Model Predictive Control (MPC): Krishna

Description:

In this project, we aim to develop a nonlinear Model Predictive Control (also known as dynamic control) to steer a car in the simulated environment around a track. The goal of the project is to keep the car in the lanes such that it does not goes out of the lane considering that there is a human in the car and the car does not drive unsafely. The simulator provides values for position, heading direction, speed and the reference trajectory of the car. The simulator basically provides coordinates for the waypoints.

Compilation:

The code successfully compiled in my laptop using the cmake and make command as required in the project. I have done the project with a speed of 60.

The Model:

The vehicle model used in this project for the car is the kinematic bicycle model as discussed in the Udacity classroom. The kinematic model ignores the dynamic effects or forces like friction, inertia, torque, drag, etc. The model uses the concepts of heading direction, velocity and position and thus can be used at low or moderate speeds. It consists of the following 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] + 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
```

where,
x and y are the respective x and y positions;
psi is the heading direction; v is the velocity;
cte is the cross track error measured from the desired reference path;
epsi is the error in the heading direction
a: is the acceleration of car (throttle)

delta: is the steering angle

The main objective is to find the control inputs which are acceleration and steering angle such that it minimizes an objective function or cost function. The cost function is a combination of different factors like cross track error, heading direction error, difference of actuators to penalize many actuator's actions, or penalize sharp changes, etc. All these are squared and then added to form the cost function.

Timestep Length and Elapsed Duration (N & dt):

The number of points (N) and the time interval duration dt together defines the prediction horizon for the MPC control. The number of points though increase the performance but it makes the system run slower. I tried N values for 10, 15 and 20 and dt 100,250,500 ms accordingly. But finally I, decided to leave them fixed at 10 and 100ms. Though 15 and 250ms also worked fine.

Polynomial Fitting and MPC Preprocessing:

Now, in order to simplify the polynomial fitting process, I converted the points from the simulator's map coordinate system to vehicle's coordinate system. After the conversion, a 3rd degree polynomial is fit to the converted waypoints by calling the polyfit function. The polyfit function returns 3rd degree coefficients which are then used to calculate the cte and epsi.

Model Predictive Control with Latency:

MPC handles latency very easily. To account for the 100ms latency in the system, I added an additional step to calculate the state using the latency time interval and then used that state to calculate the model instead of the initial one. Thus, in this way the latency problem which occurred during PID, is taken care of.

Simulation:

The vehicle completes one lap without going over the edges. Though there are fluctuations in the movement, but the car performs pretty well in this situation.

However, the current model might not work well for higher speeds as the parameters are tuned for speed below or around 60.