Udacity – MPC Project

Submitted by: Shaurya Dwivedi

Date: 25th April, 2018

Objective

The objective of this project is to create a MPC and to use it to drive the car in simulator successfully.

MPC

MPC stands for Model Predictive Control. It's an advanced way of controlling a process while fulfilling a set of constraints. We implemented the **kinematic model** which is a simplified version of dynamic model, and ignores many forces (like tire forces, gravity, mass, etc…).

State

The state helps in keeping track of the vehicle. We are using following parameters to track the state of the vehicle:

Parameter	Remarks
X	The x location in 2d plane
Y	The y location in 2d plane
Psi (ψ)	Orientation of vehicle
V	Velocity of the vehicle
CTE	Cross Track Error (error in trajectory and predicted path of vehicle)
eψ	Error in vehicle orientation

Actuators

Actuator input allows to control the vehicle state. We are using only two actuators here (for simplicity)

Parameter	Remarks
Steering (δ)	The steering value, +ve denotes steer to right while –ve denotes steer to left
Throttle (a)	+ve value denotes forward, while –ve value denotes brake. Its value ranges between -1 and +1

Update Equations

$$x_{t+1} = x_t + v_t cos(\psi_t) * dt$$

$$y_{t+1} = y_t + v_t sin(\psi_t) * dt$$

$$\psi_{t+1} = \psi_t + \frac{v_t}{L_f} \delta_t * dt$$

$$v_{t+1} = v_t + a_t * dt$$

$$cte_{t+1} = y_t - f(x_t) + (v_t * sin(e\psi_t) * dt)$$

$$e\psi_{t+1} = \psi_t - \psi des_t + (\frac{v_t}{L_f} * \delta_t * dt)$$

dt => rate of change of state
Lf => distance between the vehicle's center of mass and its front axle

Time step Length and Elapsed Duration (N & dt)

The time step length defines how many states we need to look ahead (future prediction). The elapsed duration defines what is the frequency we expect the state (environment) will change. For this project I used the time step length (N) as 10 and the value of Elapsed duration (dt) as 100 milliseconds.

Polynomial Fitting and MPC Preprocessing

As suggested in the course material I used 3rd degree polynomial. I tried 2nd and 4th degree polynomials as well but the predicted path was not as smooth as I got for 3rd degree.

The given waypoints are in the global coordinate system; thus these are needed to convert to the vehicle's local coordinate system. For this the following formula is used (same as we did in the Extended Kalman Filter)

```
for(unsigned int i=0; i < ptsx.size(); i++){

   double x = ptsx[i] - px;
   double y = ptsy[i] - py;
   //rotation of coordinates
   ptsx[i] = x*cos(psi) + y*sin(psi);
   ptsy[i] = -x*sin(psi) + y*cos(psi);
}
```

Final penalty weights that I settled with (the main aim is to give weight to the cost parameters so that the weightage of a particular parameter be increased. More the weight, more penalty it will have on the cost error):

Weight for	Weight value	Remarks
CTE	2	Cost weight for reference state
EPSI	20	
Velocity	20	
Delta (Steering)	450	Cost for actuators
Acceleration (Throttle)	20	
Delta (sequential)	450	Cost for value gap between sequential actuations
Acceleration (Sequential)	20	

Model Predictive Control with Latency

To mimic the latency a sleep of 100ms is added in the code. To incorporate the latency, I used a value of 100ms (named it dt – which is same as used in MPC.cpp) to calculate the next state of psi, v and *cte*. These values are then fed to the `mpc.Solve` function along with other state values to predict the steer angle and throttle along with x and y coordinates of the trajectory.

Final Thoughts

I found that MPC gave much better results as compared to PID. However, it is little difficult to implement but it seems worth the efforts. Now I am looking for some more examples where can implement the MPC.