PID Project Reflection
Dan Bergeland
Udacity Self Driving Car Nanodegree
Term 2, Project 4

What is the effect of each component of the P,I,D components?

The proportional (P) part of the control is the direct steering effort as a result of how big the cross tracking error (cte) is. By increasing the gain of the P parameter, it will steer more aggressively towards the center line if it gets off track. If this is pushed too high, then the car will overshoot and weave back and forth around the center line.

The integral (I) part of the control corrects for steady state errors. The integral accumulates the cte. This is best to allow for minor off-center error to accumulate, which will then cause the car to react and steer towards the center line.

The differential (D) part of the control responds to the change in CTE between iterations. This allows for correcting some of the dynamics of just the proportion gain. By responding to the change in cte, it causes the car to steer back towards the middle when the cte is increasing (d cte/dt is positive) and steer straighter if the cte is decreasing (d cte/dt is negative).

How were hyperparameters chosen?

The PID coefficients were found through a combination of manual tuning and using the twiddle function.
The twiddle function I made depends on the car staying on the track continuously. I started by picking some round numbers like PID = { .1, .01, 1}, and tuning them up and down by whole numbers to get a feel for the change in steering. I eventually got the car going around the track, through manual tuning alone, with numbers of PID = {.2, .004, 2}. I wanted to attempt to implement the twiddle function, though.
Once the car was running around the track reliably, I built a twiddle class. The concept of the class is that it accumulates CTE over the course of a long enough episode of driving to get a sense of the quality of the system. At the end of the episode, it will check to see if the error has gone down and update the parameters. I'm not familiar with or aware of an API for the simulator, so I took advantage of the fact that the track is a circuit and I can compare lap to lap performance.
The idea is that each lap could be considered a simulation run, then the twiddle function checks the total error to determine if the new parameters have improved handling (minimizing accumulated CTE). While it's accumulating, the twiddle function adds the absolute value of the CTE. Intuitively, this penalizes weaving, steady state error, or missing turns.
I implemented the math just as Dr. Thrun described in the video lectures. First, it runs a lap with the selected parameter increased by a given step size, checks, then decreased, and increases or decreases the step size based on if the error improved or not. The step sizes were initialized to be 10% of the starting parameter. This became a useful setting as the twiddle values were used as starting points for each incremental speed.
Twiddle frequency is kind of an interesting parameter. The twiddle can iterate faster with fewer updates, but it would be comparing different parts of the track, so it might reject an improved parameter because it's going around a corner. I chose longer periods. I ended up using 1200 updates

between updates. It doesn't need to be exactly a lap, as long as the episode time is long enough to capture enough performance to grade it.
Throttle was also an important factor. The first one I ran at .2, then with improved parameters, I increased the throttle to .3, then .35, then .5, then .65 etc… Error accumulates much faster at the higher speeds, so the twiddle function can be repeated again at higher speeds and it will hill climb to a new minimum error state. By increasing the speed, this should eventually result in a system with fast response time and low overshoot, which are both highly desirable characteristics.
To log the activities, the PID values and associated errors for the episodes were saved to a text file. The final tuned value is hard coded in main.cpp.

The very last thing I added was an adaptive throttle. There are a few parts of the track that have sharp turns. The CTE shoots up at these points, so I decreased the throttle when the car gets too much CTE. It's a simple linear reduction of the throttle based on a maximum allowable CTE that was defined by manual tuning. This allows the control system a little more time to respond to the high error.

While this method of twiddle is time consuming, it's not nearly as labor intensive as manual tuning. I noticed the documentation on the website about the project was incorrect. The speed limit is only 50 mph, not 100. This PID, with adaptive throttle, was able to navigate the course at 50 mph.