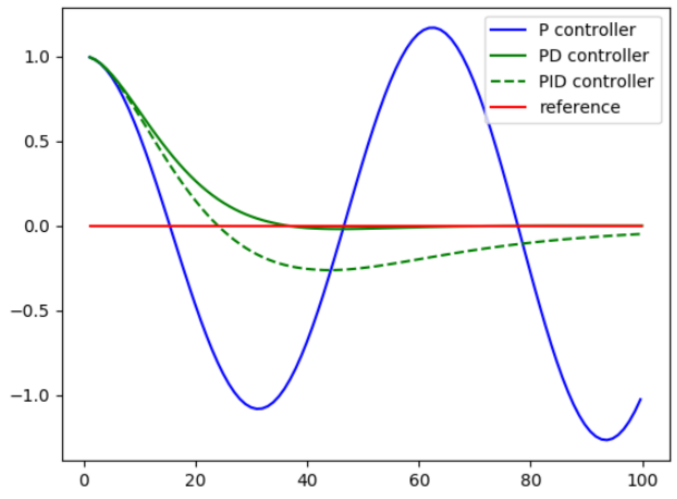
Writeup Report for PID Control Project

Introduction

PID stands for Proportional-Integral-Derivative. The evolution of PID controller starts from the intuition that steering in proportion to Cross Track Error suits best in controlling the car. Due to the nature of proportional control, the car would overshoot and oscillate around the desired path, therefore the derivative term is introduced to dampen the oscillation. Additionally, to solve the issue of systematic bias, the integral term is added, which seeks to eliminate the residual error after the application of proportional and derivative control.



Effect of each component

The proportional component of the controller is the basis of PID control, and it determines the output response to the input error, i.e. it makes the car turn if the road curves. When the car enters a curved section of the road and deviate from the center of the road, P component of the controller responds by increasing the magnitude of steering angle proportional to the deviation, which results in reducing the deviation and making the car return to the center. Increasing the P term value would increase the response speed of the car, but it would also increase the chance of oversteering the car, oscillating, and even driving off road surface.

The derivative component of the controller is responsible for keeping rate of change at bay, dampening oscillation and smoothening the steering. Since it is proportional to the derivative of the error, it is highly sensitive to noise and makes the system unstable. Thus, the value of D component is usually very small.

The integral component of the controller sums the error term over time, proportional to both magnitude and duration of the error, thus its existence drives the “steady state” error to zero. It is slow reacting at the start, and brutal as the response increases as long as the error is positive, regardless of whether the error is decreasing.

Parameter Optimization

Twiddling algorithm, also known as Coordinate Ascent, is used for tuning the hyperparameters of the PID controller. Unlike the robot example from the class, we cannot reset then rerun the car every time one of the parameters is slightly adjusted, the twiddling procedure is modified to keep track of its execution across each simulator message processing. From the pseudocode of twiddle, we can see that each iteration of tuning requires two runs of the robot, so similarly, we use two flags pre_first_run and pre_second_run to control the execution flow of twiddling logic (main.cpp line 97-175).

Moreover, from trials and errors, I learnt that in each run of the simulator, only one of the three hyperparameters should be tuned. Tuning all three in every run would only make the controller very unstable and drive the car off the road not long after started. From observation, the errors corresponding to different parameters differ greatly in magnitude. Therefore, using a single threshold as criteria of stopping twiddling does not work equally well for all three parameters. So in my implementation I simply use number of iterations as the criteria for resetting errors and delta for parameter adjustments, and re-initialize PID controller using the best parameters values so far.

Initially, I randomly chose one of the three components for tuning in the next simulator run, updated the initial value of that parameter with twiddling best result, then started tuning a different component. After about a dozen rounds of such runs, I realized the tuning results did not converge. The magnitude of all three parameters increased from round to round, and the performance of the PID controller did not significantly improve regardless of the tuning effort.

From more research on the PID control loop tuning, I learnt that these three parameters should be tuned in certain order. Ideally, we should start with all zero, and increase P until the response to a “disturbance” is steady oscillation, then increase D until the oscillation goes away. These two steps are repeated until increasing D does not stop the oscillation, then I is increased until a balance of response speed and number of oscillations is reached. In our scenario, the track of the simulator does not have much tolerance for oscillation with large magnitude, so I can only follow the general idea of this tuning process, stopping the run if the car goes off track, and re-initializing the parameter if it has made the car steering more stable and responsive to the change in road curvature.

Eventually, the car could drive around the track without going off road surface, although the small oscillations are still noticeable throughout the course.