MPC (Model Predictive Control)

Model

1. State

For this project, I have used state of 6 elements as described below

[x , y , psi , v , cte , epsi]

x ,y – x and y coordinates or position of the vehicle in simulator.

psi – orientation of the vehicle

v - speed of the vehicle

cte – cross track error

epsi – error in psi , difference between current psi value and desired psi value.

2. Actuators

There are 2 actuators to control as described below

[delta , a]

Delta – Steering angle of the vehicle

A – Acceleration

3. update equations

A kinematic vehicle motion model is used to model the vehicle as shown below

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

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

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

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

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

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

Note: Here deltaT is considered negative and sign of the last term in equations 2 and 6 has been changed to minus as in simulator left steering yields negative value of deltaT.

Tuning of N and dt

N represents number of timestamps to consider for optimization at a given current position

dt represents the time difference of one timestamp

size_t N = 20;

double dt = 0.2;

I have chosen N as 20 and dt as 0.2 seconds, I started with small value (N = 10 and dt = 0.1) of N and dt, but the performance of those parameter was not good, it was not able to steer at curves properly and vehicle was wobbling over reference trajectory. To make vehicle able to see the coming path and consider that while optimizing I increased the value to (N = 20 and dt = 0.2), dt was increased to reduce the processing time, with these values vehicle was considering 4 seconds of path ahead and was able to plan better.

Apart from N and dt, I was also used weights in cost function to make the drive smoother.

```
69      // Adding SE of CTE , EPSI and DIFF_VELOCITY to cost function
70      for(size_t t = 0; t < N; t++){
71          fg[0] +=  0.25*CppAD::pow(vars[CTE_START+t],2);
72          fg[0] +=  0.25*CppAD::pow(vars[EPSI_START+t],2);
73          fg[0] +=  0.25*CppAD::pow(vars[V_START+t]-ref_v,2);
74
75      }
76      // Adding Square of value of delta and acceleration to minimize heavy use of actuators
77      for(size_t t = 0; t < N-1; t++){
78          fg[0] +=  7e-4*CppAD::pow(vars[DELTA_START+t],2);
79          fg[0] +=  7e-4*CppAD::pow(vars[A_START+t],2);
80          // penalty for speed * steer
81          fg[0] += 0.19*CppAD::pow(vars[DELTA_START + t] * vars[V_START+t], 2);
82      }
83
84      // Adding sequential difference of actuations for smooth driving
85      for(size_t t = 1; t < N-1; t++){
86          fg[0] +=  0.03*CppAD::pow(vars[DELTA_START+t] - vars[DELTA_START+t-1],2);
87          fg[0] +=  0.0014*CppAD::pow(vars[A_START+t] - vars[A_START+t-1],2);
88      }
```

(MPC.cpp – inside class FG_eval stars from line number 70)

I have added current CTE, EPSI and error in velocity i.e. velocity – reference velocity to cost function to get a constant speed and getting rid of vehicle stoppage in middle of the way.

I have also added values of delta and acceleration as well as successive values of actuators to the cost function to minimize jerks. Penalty for speed * steer reduces the speed while turning, which will stop vehicle to go out of tracks at sharp turns.

Polynomial Fitting and MPC Preprocessing

Polynomial fitting has been used to fit the curve on given waypoints to generate reference trajectory. It is shown as yellow path in simulator. 3rd degree polynomial is being used as it fit almost perfect with waypoints.

Before fitting the Polynomial, waypoints have been converted to the vehicle's coordinate system using rotation and translation of coordinate system. This preprocessing makes calculation easier and we can use simple update equations as described above. After this vehicle state will have x, y, psi as 0.

Steering values are also being normalized before sending it to simulator by 25 degrees which is maximum possible steering angle.

Latency

There is 100 ms of latency between application of actuators and response of the application.

To compensate this latency factor has been added to the code and which calculates the state of the vehicle after 100 ms and consider that while solving optimization problem.

```
158          double new_x =  v*cos(psi)*latency; // x = x0 + v*cos(psi)*latency
159          double new_y =  v*sin(psi)*latency; // y = y0 + v*sin(psi)*latency
160          double new_psi = -(v/Lf)*steer_prev*latency; // psi = psi0 + (v/Lf)*delta*latency;
161          double new_v = v + throttle_prev*latency; // v = v0 + a*latency;
162
```

(Main.cpp – start line number: 158)