Model Predictive Control – Project 10

In this project it is required to drive the vehicle around the track at maximum possible speed without pushing the vehicle off the track. Using Model Predictive Control (MPC) the vehicle chooses a path closest to the waypoints thereby trying to stay at the center of the track.

In this project a kinematic model of vehicle is chosen with below parameters,

1. x position (x)
2. y position (y)
3. orientation (psi)
4. velocity (v)

The state also includes,

5. cross-track-error (cte)
6. orientation error (epsi)

Thereby forming a state vector of 6 parameters, (x, y, psi, v, cte, epsi)

The vehicle's control, steering accelerator and break is modeled as,

1. steering angle (delta)
2. acceleration (a)

A positive value of (a) indicates acceleration and negative value of (a) indicates breaking.

Using the state at time (t) we can predict the state at time (t+1) as below,

$$x(t+1) = x(t) + v(t) * \cos(psi(t)) * dt$$
$$y(t+1) = y(t) + v(t) * \sin(psi(t)) * dt$$
$$psi(t+1) = psi(t) + v(t)/Lf * delta(t) * dt$$

(Where Lf = distance from vehicle's center of gravity to the front of the vehicle)

$$v(t+1) = v(t) + a(t) * dt$$
$$cte(t+1) = cte(t) + v(t) * \sin(epsi(t)) * dt$$

(Where cte(t) = f(x(t)) – y computed as current cte)

$$epsi(t+1) = epsi(t) + v(t)/Lf * delta(t) * dt$$

(Where epsi(t) = psi(t) – psi(desired) and p(desired) = arctan(f'(x(t)))

Using the waypoints, we need to fit a polynomial which becomes a guiding line for the vehicle to follow. The cross-track-error (cte) and orientation error (epsi) will be evaluated against this polynomial. A 3rd degree polynomial is found to fit all the waypoints around the track in the simulator.

Before solving for the line, the waypoints in the racetrack are provided in global map co-ordinates which should be converted to local vehicle co-ordinates.

For vehicle, x-direction is the direction of the heading and y-direction is to the left of the vehicle. In the map, x-direction is along the width of the image and y-direction along the height of the image (up-down). Hence we need to apply a translation and rotation.

In-order to treat all points in local vehicle's co-ordinate, we apply a translation by subtracting the vehicle's (px, py) location from the way-point's (ptsx and ptsy) locations. In the simulator, a positive angle is counter-clockwise direction and negative angle is clockwise direction. So the orientation angle (psi) must be negated.

Way points are preprocessed as below,

Translation – x coordinate	Xdiff = ptsx[i] – px
Translation – y coordinate	Ydiff = ptsy[i] – py
Rotation – x coordinate	Xdiff * cos (0 – psi) – Ydiff * sin (0 – psi)
Rotation – y coordinate	Ydiff * cos (0 – psi) + Xdiff * sin (0 – psi)

Now we compute a 3rd degree polynomial using the transformed way points and use the computed coeff's for MPC.

```
coeffs = polyfit(ptsx_v, ptsy_v, 3);
```

Since the car is centered in the previous transformation, the initial state becomes

```
(x=0, y=0, psi=0, v, cte, epsi)
```

The impact of latency is discussed later

In MPC we need to predict a set of actuator commands (steering and accelerator) for duration of T secs.

This is defined as $T = N * dt$ where N is the number of prediction steps and dt is the delta time between 2 consecutive actuation commands.

Higher value of N increases the number of prediction steps. This means that MPC have to predict N steps ahead in time with respect to current state. We take only the first prediction and discard the remaining (N-1) predictions. Since we perform this prediction at every time step, keeping a high value of N will only increase the computation time and thereby increase the delay in sending the actuation command.

The value of dt controls the accuracy of prediction. Higher the value of dt, less accurate is the prediction. Eg. If dt = 1 then the time between predictions is 1 sec. This is dangerous especially considering the speed at with vehicle is moving. For higher speed, it is important to predict at finer granularity and send out actuation at faster intervals for smooth acceleration and breaking.

Number of predictions (N)	**Time between consecutive actuations (dt)**	**Observation**
30	1	Car struggles to stay on track around the corners.
25	0.5	Reduced computation, car still goes off the track at higher speeds
20	0.1	Car stays on track, but takes more computation time
10	0.1	Car stays on track and takes reasonable computation time

Handling latency

Best quality of MPC is modeling latency. It is required to drive the car around the track at high speed with a latency of 100ms from the time actuation is computed and applied in the simulator.

Using the state update equations, we can compute the state of (x, y, psi, v) 100ms ahead in time before computing the polynomial and applying MPC. Other approach is to play with dt and select an average of “k” output actuations.

In the provided solution the state is predicted assuming 100ms latency by using current steering angle and throttle values as below,

```
double delta = j[1]["steering_angle"]; //Read current steering angle
double acc = j[1]["throttle"]; //Read current throttle value

//predict state in 100ms
double latency_in_sec = 0.1;

//Only if you have a latency predict ahead
if(latency_in_sec > 0.0) {
  px = px + v * cos(psi) * latency_in_sec;
  py = py + v * sin(psi) * latency_in_sec;
  //Make sure psi at t+1 is psi at t "minus" the update
  psi = psi - v * delta/Lf * latency_in_sec;
  v = v + acc * latency_in_sec;
}
```

The project is available at https://github.com/shyam-jagannathan/CarND-MPC-Project

Modified files are src/MPC.cpp and src/main.cpp

A video of a successful run with 100ms latency is available at, https://youtu.be/6sW3Nzj-FyQ