Model Predictive Control (MPC)

1. Files Submitted

- main.cpp – main program
- MPC.cpp – MPC C++ code of implementation

2. Implementation

Note: due to OS and build environment difference between my local MacBook Pro and web-based Udacity workspace, using size_t instead of int in for loop to avoid the warning of signed and unsigned integer comparison

(warning: comparison between signed and unsigned integer expressions [-Wsign-compare])

2.1 The Model.

The model state vector {px, py, psi, v, cte, epsi} is fed by simulator, and transformed into a car's coordinate, resulting {px, py, psi} are 0 after the transformation, the simulator expects this transformation for the returned plotted center line and vehicle path projection.

The simulator also supplies waypoints for the center of the road (desired path). These waypoints are fitted to a 3rd order polynomial. Hence, the cte and epsi are calculated

The kinematic equations used in the model:

```
x1 = (x0 + v0 * CppAD::cos(psi0) * dt);

y1 = (y0 + v0 * CppAD::sin(psi0) * dt);

psi1 = (psi0 + v0 * delta0 / Lf * dt);

v1 = (v0 + a0 * dt);

cte1 = ((f0 - y0) + v0 * CppAD::sin(epsi0) * dt);

epsi1 = ((psi0 - psides0) + v0 * delta0 / Lf * dt);
```

Where f0 is the result of a function that uses the derived coefficients and y0 to provide a distance from the desired path.

The cost function is from the MPC quiz and the walkthrough. It sets cost values for error states (cte, epsi) and velocity. With very high cost associated with the error functions. It also sets constraints on the actuators for both minimizations of use and sequential state to state changes.

```
// The part of the cost bsed on the reference state
for (int t = 0; t < N; t++) {
  fg[0] += w_cte * CppAD::pow(vars[cte_start + t] - ref_cte, 2);
  fg[0] += w_epsi * CppAD::pow(vars[epsi_start + t] - ref_epsi, 2);
  fg[0] += w_ref_v * CppAD::pow(vars[v_start + t] - ref_v, 2);
}

// Minimize the use of actuators
for (int t = 0; t < N - 1; t++) {
  fg[0] += w_delta * CppAD::pow(vars[delta_start + t], 2);
  fg[0] += w_a * CppAD::pow(vars[a_start + t], 2);
}

// Minimize the value gap between sequential actuations
for (int t = 0; t < N - 2; t++) {
  fg[0] += w_delta_diff * CppAD::pow(vars[delta_start + t + 1] - vars[delta_start + t], 2);
  fg[0] += w_a_diff * CppAD::pow(vars[a_start + t + 1] - vars[a_start + t], 2);
 }
```

The finalized parameters are:

```
// Goal states for cte, epsi and velocity
double ref_cte = 0;
double ref_epsi = 0;
double ref_v = 120;
```

```
// Important to constrain errors
const int w_cte = 2500;
const int w_epsi = 2500;
const double w_ref_v = 1.0;

// Not important to constrain actuators
const int w_delta = 1;
const int w_a = 1;

// Important to minimize the gap between sequential actuators to reduce jerk
const int w_delta_diff = 200;
const int w_a_diff = 5;
```

The cte and epsi costs need to set a bit higher to drive within the lane lines even the speed is low. Some experiments found that lower weights of cte and epsi in the 3rd and 4th power lead to an unstable vehicle route.

A high-enough ref_v value penalizes distance from the speed of 120, and the model is very sensitive to the weight applied to that cost(w_ref_v=1.0). At w_rev_v=2.0 the model fails quickly in certain turns, at 1.5 it would probably pass, but occasionally crashes after multiple laps. The parameter of w_a_diff. 5 seems to be the largest value to achieve the goal.

2.2 Timestep Length and Elapsed Duration (N & dt)

The final selection of Timestep Length and Elapsed Duration is 100:1 (N = 10 & dt = 0.1). Some other values seem got no actual meaning and better not to change much in the test

The values of dt=0.8 and N=8 reflect some instability when tested with the latency included. No improvement was noted for dt of 0.12 and 0.14 so default set as 0.1.

2.3 Polynomial Fitting and MPC Preprocessing

Waypoints are polynomially fitted, and additional set of values are generated in next_x and next_y json to give more points to plot rather than just returning transformed ptsx and ptsy from input telemetry.

The waypoints returned are not pre-processed. However, the coefficients are derived from the pre-processed telemetry waypoints. The processing is a rotation and translation to bring the points into car coordinate system with the first point having x, y and psi values of 0.

2.4 Model Predictive Control with Latency

The 100ms latency is achieved by projecting the vehicle state 100ms to the future and using that state as input to the MPC unit so that when the control parameters are used after the 100ms delay.