Model Predictive Control (MPC) Project

The Model

The state contains the x position, y position, vehicle orientation, velocity, cross track error or **cte**, which express the error between the center of the road and the vehicle's position, and the orientation error or **epsi**.

It also has two actuators: the steering wheel, and the throttle and brake pedals considered as a single actuator (negative values signifies braking, and positive values signifies accelerating). Therefore, we have two control inputs, the steering angle **δ** and the acceleration **a**.

The model we have chosen in the one explained in the course, with the following update ecuations:

> 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
> v_[t+1] = v[t] + a[t] * dt
> cte[t+1] = f(x[t]) - y[t] + v[t] * sin(epsi[t]) * dt
> epsi[t+1] = psi[t] - psides[t] + v[t] * delta[t] / Lf * dt

Please notice that **Lf** measures the distance between the front of the vehicle and its center of gravity.

Our constrains are **[δ, a]**, with steering angle **δ∈[-25,25] degrees** (or **δ∈[-0.436332, 0.436332] in radians**) and acceleration **a∈[-1,1].**

In my model, I am trying to minimize the predicted distance of the vehicle from the trajectory (cte). Also, I try to minimize the predicted difference between the vehicle orientation and trajectory orientation (epsi).

Timestep Length and Elapsed Duration (N & dt)

The duration over which future predictions are made is the prediction horizon or **T**.

N is the number of timesteps in the horizon. ***dt*** is how much time elapses between actuations.

> **T = N x dt.**

In my final model, I have chosen N=10 and dt=0.1, which gives a prediction horizon equals 1 second. I expect that in 1 second, the environment does not change too much.

I have tried other combination of values, such as {N=10, dt=0.5} and {N=20 , dt=0.1}, but the one I have noticed better results is the one I mentioned at the beginning {N=10, dt=0.1).

Polynomial Fitting and MPC Preprocessing

I calculate waypoints by using the following formula:

```
    waypoints_x[i] =  cos_psi * (ptsx[i] - px) + sin_psi * (ptsy[i] - py);
    waypoints_y[i] = -sin_psi * (ptsx[i] - px) + cos_psi * (ptsy[i] - py);
```

I use a helper (**polyfit**) to fit a third order polynomial to waypoints, and another helper function (**polyeval**) to calculate the cross track error (**cte**) and orientation error (**epsi**).

The initial state is set with **x, y** and **psi** equals to 0. The initial state values for **cte** and **epsi** are the ones obtained the former paragraph. The initial state value for **v** is calculated as described in the next section (**Model Predictive Control with Latency**).

```
          // set the initial state, with x, y and psi equals 0
          VectorXd initial_state(6);
          initial_state << 0, 0, 0, v, cte, epsi;
```

Steering angle and throttle are calculated invoking **mpc.Solve()** method as follows:

```
          // obtain throttle and steer via mpc
          MPC_Results res = mpc.Solve(initial_state, coeffs);
          double steer_value    = -res.steering;
          double throttle_value =  res.throttle;
```

Model Predictive Control with Latency

In real cars, an actuation command doesn´t execute instantly. There is a delay as the command propagates through the system. This is known as "latency".

To deal with latency, we implement the following equations:

```
          // Deal with latency
          px = px + v * cos_psi * latency;
          py = py + v * sin_psi * latency;
          psi = psi - v * steering / 2.67 * latency;
          v = v + throttle * latency;
```

where 2.67 is the Lf. That value was obtained by measuring the radius formed by running the vehicle in the simulator around in a circle with a constant steering angle and velocity on a flat terrain.