MPC Reflections

From the simulation, the mpc controller receives the current state of the car and a set of desired waypoints. The state of the car includes the current position and heading of the car x , y and psi in map coordinates and also the current velocity, throttle and steering angle.

1. Map coordinates to car coordinates

To make things simple the waypoint coordinates are first transformed into the car coordinate system. This happens between the code lines 104-119 where each waypoint is looped through and put into the transformation matrix. The end result means we can set the cars x, y, psi values to 0, making implementation of the mpc much easier.

2. Polyfit, finding the error

At this point the car is still missing values of CTE and EPSI from its current state vector, these values simply define the error of being in the current state. It is these values which will optimize our steering and throttle parameters later in the mpc controller. To calculate the CTE we first need to fit our transformed waypoints, line 121, to a 3^{rd} order polynomial. In our cars local coordinate system its state for x is always 0, so we insert this value into the polynomial, line 123, and the output will be an value indicating how far away our car is from the y value of the waypoint at position x=0. The difference in y values, between the car and the waypoint, is our CTE error.
Next the EPSI error is calculated on line 123. It is simply tangential angle of the polynomial, as in car coordinates our current direction is set to 0. This gives the angle difference between our current heading and the waypoint heading.

3. Handling lag

The mpc controller simulates a 100ms delay to account for actuator lag on a real self-driving car. The simplest way to handle this, by researching the Udacity forums, is to simply predict where the car will be 100ms into the future and send this state to the mpc controller. Lines 129-134 define the new predicted states after 100ms, using the bicycle model of motion. These values are then saved to a new state, line 138 main.cpp, which now includes our lag for the mpc controller.

4. Time steps and MPC controller indexing

A time step of 0.1 (100ms) with 10 steps was chosen on lines 9-10 mpc.cpp. These values were selected through trial and error and seemed to give good results. The mpc controller also expects the current state, and future N states with throttle and steering values as a 1 dimensional array. In order to keep track of variables in this array index locations are save in Lines 13-20 mpc.cpp.

5. Cost function in fg_eval

A core part of the mpc is defining the cost function. This is implemented in lines 54-75 in mpc.cpp. From trial and error it was found the cost for having a high CTE and EPSI error should be very large compared to cost contributions. The cost contribution from lines 73-75 mpc.cpp were especially useful for keep the car on the track. This cost is simply the velocity

multiplied by the steering angle. This had the effect that the car would avoid turning at high speed, essentially leading the car to break around the corners.

6. Constraints and variable bounds

Constraints are defined on lines 79 -119 mpc.cpp. These constraints first fix the car at its starting position, so the mpc can't just teleport the car to the lowest cost state. Then for each subsequent time step the constraints limit the position of the car as defined by the bicycle motion model. The optimizer is allowed to pick any values between -1.0e19 and 1.0e19 for each future state variables so long as it does not break the constraints. The exception is throttle and steering variables are fixed to be between -1 to 1 and -0.43 to 0.43 respectively. These limits are set between lines 166-183 mpc.cpp. Lines 185-207 mpc.cpp define the flexibility of the constraints and these are set to not allow the car the break the bicycle motion model.

7. Result

Finally IPOPT is called to find values for future states that satisfy all the above constraints and variable limits. These results are returned the mani.cpp where the car will use the best predicted throttle and steering values for the next time step. The rest of the values are discarded, with new values being calculated on the next loop. The predicted trajectory from IPOPT solver is shown as a green line on complete_lap.mp4.