * dt)$ where L_f is the distance, in meters, between the front of the vehicle and its center of gravity, this is a purely physical parameter and varies from a car to car provided they are physically dissimilar (See the previous figure), for instance Lf is higher for Lamborghinis and smaller for mini coopers.
cte	$cte_{t+1} = cte_t + v_t * \sin(e\psi_t) * dt$
epsi($e\psi$)	$e\psi_{t+1} = e\psi_t + (\left({}^{v_t}/_{L_f}\right) * \delta_t * dt)$

Timestep Length and Elapsed Duration (N & dt)

Objective: Student discusses the reasoning behind the chosen N (time step length) and dt(elapsed duration between time steps) values. Additionally, the student details the previous values tried.

The N is the number of time steps that an MPC model can predict in future, and dt is the length of each timestep. So, together N*dt gives the duration in the future the controller can predict. For instance, if there are 10 steps (N = 10) with time difference between each time step is 0.1s (dt = 0.1), it signifies that this MPC controller can predict states for 1 (=10*0.1) second ahead of current time.

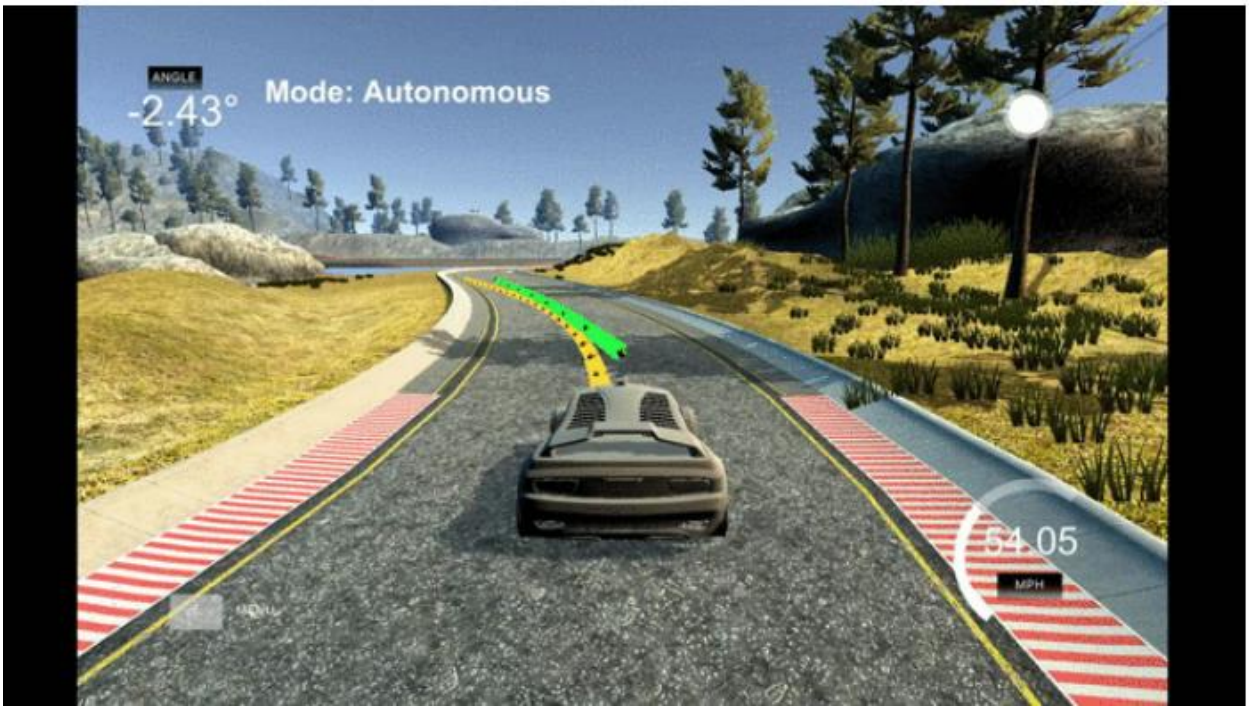
The N and dt are two very important design parameters in an MPC controller, where N tells the how much ahead in future we want to predict and dt shows how frequently the states need to be estimated and updated.

The observation of how a car moves along the track was the only criterion to choose the selected values of N and dt. So, the parameters were chosen in such a way that their product i.e. N*dt should neither be too large nor too small.

Polynomial Fitting and MPC Preprocessing

Objective: A polynomial is fitted to waypoints.

A third order polynomial is fitted here using the *polyfit()* function, and the corresponding four coefficients of the polynomial are computed using the given ***lake_track_waypoints.csv*** files. The fitted polynomial is shown by the yellow line in the simulator and the *x* and *y* coordinates of the MPC trajectory by the green line as shown in the figure below. In the close to the ideal case, the difference between green and yellow line should be as least as possible.



MPC Processing steps:

1. Set the time step length (N) and the elapsed duration between each time step (dt)
2. Define the cost function, that depends on the cross track error, orientation error and velocity error.
3. Run the MPC recursively in the loop
    a. Pass the current state as the initial states.
    b. Using the solver function, compute the control inputs to the actuators.
    c. Apply the control inputs to the actuators (angle to steering wheel and velocity/acceleration to the throttle)
    d. Go back to step a.

Model Predictive Control with Latency

Objective: The student implements Model Predictive Control that handles a 100 millisecond latency. Student provides details on how they deal with latency.

The walkthrough video clarifies most of the steps needed to complete the project. However, replicating the walkthrough did not work as expected. So, there was countless attempts of tuning/tweaking up the parameters.

Without incorporating the latency, the vehicle behaved based on what the MPC predicted it should've done 100ms ago. As a result, the car responds slowly to the changes in position. It was also observed that faster the vehicle speed, the worse the effects of latency if not taken into consideration.

As suggested in the lectures. the kinematic model was used to compute the state of the car 100ms ahead of time and then feed the predicted state into the MPC solver. This functionality is coded in the main.cpp file. The equations that were used to determine the current state that contain the latency factor have been extracted from the lessons.