UDACITY

项目

Model Predictive Control (MPC)

此部分属于 Self-Driving Car Program

项目审阅

代码审阅 1

注释

与大家分享你取得的成绩！

Requires Changes

还需满足 1 个要求 变化

You're almost there and you're going in the right direction. This project is challenging and many of the concepts may not have been made cleared in the lessons but I think it will be rewarding once you figured out them from the project.

Compilation

Code must compile without errors with `cmake` and `make`.

Given that we've made CMakeLists.txt as general as possible, it's recommend that you do not change it unless you can guarantee that your changes will still compile on any platform.

Implementation

Student describes their model in detail. This includes the state, actuators and update equations.

You have included a report which also contain update equations that can come handy when implementing the code for latency handling.

Student discusses the reasoning behind the chosen *N* (timestep length) and *dt* (elapsed duration between timesteps) values. Additionally the student details the previous values tried.

Good observation. In general, smaller dt gives better accuracy but that will require higher N for given horizon (N*dt). However, increase N will result in longer computational time which effectively increase the latency. The most common choice of values are N=10 and dt=0.1 but anything between N=20, dt=0.05 should work.

A polynomial is fitted to waypoints.

If the student preprocesses waypoints, the vehicle state, and/or actuators prior to the MPC procedure it is described.

You have implemented coordinate conversion and polynomial fitting correctly as evidenced from the waypoint plotting in the simulator. Nice!

The student implements Model Predictive Control that handles a 100 millisecond latency. Student provides details on how they deal with latency.

You have chosen the right approach but you haven't implemented the update equations correctly.
If you do the update after polynomial fitting, then this involves 2 steps:

- At current time t=0, your car's states are px=0, py=0, psi=0 right after converting to car coordinate. There you calcualte cte and epsi

```
cte = desired_y - actual_y
    = polyeval(coeffs,px)-py
    = polyeval(coeffs,0) because px=py=0
epsi =  actual psi-desired psi
     = psi - atan(coeffs[1]+coeffs[2]*2*px+...)
     = -atan(coeffs[1])  because px=psi=0
```

- Now predict all the states for t=latency. You can simplify them further knowing some of the variables are 0.

```
//  change of sign because turning left is negative sign in simulator but positive yaw for MPC
double delta = -j[1]["steering_angle"];
px = px + v*cos(psi)*latency;
py = py + v*sin(psi)*latency;
psi = psi + v*delta*latency/Lf;
epsi = epsi + psi;
cte= cte+v*sin(epsi)*latency;
v = v + a*latency;
```

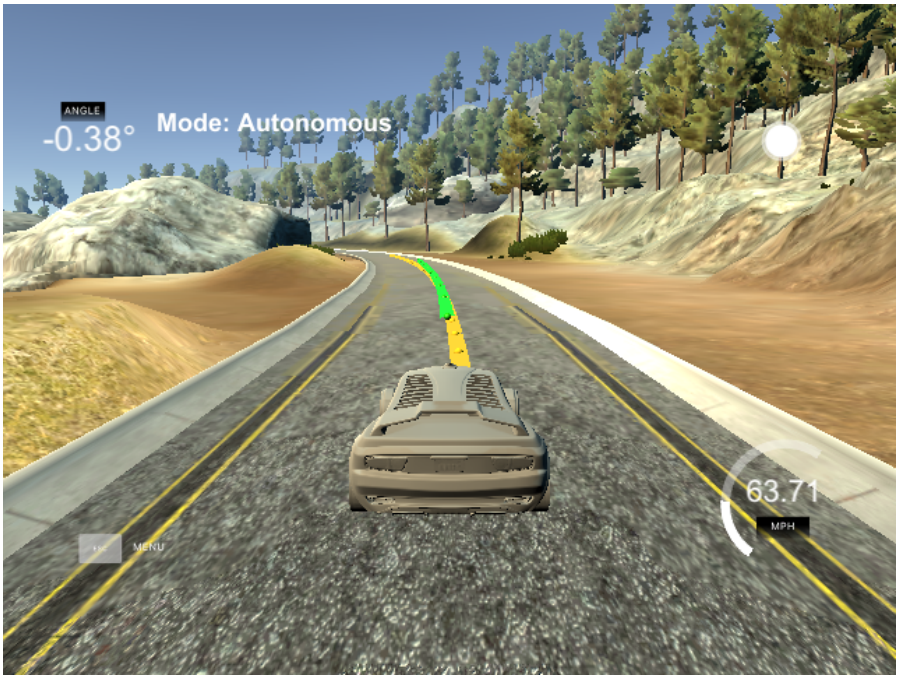
There is actually a simpler alternative to do the prediction before polynomial fitting where you update x,y,psi and v in global map coordinate. Then the fitted polynomial will be based on your states in the future, in other words, when x=0, y=0 in car coordinate will refer to your car's future position and you can use the simple cte and epsi equation. It can seems confusing in the first place but it will make sense once you understood the relation between fitted polynomial and your car position.

Simulation

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

The car can't go over the curb, but, driving on the lines before the curb is ok.

Your car was driving rather well at 64mph which I presume is almost the same as your like submission. It almost touched (but didn't) the curb at corners. If you have implemented latency handling correctly, you should be able to achieve at least 70mph or even higher.



重新提交

下载项目