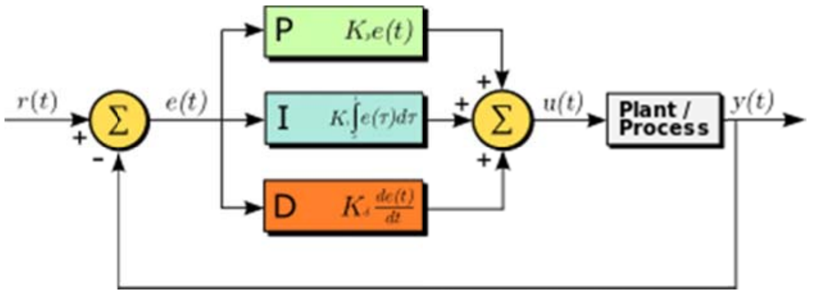
Introduction:

In this project, a PID controller is used to be tuned and control the steering of a vehicle in simulator. The input to PID controller is CTE(Corss Track Error, which is a measure from hthe lane center) and steering angle, and PID outputs steering angle in order to track the road in the simulator.

Below image shows the block diagram of a PID controller. (Source: Wikipedia)



- Term P is proportional to the current value of the SP – PV error e(t). For example, if the error is large and positive, the control output will be proportionately large and positive, taking into account the gain factor "K". Using proportional control alone will result in an error between the set point and the actual process value, because it requires an error to generate the proportional response. If there is no error, there is no corrective response. In our project, the P is accountable to steer proportional to the distance from the lane center (e.g., steering bigger if car is further from the center line and vice versa).
- Term I accounts for past values of the SP – PV error and integrates them over time to produce the I term. For example, if there is a residual SP – PV error after the application of proportional control, the integral term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error. When the error is eliminated, the integral term will cease to grow. This will result in the proportional effect diminishing as the error decreases, but this is compensated for by the growing integral effect. In this project, it looks that I accounts for reducing the CTE around curves and probably counteracts biases (e.g. steering drift) on CTE which in case of only PD controller could cause the vehicle not to reach the center line.
- Term D is a best estimate of the future trend of the SP – PV error, based on its current rate of change. It is sometimes called "anticipatory control", as it is effectively seeking to reduce the effect of the SP – PV error by exerting a control influence generated by the rate of error change. The more rapid the change, the greater the controlling or dampening effect. In this project, D avoids overshoots caused by P portion of the controller and helps the vehicle to come close to center line in a smother way.

Project Rubric:

CRITERIA	MEETS SPECIFICATIONS
Compilation: The code compiles correctly.	Code must compile without errors with cmake and make. Given that we've made CMakeLists.txt as general as possible, it's recommend that you do not change it unless you can guarantee that your changes will still compile on any platform. No Errors in the compilation, cmake, and make
Implementation: The PID procedure follows what was taught in the lessons.	It's encouraged to be creative, particularly around hyper-parameter tuning/optimization. However, the base algorithm should follow what's presented in the lessons. In PID.cpp, there are two major sections: - Updating the error in which P, I, and D errors are calculated. - Total Error in which uses P, I, and D parameters and the errors for each P, I and D and calculates the total error In main.cpp, first update error for cte is calculated, and then the steering value is calculated by subtracting from the total error calculated.
Reflection: Describe the effect each of the P, I, D components had in your implementation.	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 "P" is accountable to steer proportional to the distance from the lane center (e.g., steering bigger if car is further from the center line and vice versa). Using a controller with only P parameter, caused several fast reactions and overshoots, and finally vehicle went off the road. I did try for several "only P" scenarios. See "only P" video for P= 0.2 • The "I" accounts for reducing the cte around curves and probably counteract biases (e.g. steering drift) on cte which in case of only PD controller could cause the vehicle not to reach the center line. I alone caused the vehicle to drive in circle. See "only I" video for I= 0.5 • The "D" avoids overshoots caused by P portion of the controller and helps the vehicle to come close to center line in a smother way. D alone caused the vehicle. See "only D" video for D= 2
Reflection: Describe how the final hyper-parameters were chosen	Student discusses how they chose the final hyper-parameters (P, I, D coefficients). This could have been done through manual tuning, twiddle, SGD, or something else, or a combination! I used manual try and error method. First set all parameters to zero to see

	how vehicle drives and for the integrity test on the debugging, and connection to simulator (which as expected, the car drove just straight since there is no controller to update the steering angle). Then starting by only tuning only P form very small parameters such as .01 to 3.5 randomly, then playing with only I, then tuning only D with several try and errors. Finally tuning with P,I, and D together. For instance based on vehicle reactions on curves I realized to add more to P if it was not fast enough that caused the vehicle to touch the side lines, or reduce P if I saw fast reactions that caused the vehicle to go out of the road on the other hand. and my final parameters were set as below:
	P=0.2
	I=.0002
	D=2.0

Simulation: The vehicle must successfully drive a lap around the track.	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 vehicle drives relatively well and encouraging with no tires touch the side lines ever. See video “final parameters”

CarND-Controls-PID

Self-Driving Car Engineer Nanodegree Program

Dependencies

- cmake >= 3.5
- All OSes: click here for installation instructions
- make >= 4.1(mac, linux), 3.81(Windows)
  - Linux: make is installed by default on most Linux distros
  - Mac: install Xcode command line tools to get make
  - Windows: Click here for installation instructions
- gcc/g++ >= 5.4
  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - [install Xcode command line tools]((https://developer.apple.com/xcode/features/)
  - Windows: recommend using MinGW
- uWebSockets
  - Run either `./install-mac.sh` or `./install-ubuntu.sh`.
  - If you install from source, checkout to commit `e94b6e1`, i.e.
  - `git clone https://github.com/uWebSockets/uWebSockets`
  - `cd uWebSockets`
  - `git checkout e94b6e1`

  Some function signatures have changed in v0.14.x. See this PR for more details.

- Simulator. You can download these from the project intro page in the classroom.

Fellow students have put together a guide to Windows set-up for the project here if the environment you have set up for the Sensor Fusion projects does not work for this project. There's also an experimental patch for windows in this PR.

Basic Build Instructions

1. Clone this repo.
2. Make a build directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`

4. Run it: `./pid`.

Tips for setting up your environment can be found here

Editor Settings

We've purposefully kept editor configuration files out of this repo in order to keep it as simple and environment agnostic as possible. However, we recommend using the following settings:

- indent using spaces
- set tab width to 2 spaces (keeps the matrices in source code aligned)

Code Style

Please (do your best to) stick to Google's C++ style guide.

Project Instructions and Rubric

Note: regardless of the changes you make, your project must be buildable using cmake and make!

More information is only accessible by people who are already enrolled in Term 2 of CarND. If you are enrolled, see the project page for instructions and the project rubric.

Hints!

- You don't have to follow this directory structure, but if you do, your work will span all of the .cpp files here. Keep an eye out for TODOs.

Call for IDE Profiles Pull Requests

Help your fellow students!

We decided to create Makefiles with cmake to keep this project as platform agnostic as possible. Similarly, we omitted IDE profiles in order to we ensure that students don't feel pressured to use one IDE or another.

However! I'd love to help people get up and running with their IDEs of choice. If you've created a profile for an IDE that you think other students would appreciate, we'd love to have you add the requisite profile files and instructions to ide_profiles/. For example if you wanted to add a VS Code profile, you'd add:

- /ide_profiles/vscode/.vscode
- /ide_profiles/vscode/README.md

The README should explain what the profile does, how to take advantage of it, and how to install it.

Frankly, I've never been involved in a project with multiple IDE profiles before. I believe the best way to handle this would be to keep them out of the repo root to avoid clutter. My expectation is that most profiles will include instructions to copy files to a new location to get picked up by the IDE, but that's just a guess.

One last note here: regardless of the IDE used, every submitted project must still be compilable with cmake and make./

How to write a README

A well written README file can enhance your project and portfolio. Develop your abilities to create professional README files by completing this free course.