**Project 4 : PID Control for lane-centering**

**Objective**

In this project, we are assigned to design PID controller that stabilize simulator vehicle such that it drives on lane-center throughout simulated route.

**Design Procedure**

The PID controller is designed according to equation below:

(1)

Where,

err(t) = 0 – cte

Note: cte is defined as distance from route centerline toward position of simulated vehicle.

Our task is to figure out Kp, Ki, and Kd terms that will make vehicle stable and drives around on center-line for most of time.

In the actual implementation, integral term and derivative term of eq 1 above is discretized according to formula below:

(2) , is accumulated integral error

from previous time-step

(3)

We assume that TimeElapsed is equal to 1 second in this project, since the information isn’t provided by simulator during simulation time.

**Tuning Procedure**

During initial step, I perform manual tuning for Kp, Ki, and Kd to get a general “feel” how host vehicle will perform. Based on my observation, I will summarize effect of Kp, Ki, and Kd in “common” language below:

1. Kp provides majority of correction for host-vehicle to stay in center-lane. However, it lacks of granularity to control smooth trajectory. Therefore, performance of Kp without additional parameters will tend to oscillate vehicle and will cause vehicle goes off-track.

(See PControlOnly.mp4 video).

1. Kd provides damping effect to oscillations caused by Kp. From my observation, Kp and Kd will be minimum required parameter to do lane-centering. However, it still lacks of granularity to remove bias error. From simulation, I can see this effect when during in curved route. The vehicle tends to stay longer off-center after it exits the curve. (See PDControlOnly.mp4)
2. Ki is final term that provides “granularity” that Kp and Kd terms can’t provide. It applies incremental correction to the bias error such that over-time it will become zero.

To facilitate “manual” tuning I modify main code to use separate thread, where I can manually update Kp, Ki, and Kd terms without the need to stop simulation. This process sped up the tuning process, by allowing me to receive instantaneous visual feedback every-time I change the parameters.

Final value for Kp, Ki, and Kd after the tuning is:  
Kp = 0.2, Ki = 0.1, and Kd = 30

In addition to manual tuning, I attempt to implement “twiddle” tuning in “PID” member function “Twiddle”. The first challenge that I see is the original twiddle algorithm that is provided during lecture, assumes that it has full control over when simulation should be run and stopped. However, the simulator doesn’t provide signals to stop or start simulation within websocket interface in the main-code. Therefore, I modify original twiddle algorithm to use internal counter to determine when to start / stop collecting data. The counter will be internally reset when enough data has been collected, then restarted after twiddle mechanism changes one of the PID parameters.

Unfortunately for my case, result from twiddle isn’t satisfying. Instead of converging into “optimal” parameters, the parameters just bouncing up / down from initial values. Therefore, I think I will stay with result from “manual” tuning process.

In the future work, I plan to do two things. First, further checking into “modified” implementation of twiddle algorithm to make sure I don’t make bug that produces parameters “jittery” behavior as described in paragraph. Second, I plan to implement gradient-descent algorithm to tune the parameters. I suspect parameters tuned by twiddle will converge too slow, because it modifies one parameter at a time. The “gradient-descent” will modify all parameters on each iteration in steepest direction toward “minimum square error” (L2) value. Thus, it should converge much faster than twiddle.