Term2: Project4 - PID Controller

In this project, the P(proportional), I(Integral) and D (Differential) co-efficients of a PID control need to be optimized such that the Cross-track error(CTE) is minimized.

The PID algorithm is run on a simulated track provided by Udacity which provides CTE based on the location of the car in each frame.

1. **Attachments**: Attached updated PID.cpp, PID.h and main.cpp files along with a small recorded video clip. Grayed out the Twiddle algorithm lines in the submitted files. Note that there was some extra latency induced on the processor when the simulator was run simultaneously with video recording. It looks better when run alone.

2. **Methodology**:

a) Eventhough I was initially interested in using the Twiddle algorithm used in the Udacity lessons to fine-tune the PID co-efficients, I realized that there was no easy way to iterate the Twiddle loop over a part of the simulator track repeatedly. So, I applied a modified version of the Twiddle algorithm to update the PID co-efficients from time steps 100 to 200. Even though not as efficient as the original Twiddle algorithm, it did help me narrow down the values to [0.1, 0.0002, 2.9]. This modified version takes in the CTE at each step and compares it against the Best_CTE or previous CTE and updates the co-efficients accordingly.

b) Also some tuning had to be made to the throttle, to take it as a function of Steering_value, to slow down the vehicle during hard turns. Restricted the Steering_value between -1 and 1 using a MaxMin function to remove high oscillations.

3. **Effect of Co-efficients**:

Kp: As the track in the project involves many curves and not a straight line as seen in the udacity lessons, you are bound to have some CTE whenever the track's curvature changes. As Kp is directly associated to the CTE itself, I realized that it's

value needed to be brought down to prevent high oscillations yet big enough to make the vehicle sharp turns when necessary.

Kd: A relatively high value of Kd penalized the difference between CTE and previous CTE. So, it helps the vehicle correct immediately when there's a big difference between two consecutive CTEs.

Ki: As Ki is multiplied to the sum of all previous CTEs(i_error), I wanted to make sure that this portion of the steering value doesn't get large enough over a period of time. So, kept Ki as low as possible and also monitored i_error to make sure that the positive and negative CTEs keep canceling out and stays relatively low.

Conclusion: There seem to be many other interesting ways to fine-tune the parameters and even include speed as a function of CTE to make the vehicle go much more smoothly. One approach of using gradient descent to update the coefficients from this blog(https://medium.com/towards-data-science/tuning-pid-controller-parameters-with-backpropagation-c42c6f80d3cd) caught my attention, and I hope that concept if properly applied can yield better results.