PID Controller Project

The goal and objective of this project is to implement a PID controller and tune the PID parameters for controlling a vehicle around a track. The simulator provides a cross track error (CTE), speed and steering angle data. Two separate PID controller are designed for steering and throttle commands to drive the car reliably around the simulator track.

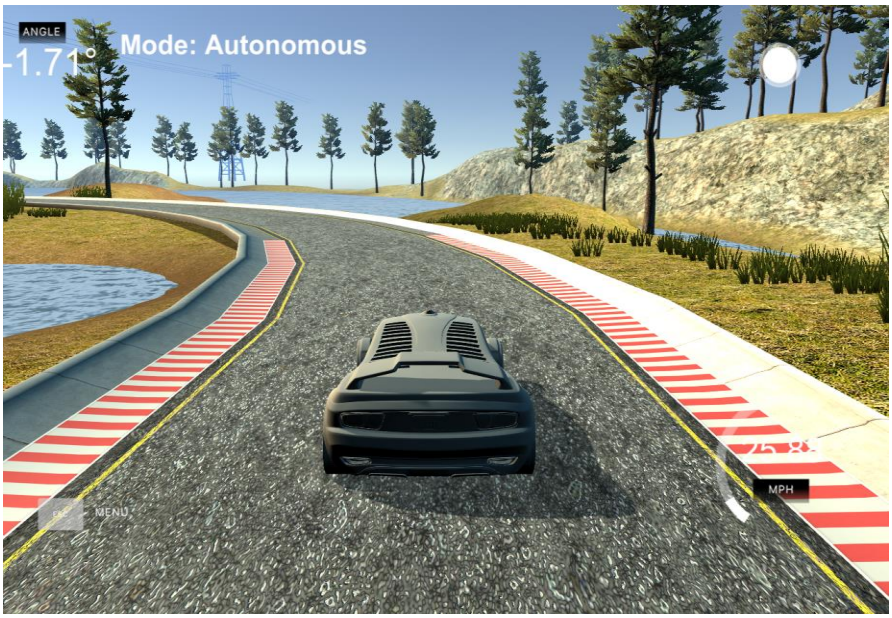

Fig 1. Snapshot of a car being controlled across the track.

Based on the information obtained from the simulator, PID controller was implemented for steering and throttle. Steering command is governed by the steering PID whereas the throttle command is combined by a combination of steering PID and a throttle PID. This is implemented with a few of reducing throttle when turning or doing high steering maneuvers.

PID is a simple controller concept designed to reduce cross track error to zero with the help of a proportional, integral and derivative control. The objective of this project was to drive in a stable manner around the track. Due to lack of time and simple implementation of both speed and steering control, the performance is not the focus of this submission.

To tune the hyperparameters, A combination of manual tuning and twiddle was used. Twiddle is implemented as explained in the lessons to tune the PID controller.

Compiling:

Code must compile without errors with cmake and make.Given that we've made CMakeLists.txt as general as possible, it's recommended that you do not change it unless you can guarantee that your changes will still compile on any platform.

The code compiles without errors. I have not made any manual changes to the CMakeLists.txt file. I had used the instructions for using Ubuntu BASH on Windows and have written and coded using the terminal on windows.

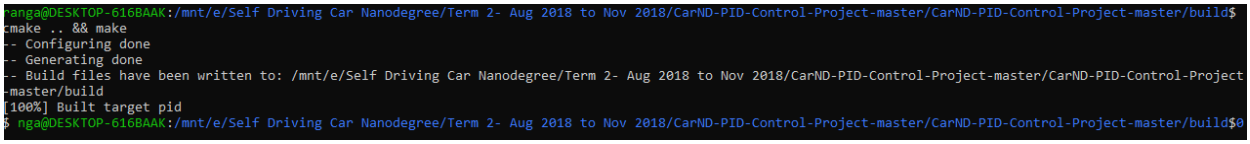

Fig2. No Compilation Errors when using Cmake.. && Make

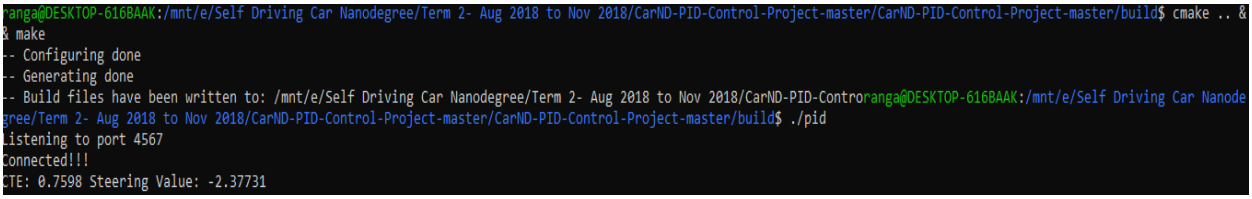

Fig3. Connects to simulator when the built executable is run

Implementation:

It's encouraged to be creative, particularly around hyperparameter tuning/optimization. However, the base algorithm should follow what's presented in the lessons.

The base PID algorithm is implemented as explained in the lectures for both steering and throttle. The tuning of the hyperparameter is also done through a combination of manual tuning and twiddle methodology.

The throttle command provided to the simulator is not exactly the output of PID but a combination of Steering PID and a Throttle PID.

Reflection:

Student describes the effect of the P, I, D component of the PID algorithm in their implementation. Is it what you expected? Visual aids are encouraged, i.e. record of a small video of the car in the simulator and describe what each component is set to.

The actual implementation of code for a basic PID controller is straightforward, but making the controller perform well is the tough part.

- The Proportional controller means that the car will steer in proportion to the cross-track error (CTE). CTE is the distance of the car from the middle line of the road. The proportional component helps steer the car in the opposite direction of CTE. This has the most directly observable effect on the behavior of the car. However, the car is extremely sensitive to higher P hyperparameter. If the coefficient is too high, the car will steer and overshoot in the opposite direction causing high oscillating steering commands. If the coefficient is too low, the car may react too slowly along the curves.
- The Integral controller is designed to reduce the sum of all CTE to zero. This is used to achieve steady state tracking of any reference and reduce the error/CTE to zero. In this case, CTE is zero when car is in middle of the road. This controller is useful to counteract any bias in CTE due to any steering drift. The I hyperparameter also has an impact on the stability of the PID implementation. If coefficient is too high, the car tends to have quicker oscillations. A lower coefficient will not help the car fix the drift in CTE and car tends to be towards either sides of the lanes.

- The Derivative controller is designed to reduce change in consecutive values of CTE. This controller helps reduce the oscillations caused by either P or the I controller to an extent. It performs the behavior of a damper to the PID system. It helps smoothen the command once reaching the middle lane of the road. However, even the D hyperparameter is sensitive and impacts stability to an extent. It helps improve the stability of the system by reducing oscillations, but higher values of derivate gain might cause the system to be highly damped causing the P and I controller outputs to destabilize the system. In our case, high gain causes almost constant steering angle whereas low gains do not help address the oscillations due to the P and I controller.

Student discusses how they chose the final hyperparameters (P, I, D coefficients). This could be have been done through manual tuning, twiddle, SGD, or something else, or a combination!

Hyperparameters were tuned manually at first. This was necessary because of the sensitivity of the system to wrong initialization values and lac of good information since the car leaves the track. This does not help any optimization procedures such as twiddle to work.

Once an approximate tuning was achieved manually, twiddle was implemented. Twiddle as much as it is helpful in auto-tuning had mixed results. Parameters were changed and edited every half lap and observations were made on overall stability of system and the ability to course around lap without touching the sides of the track. Multiple twiddle iterations were performed to fine tune some calibrations.

Towards the end however, twiddle helped me get to a value with more resolution. Visual aids were used to observe performances of PID calibrations around the test track and some calibrations were modified manually to fix some of the obvious errors in certain curvy regions around the track. This was essentially needed since the throttle command was modelled as a combination of Throttle and Steering PID. This is nowhere close to an ideal setup due to individual tuning of hyperparameters of 2 PID's which are related to one another. The solution came out to be satisfactory for a crude PID implementation.

Simulation:

No tire may leave the drivable portion of the track surface. The car may not pop up onto ledges or roll over any surfaces that would otherwise be considered unsafe (if humans were in the vehicle).

The simulation video attached satisfies the above-mentioned criteria. The performance could have been better, and this video is just a crude solution submitted due to time constraints.