Udacity Self-Driving Car Nanodegree

Project 4 - PID Controller

Contents

1 Reference to Project Code and Files

Here is a link to my *project code and files*. My project includes the following files:

- **Main.cpp**, 2 functions (*link*):
  - To pass steering and throttle values to the simulator.
  - To look for optimal hyper-parameters P, I and D.

- **PID.cpp** (*link*):
  - To initialize and update the differential and integral terms of CTE (Cross-Track Error).
  - To calculate steering value from CTE.

- **PID.h**: header file for PID class (*link*).

- **Test videos**: to illustrate the effect of each parameter (*link*).

- **Project_4_writeup.pdf** summarizing the results (*this memo*)

As a result of this implementation, the vehicle is able steer safely around the track while optimizing its hyper-parameters P, I and D in real time.

2 Finding Optimal Hyper-Parameters

I started by **initializing** P, I and D hyper-parameters to respectively 0.2, 0.004 and 3.0 using the values provided by Sebastian as part of the course. Ideally I would rather start with zeros for all three parameters but the car going off track would put an immediate stop to our parameter optimization.

Then I implemented **twiddle** in the main.cpp file:

- The course of the car is driven by the **simulator**, so we do not have the flexibility to stop and restart the streaming at will.

- Therefore I defined **a twiddle interval**, i.e. an interval of web socket events, during which I was letting the simulator running without touching hyper-parameters.
    - I set this interval to 15 web socket events.

- Every 15 web socket event, the program performs a **twiddle checkpoint**:
    - It tries **new values**, based on incremental variations from prior values.
    - Then after letting the simulator run, it compares the current **cumulated squared CTE** to the best cumulated squared CTE to date.
    - Depending on this comparison, it adjusts hyper-parameters, reverts prior changes, or refines the research by narrowing down to smaller incremental variations.

- I also defined a larger interval, as a multiplier of the aforementioned twiddle interval, to **switch between parameters** so that each hyper-parameter gets optimized.
    - I set this larger interval to 6 twiddle intervals.
    - So the program switches to the next parameter every 6 * 15 = 90 web socket event.

- I measured the optimization process by defining a notion of **gain**, as following:
    - Incremental variations are stored in the **twd_dK** vector, and are narrowed down to smaller values every time the algorithm improves its cumulated error, as explained above.
    - Then the **twd_weights** vector is used to assign a weight to each initial value of twd_dK.
    - Finally the gain is defined as 1 / dot product of (twd_dK * twd_weights).
    - Its initial value is 1.

- This gain rises when the algorithm finds better values. Then it stops when the gain reaches a threshold of 6.

- Ideally this process should be run when the car is driving in **straight line**, until optimal parameters are found. But we do not have control over the track.
    - So I set the **throttle value** to a small number when “twiddling” (less than 10mph).
    - As the algorithm improves its gain, it gradually **increases speed**.
    - When optimal parameters are found, the speed it stabilized to a default value.

This implementation of Twiddle could be qualified as a logical permutation of the algorithm presented by Sebastian: instead of running multiple iterations of the model at will, which we technically cannot do here, we are letting the model run and inserting **logical flags** to improve it **on the fly**.

3 Analyzing the Effect of Different Parameters

As mentioned in section 1, the project includes a final video showing the car completing a lap around the track, using its optimal hyper-parameters: P = 0.1331, I = 0.00201008, D = 3.0.

To visualize the effect of each parameter, the project also includes 3 test videos, each captured after including parameters gradually.

- *PID_P.mp4* file: keeping **proportional term** (P) only.
    - *Note: I did not record a video without term P as it would result in the car not being controlled at all.*
    - This simulation results in the car constantly **overshooting the middle** line of the track. It follows a zigzag motion and eventually goes off track even before the first major turn.
    - It shows that P alone is not enough to steer the car correctly.

- *PID_PD.mp4* file: including both **proportional** (P) and **differential** (D) terms.
    - This time the car stops zigzagging and better follows the track.
    - It shows how term D helps the car **stick to the middle** area of the track.
    - But at this stage the car still steers a bit off the middle.

- *PID_PID.mp4* file: including all **proportional** (P), **integral** (I) and **differential** (D) terms.
    - Adding the integral term helps steering in the **center** of the track.

*Note: I used **VLC Media Player** as a workaround to capture my desktop as I did not know how to save images from the simulator.*