Report

This code implements Model Predictive Control or MPC. The basic idea of MPC is a loop which takes the current state of the car and the waypoints from Path Planning module in each iteration and calculates the actuator values which find the trajectory with the lowest cost from desired path. The first calculated actuator values will be used to actuate the car and the rest will be discarded and then the loop repeats.

-The loop is implemented in main.cpp file in h.onMessage event handler:
First, the state variables and waypoints are extracted from the input parameters. Then the waypoint coordinates are transformed to car coordinate system. The waypoints data in car coordinate system will be passed to "polyfit" function to find the 3rd degree polynomial fitting the waypoints. From the polynomial, values of the errors for cross track error and orientation error are calculated. These error values along with initial state variables make the overall state variable (x,y,psi,v,cte, epsi). Then this state and waypoints are passed to "mpc.solve" function to calculate the set of actuator values alpha and delta which minimize the cost from the desired path. We get the first values from this solution and set the values of "steer_value" and "throttle_value" by those. Also we populate the variables to display the waypoints and calculated trajectory in simulator.

-It is MCP class implemented in MCP.cpp which actually finds the trajectory based on the waypoints polynomial and current state. It sets constraints upper and lower bounds and uses FG_eval class to compute objectives and constraints. FG_eval calculates the next states based on the update equations and the cost based on the sum of the squares of reference state, actuators and gap between sequential actuations. I tried different weights for the elements of the cost to make the car drive steady. I calculated the 3rd degree polynomial value instead of line which was in class example to have a better fit to the waypoints.

-To adjust N and dt, we know that T = N * dt should be as large as possible, while dt should be as small as possible. In the case of driving a car, T should be a few seconds, at most. Beyond that horizon, the environment will change enough that it won't make sense to predict any further into the future.

I set the ref_v = 90 to see if the car can drive safely with this high speed. I chose T = 1s and started with N = 5 and dt = 0.2 which created a short trajectory which is not following the waypoints well and the car gets off the road quickly.

Next N = 10 and dt = .1. With these values the green trajectory follows the waypoints pretty good and car drives the complete track properly. I kept these values for submission.

For experiment, I tried N = 20 dt = .05, which causes the car gets too sensitive to errors and car sways to follow the waypoints until it gets unstable and gets off the road.

-The code simulates the latency by sleeping the thread for certain amount of latency like 100ms before it sends the actuations to the car. This is a good property of MPC that latency can be modeled as part of the system dynamics. I changed the value of latency to values higher than 100ms which made car lazy to react and get off the road. For high latency, lower speeds makes car more stable although it still sways but stays on the road. Changing latency to values lower than 100ms made car too sensitive which caused car gets unstable in the curves. In this case also, lowering the speed makes car drive more stably.