CarND-Controls-PID

Self-Driving Car Engineer Nanodegree Program

Objective

The goal of this project is to build a PID controller and steer the car around the simulator track. The cross-track error (CTE) is available in the simulator environment. The PID controller is expected to provide steering and throttle values to drive the car without falling-off the track.

PID Controller

Proportional component

This is the main component responsible for reducing the CTE by setting the steering angle as negative of scaled CTE value. If I and D values are set to zeros, it is possible to steer the car slowly with P component alone. But there will be oscillations and sometimes the car falls off the track due to overshoot

Differential component

The D component avoids overshoots by adding a component that scales with rate of change. This anticipatory control helps to settle smoothly.

Integral Component

The I component eliminates residual error. I was surprised to see that it did not have significant impact on steering or throttle PID control. I was able to get the car steer/throttle without the I component.

Ziegler–Nichols method

I used Ziegler–Nichols method (wikipedia source) to fine tune the PID controller. Somehow the twiddle optimization did not work and the car would go off the track during optimization.

I used two separate PID controller for steer and speed. The PID controller for steering was optimized first by setting constant speed of 0.3.

PID for Steering

I started with Kp=Ku=1.0, Kd = 0 and Kp = 0. The Kp=Ku was adjusted to get the car moving on the track without falling out of it. With some trial and error, the Ku of 0.7 provided the best result where car could steer until the first sharp turn. The oscillations were found to be about 3 seconds. This gives us Kd value of 0.7x3 = 2.1 and Kp was reduced to 0.2x0.7 = 0.14.
These values correspond to Ziegler–Nichols method with no overshoots (again courtsey wikipedia).

I did not find any impact of setting Ki except large values would result in slowing down. So, I used Ki = 0 for steering.

PID for Speed

Intuitively, the speed should decrease with CTE values and is symmetric wrt right/left CTE values. Hence absolute value of CTE was used. The speed/throttle was set as speed = 1.0 - PID_output_speed. The abs(CTE) was used as input to speed PID control. Started with Kp=1.0, Kd and Ki = 0. The trial and error gave Kp = 0.6. Since the PID for speed uses absolute(CTE) and also I did not notice much speed variation, I started with lower values of Kd and found Kp=0.4, Kd = 0.02 was good enough. However, there was excessive breaking at the curves. So, I reduced to max speed to 0.75.

Joint Speed and Steer PID Optimization

With PID values for steering and speed control as above, the car was able to drive around the track. However, there was large variation in the speed. So, slowly increased the Ki for steer to 0.00015, decreased the Kp for steer to 0.13 and decreased Kp for speed to 0.3.

Conclusion

The knowledge of plant model G(s) - in this case, response function of the car for steering and speed could have helped in better modeling of PID system and derive initial values of the PID controller analytically (i.e optimize the rise time/settlingtime/dampening factor) And then adjust/optimize the PID values around these analytical values.

Not sute if such modeling of vehicles for steering and throttle is available and would help in optimizing PID controller.

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
  - 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.

There's 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`.