Writeup for PID controller Project

Part of Udacity self driving car nanodegree

Sven Eriksson

September 11, 2017

Text in headings and in bold comes from the project rubric. My answer to the question are written just underneath.

1 Compilation

Your code should compile. 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.

I have not changed the CMakeList.txt and the code complies outside using a terminal outside CLion (the IDE I am using).

2 Implementation

The PID procedure follows what was taught in the lessons. It's encouraged to be creative, particularly around hyperparameter tuning/optimization. However, the base algorithm should follow what's presented in the lessons.

The PID algorithm located in the file PID.cpp is the standard PID that was taught in the lesson. Some additional errorvalue calculations has been added for optimization purposes but these errorvalues are not used for the actuator regulation.

The PID object is called through a new class and object called Twiddle. This allows me to automaticly restart the simulator after a certain number of timesteps and test a new set of P, I, D values.

3 Reflection

3.1 Describe the effect each of the P, I, D components had in your implementation.

Student describes the effect of the P, I, D component of the PID algorithm in their implementation. Is it what you expected? Visual aids are encouraged, i.e. record of a small video of the car in the simulator and describe what each component is set to.

While visual aids are encouraged (but not a requirement), I will not use them as I prefer describing the system in text.

As I had no easy way to find dt, the error of I and D were calculated assuming a dt of 1. Final P, I, D values were P = 0.208994, I = 0.00240444, D = 2.41875.

Effects of P:
This factor steers proportional to the distance from the middle in the y direction. If the car is left of the middle it steers right and if the car is right of the middle it steers left. This is expected and the purpuses of this factor in the regulator. Without the dampening of the D factor this creates an unstable oscilation that increases in amplitude as time goes on. That the P factor might create an unstable osciliation is expected. The high P is helpful for being able to take the curves more easily without leaving the track. A lower P value would not oscilate as much but might also have more problems in the sharper curves.

Effects of I:
When I set this to 0 do not see any large difference, but with the small final value of I that is to be expected. I guess there is no or very small error with the simulated vehicle steering actuator. The only 'error' that occurs during almost the entire track is the near constant left turn. The integrator value might help with smoothening the path by steering slightly more to the left due to haveing been on the right side the majority of the time.

Effects of D:

This conteracts movement in the y direction. When moving left it applies a steering to the right and when moving right it applies steering to the left. The amount of steering is proportional to the speed in y direction relative the middle of the road. This plays a significatant role of dampening the steering as the car would oscillate with increasing amplitude around the center and soon leave the track if this was set to 0 (has been tested). This behavior is expected and the derivative term is usually used for dampening the oscilations of systems.

4 Describe how the final hyperparameters were chosen.

Student discusses how they chose the final hyperparameters (P, I, D coefficients). This could be have been done through manual tuning, twiddle, SGD, or something else, or a combination!

I did some initial test with manual tuning and did then use that as a starting point for my implementation of the twiddle algorithm that started with relativly large steps for each factor.

Staring points: P = 0.3, I = 0.003, D = 3. Initial step sizes: dP = 0.2, dI = 0.002, dD = 2.

The code responsible for finding better P, I, and D values are located in the Twiddle.h and Twiddle.cpp files. The cost-value used in the twiddle algorithm was:

$$\sum_{t=init}^{final} (d_t)^2$$

Where d is the distance from the middle. The twiddle algorithm searched for a lower cost-value. The intial and final timestep for this summation was choosen so that slighty more than one lap had been traveled with a speed of 50mph.

5 Simulation

The vehicle must successfully drive a lap around the track. 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).

When driving in the simulator on my computer the car does succesfully complete several laps without leaving the track surface.