PID Control Project

Introduction

The purpose of this project was to build a PID controller and tune the PID hyperparameters by applying the general processing flow, and to test the solution on the simulator. The simulator provides cross-track error (CTE), speed, and steering angle data via local websocket. The PID (proportional/integral/differential) controller must respond with steering and throttle commands to drive the car reliably around the simulator track. In addition, the value of throttle has been increased into 0.5 for purpose of increase the speed in this project.

Compilation

The programs that have been written to accomplish the project are: src/PID.cpp, and src/main.cpp.

This project made a 'build' directory in the project folder, and start compiling by doing the following:

```
cmake .. && make
```

The C++ code will be compiled without errors.

Implementation

Steering control is implemented in src/main.cpp which uses PID class (implemented in PID.cpp). Steering angle using a PID controller is given by:

```
steer_value = -Kp_ * p_error_ - Ki_ * i_error_ - Kd_ * d_error_
```

Here, Kp_, Ki_, Kd_ are the coefficient for PID components. p_error_ is the cross-track error (CTE), i_error_ is the integral of CTE, and d_error_ is the differential of CTE.

Reflection

The P, or "proportional", component had the most directly observable effect on the car's behavior. It causes the car to steer proportional (and opposite) to the car's distance from the lane center (which is the CTE) - if the car is far to the right it steers hard to the left, if it's slightly to the left it steers slightly to the right.

The D, or "differential", component counteracts the P component's tendency to oscillating and overshoot the center line. A properly tuned D parameter will cause the car to approach the center line smoothly without oscillating.

The I, or "integral", component counteracts a bias in the CTE which prevents the P-D controller from reaching the center line. This bias can take several forms, such as a steering drift and so on. The I component particularly serves to reduce the CTE around curves.

For this project, all parameters were tuned manually. This was necessary because the narrow track left little room for error. The tuning manually method provides an intuitive understanding of the importance of the different contributions. I felt it necessary to complete a full lap with each change in parameter because it was the only way to get a decent score for the parameter set. The algorithm used was roughly as follows.

1. Single P controller Kp_ for one setpoint, with Ki_ and Kd_ = 0
2. Increase coefficient Kd_ until oscillations subside.
3. In case of crashes, find cause: (a) If slow reactivity is the cause, then increase Kp_. (b) If oscillations are the cause, then increase Kd_.
4. The coefficient Ki_ is not necessary for this specific case, or set it as a very small value.

In this project, I have increased the throttle of car from original value 0.3 (related speed is around 30 mph) to new value 0.5 (set in main.cpp: msgJson["throttle"] = 0.5;) for purpose of increase the speed (related speed is around 45 mph). As a result, turning the PID hyper-parameters by throttle=0.5 become harder than throttle=0.3. In the end, the final values were determined by manual tuning. The coefficients as (0.60, 0.00005, 60.0) is to work well. And I also tried lowering & raising them in conjunction with each other as well as tuning each individually, but the results is not good, even worse.

Note: The coefficients (0.60, 0.00005, 60.0) was obtained under throttle=0.5. But I noticed that the same coefficients work well under throttle=0.3. -- But not vice versa. If a set of coefficients was obtained and work well for throttle=0.3, it won't work well for throttle=0.5 in most of situation.

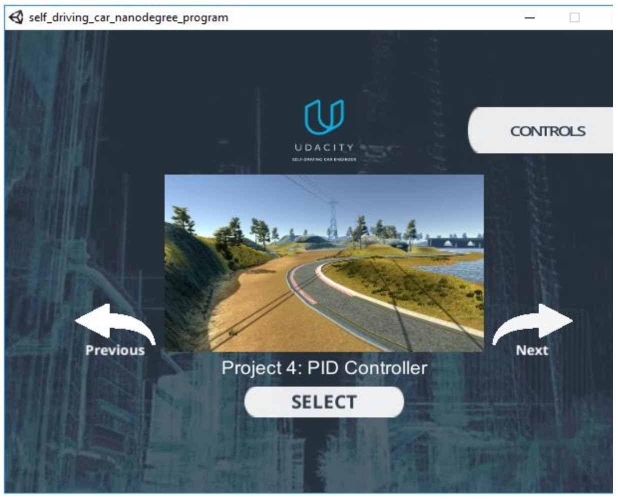
Simulation

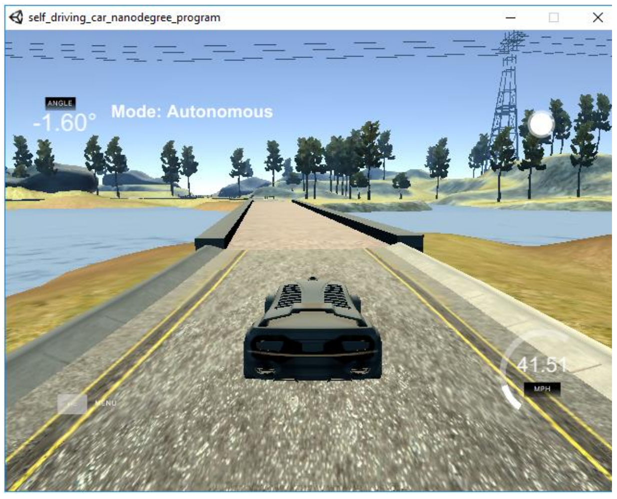
After the C++ code is compiled by command 'cmake .. && make', you can input following command:

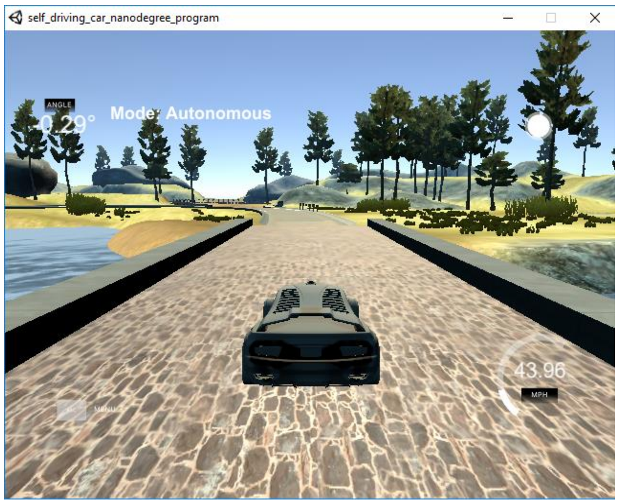
```
./pid
```

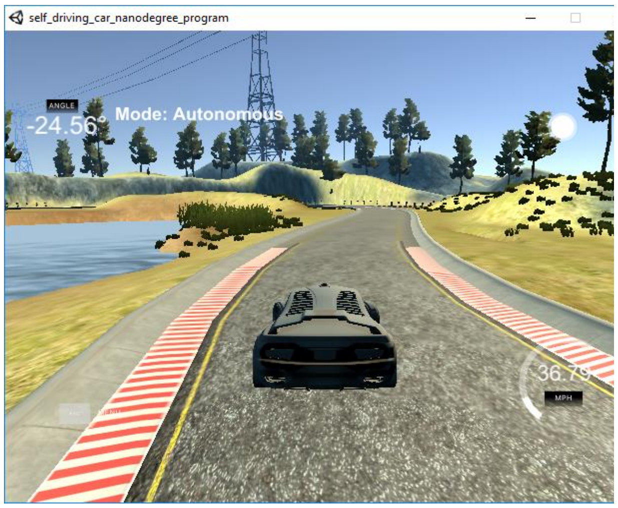
Now the PID controller is running and listening to port 4567 for messages from the simulator. Next step is to open Udacity's simulator. Using the right arrow, you need to go to the Project 4: PID Controller project. Click the "Select" button, and the car starts driving. You will see the debugging information on the PID controller terminal.

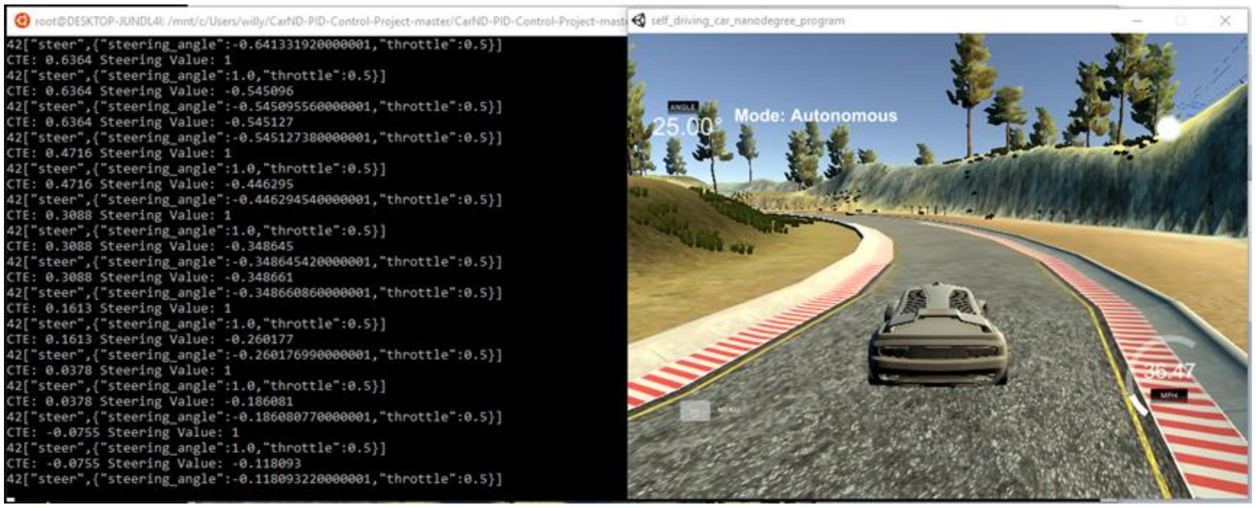
The pictures below show the demonstration on PID Controller in the simulator.


self_driving_car_nanodegree_program
UDACITY
CONTROLS
Previous
Next
Project 4: PID Controller
SELECT








root@DESKTOP-JUNDL4I: /mnt/c/Users/willy/CarND-PID-Control-Project-master/CarND-PID-Control-Project-mast
self_driving_car_nanodegree_program
ANGLE
25.00°
Mode: Autonomous
36.47
MPH

```
42["steer",{"steering_angle":-0.641331920000001,"throttle":0.5}]
CTE: 0.6364 Steering Value: 1
42["steer",{"steering_angle":1.0,"throttle":0.5}]
CTE: 0.6364 Steering Value: -0.545096
42["steer",{"steering_angle":-0.545095560000001,"throttle":0.5}]
CTE: 0.6364 Steering Value: -0.545127
42["steer",{"steering_angle":-0.545127380000001,"throttle":0.5}]
CTE: 0.4716 Steering Value: 1
42["steer",{"steering_angle":1.0,"throttle":0.5}]
CTE: 0.4716 Steering Value: -0.446295
42["steer",{"steering_angle":-0.446294540000001,"throttle":0.5}]
CTE: 0.3088 Steering Value: 1
42["steer",{"steering_angle":1.0,"throttle":0.5}]
CTE: 0.3088 Steering Value: -0.348645
42["steer",{"steering_angle":-0.348645420000001,"throttle":0.5}]
CTE: 0.3088 Steering Value: -0.348661
42["steer",{"steering_angle":-0.348660860000001,"throttle":0.5}]
CTE: 0.1613 Steering Value: 1
42["steer",{"steering_angle":1.0,"throttle":0.5}]
CTE: 0.1613 Steering Value: -0.260177
42["steer",{"steering_angle":-0.260176990000001,"throttle":0.5}]
CTE: 0.0378 Steering Value: 1
42["steer",{"steering_angle":1.0,"throttle":0.5}]
CTE: 0.0378 Steering Value: -0.186081
42["steer",{"steering_angle":-0.186080770000001,"throttle":0.5}]
CTE: -0.0755 Steering Value: 1
42["steer",{"steering_angle":1.0,"throttle":0.5}]
CTE: -0.0755 Steering Value: -0.118093
42["steer",{"steering_angle":-0.118093220000001,"throttle":0.5}]
```

The vehicle successfully driven a lap around the track, and no tire leave the drivable portion of the track surface. The car didn't pop up onto ledges or roll over any surfaces.

Conclusion

While PID controllers are extremely important in the industry, this static version is not good enough for a production self-driving car. The final results we have after optimizing the hyperparameters is a solution that may very well be useful for a single application but will fail to react to changes in the environment and mechanical failures of the car - such as heavy winds or a steering bias being introduced after hitting a pothole.