PID Controller Project

Udacity Self Driving Car Engineer Nano-Degree (Term 2)

Michael Palazzolo

Introduction:

For this project, the goal was to implement a PID controller to control a simulated vehicle as it negotiates a track environment. We used what is called the cross track error (CTE) which described more plainly is the later offset of the vehicle from the center of the lane.

Implementation:

To achieve the project target we implement a PID controller (Proportional, Integral and Derivative). The three elements serve distinct purposes and tuning them appropriately can lead to safe and comfortable control of the vehicle.

Proportional Control:

The proportional or P term acts on the proportional error of the target variable relative to the ground truth target. The tendency of this controller on its own will be to overshoot the target and cause a resonance in the system.

$$Proportional\ Error = K_P * Error$$

Integral Control:

The integral control element can be used to reduce a signal drift that is present in the system. It represents the accumulated error present in the system over each timestep. Were the system to continue in a path away from the target, eventually enough error would accumulate at the system act to return towards the it's nominal state.

$$Integral\ Error = K_I \sum error$$

Differential Control:

Differential control acts on the rate of change of error in the system response. Sudden system changes will cause the differential control error to dominate the system and return towards the nominal state.

$$Differential\ Error = K_d \frac{d}{dt} Error$$

PID Controller Equation:

Bringing it all together we get -

$$Total\ Error = K_P * Error + K_I \sum error + K_d \frac{d}{dt} Error$$

Effect of tuning paremeters on the system response in simulation:

Initially I started with just a P controller in order to see how this would affect the vehicle. I noticed that an unstable resonance would occur, and the vehicle wouldn't make it beyond the first turn of the lap before crashing. At this point I brought in the differential error term and noticed that it would damp out the oscillations causing the car to become more stable. The car still wasn't able to complete a full turn an I noticed a drift in the error at certain locations that I couldn't correct with just the P and D terms. By including the I term the vehicle navigates along the center of the lane as I had anticipated that it would.

Selection of Hyper Parameters:

I did not use an algorithm to select the terms because I wanted to gain the intuition on how to generate them myself first. I started by setting the P term to about 25% of the D term and seeing how the vehicle would navigate the course, further tuning was needed after that to balance the terms. It became clear to me that I would need to introduce the Integral error element in order to make the route, there was some sustained cross track error that I wouldn't clip without using the Integral term. I approached it iteratively by holding 2 out of the 3 variables fixed and making changes to remove issues I saw in the vehicle response until I landed on the parameters below.

Final Parameters:

P = -0.19

I = -0.00035

D= -3.0

******CALLING THE PID CONTROLLER FROM BASH:**

For my project, you must call the PID command as follows:

- From the /build folder type cmake ..&& make
- Type ./pid -0.19 -0.00035 -3.0

In Summary:

I was able to successfully tune the vehicle PID controller to navigate the track at 30 MPH, I really enjoyed this project and it allowed to gain an intuition on how the tuning works in an implementation that is relevant to the self driving car space.