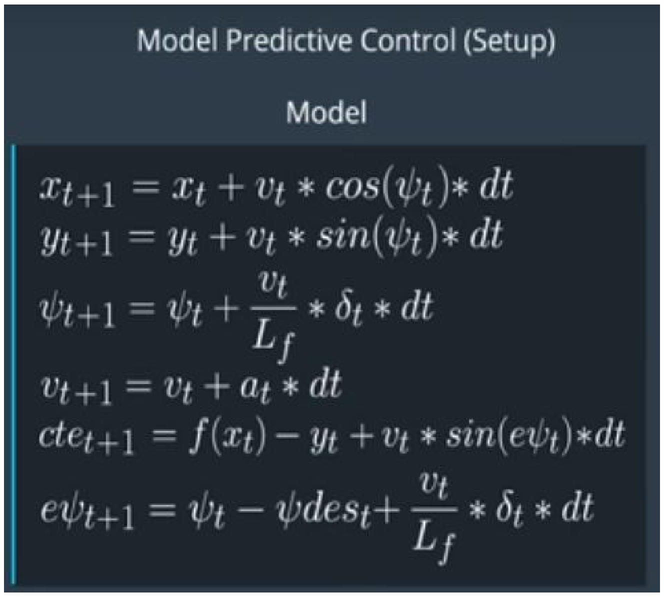
CarND MPC Project Writeup

Kyle Martin

Overview

The goal of this project was to implement and tune a MPC (Model Predictive Controller) to steer Udacity's racing simulator safely. After tuning I was able to get the car to drive over 100 MPH while staying on the track.

The Model



Model Predictive Control (Setup)

Model

$$
\begin{aligned}
x_{t+1} &= x_t + v_t * cos(\psi_t) * dt \\
y_{t+1} &= y_t + v_t * sin(\psi_t) * dt \\
\psi_{t+1} &= \psi_t + \frac{v_t}{L_f} * \delta_t * dt \\
v_{t+1} &= v_t + a_t * dt \\
cte_{t+1} &= f(x_t) - y_t + v_t * sin(e\psi_t) * dt \\
e\psi_{t+1} &= \psi_t - \psi des_t + \frac{v_t}{L_f} * \delta_t * dt
\end{aligned}
$$

The goal of our model is to mathematically describe the forces acting on a vehicle and output a steering angle and acceleration that will bring the vehicle to an optimal state.

This model makes the assumption that we have already accurately calculated the velocity, acceleration, orientation, and location of the car from a prior localization step.

With this model we can then predict the state of the vehicle in the future. Along with location of our car, we would also like to characterize the magnitude of how far away the vehicle's position and orientation is from the ideal.

I used the equations derived in Udacity's Vehicle Model class module (as seen in the above screenshot). The implementation for this is in main.cpp at lines 134-139.

The output of the model, steering angle and acceleration, are found by using a using an algorithm called Interior Point Optimization (IPOPT). Given a system of non-linear equations, constraints, and a cost function it can find optimal input values. The implementation of this step can be found in MPC.cpp.

Please note this model does not take into account more complicated forces on the car such as air resistance, tire forces, and complexities of the control system. Each of these forces could require a complete team of engineers to accurately model them.

Timestep Length and Elapsed Duration (N & dt)

Now that we have a model for our vehicle's behavior we need to determine how far ahead we want to plan the vehicle's path. This requires a careful balance between prediction horizon and hardware speed.

N is the number of timesteps we'll project into the future for each model simulation and dt is seconds between each timestep. We call (N * dt) the prediction horizon.

If we dt is too small then the controller can adapt very quickly to new situations, however the controller could become unstable and cause the vehicle to crash. If we allow dt to be too big then the controller will not be able to handle sharp road curvatures.

The second consideration is how big to make N. If we make N too small then the projected paths will have rapid changes in order to fit the model constraints. If we make N too big then the model solve time will take too long to be useful for real-time control. The ideal N results in smooth controls with plenty of time to perform a recalculation if needed.

The last consideration is how fast our hardware is. The faster the hardware the farther into the future we can simulate possible vehicle paths to find ideal actuation values. For this project I'm working on a Macbook Pro so I tuned N and dt to maximize top vehicle speed while staying on the road.

I tuned these values by hand and found the optimal parameters for my hardware were:

N = 8
dt = 0.100

Polynomial Fitting and MPC Preprocessing

In order to optimally control our vehicle we must have an ideal path that our vehicle should take. Is this project we assume this ideal path has been calculated in a prior path planning step and is an input to our control system.

The vehicle simulator provides us with waypoints of an ideal path in world coordinates, which are 90 degrees out of sync with the vehicle's coordinate system. So we need to transform each of the path waypoints into vehicle coordinates.

This implementation of this can be found in main.cpp at lines 103-113.

Model Predictive Control with Latency

As the model currently stands, we've assumed our vehicle can instantly change it's steering angle and acceleration. Realisticly, vehicles still have mechanical parts that take time to move from point A to point B.

In this project, we are given a static value of 100ms for the car to change actuation inputs. To compensate for this, we need to project the state of the vehicle 100ms into the future and use this new state when simulating different paths the vehicle could take. With this delay in place our model can more accurately predict the vehicle's path.

The implementation of this can be found in main.cpp at lines 128-138.