Tuning PID Controller for Self-Driving Cars

Madhu Dev · Just now · 6 min read

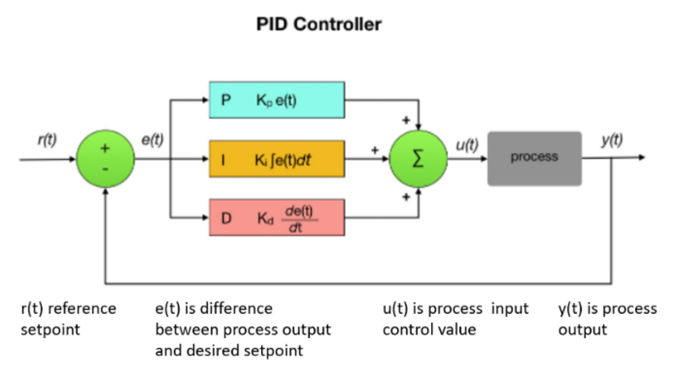
> *This project illustrates the concept of PID controller applied to self-driving cars as part of Udacity Self-Driving Nanodegree program*

A PID controller is a control loop feedback mechanism that calculates the difference between a desired setpoint and the actual output from a process, and uses the result to apply a correction to the process. PID Controller find wide range of applications in industrial and robotic process control applications.

In context of Self-Driving cars, they play a important role in controlling driving parameters like steering, throttle etc. The complex algorithms used in Self-Driving cars essentially calculate a trajectory and the velocity for self-driving car to drive. Autonomy can only be realized if the car follows the trajectory with the given speed. This is exactly where PID controller plays it role to ensure the self-driving car adheres to the computed parameters. Any deviation from the computed parameters means unforeseen or catastrophic results.

1. Theory of PID

The PID term consists of three terms represented by *Proportional, Integral, Derivative*. The figure below shows the relation between all three terms and the final response.



PID control Process

Proportional(P): This simply takes some proportion of the current error value. The proportion is specified by a constant and is represented by the letters *Kp*. This is used to compute the corrective response to the process. Since it requires an error to generate the proportional response, if there is no error, there is no proportional part of the corrective response.

Integral(I) : This takes all past error values and accumulates them over time. This results in the integral term growing until the error goes to zero. When the error is eliminated, the integral term will stop growing. If an error still exists after the application of proportional control, the integral term tries to eliminate the error by adding in its accumulated error value. This will result in the proportional effect diminishing as the error decreases. The integral constant is denoted by *Ki*

Derivative(D): The derivative term is used to estimate the future trend of the error based on its current rate of change. It's used to add a dampening effect to the system. The more rapid the change, the greater the controlling or dampening effect. The derivative constant is denoted by *Kd*

The input to the Controller is a error term e(t) which is then multiplied with the corresponding constants and summed up to arrive at the process input. The difference between Process output and the setpoint reference is again the new error and sent back to the Controller again for corrective response.

2. Goal of the Project

- Create a PID control algorithm to guide a vehicle around a track.
- Tune PID hyperparameters (Kp ,Ki ,Kd) so the vehicle smoothly follows the road, minimizing the cross-track error.

The virtual Self-Driving is available as a simulator provided by Udacity. The simulator takes in steering angle and throttle at definite frequency and moves the car accordingly as well as returning the cross track error (cte) for the PID controller.

3.Implementing the control algorithm

The controller algorithm was implemented as C++ code. Source code consists of PID class in file PID.cpp The initial values of error and PID Hyperparameters are initialized by the function Init(). The cross track error returned by simulator is used to update error terms for P, I and D terms namely current error, sum of errors and difference in error in *UpdateError()* function. The function *CalculateResponseValue()* will calculates and returns the PID response based on current cross track error values.

4.Tuning PID Hyperparameters

There are many ways a PID controller can be calibrated or tuned. Apart from manual tuning, it is also possible to tune it automatically using Twiddle algorithm. In this project, i have used a PID controller to control the steering and a PID controller to maintain the desired speed of 30MPH.

4.1 Tuning hyperparameters for steering control

Tuning Kp : This value was tuned by starting at a small value and gradually increasing until the vehicle began swerving back and forth, showing the oscillation of the P term. I started at value 0.05 but settled at 0.15 as a appropriate value. The plot below shows the oscillations of error for different values of *Kp*. With increase in Kp value, the oscillations increase frequency and amplitude which has to be dampened.

The video below shows the violent oscillations of the car after the start with only *Kp* value of 0.15

Tuning Kd : With *Kp* value set at 0.15, I increased the *Kd* value until the oscillations stopped. I chose a value of 3.0 ensure a smooth ride. The plot below shows the curves for different *Kd* values. The red curve represents the value 3.0. Its interesting to note the oscillations are not damped until the value reaches 2.0. The curve for 2.0 and 3.0 looks very similar but the value 3.0 performed well at curves where the error gradient is very high.

The video below shows the drive with Kp and Kd values set at 0.15 and 3.0

Tuning Ki : *Ki* values theortically eliminates the steady state error. However leaving the value close to zero (1E-5) produced the best results.

4.2 Tuning hyperparameters for speed control

As a first step the *Kp* value was gradually increased from 0.1 to 0.25. But as shown in the plot below, the curve settled at a steady state error for 0.25(The gap between reference line and blue line). Since there was no oscillations to dampen , i left the *Kd* value zero. To remove the steady state error, i set *Ki* value of 1E-4, which eventually maintained the desired speed of 30 MPH.

5. Conclusion

With the selected *Kp, Ki, Kd* values for steering and speed control, i was able to guide the car through the track without the car leaving the track or making any dangerous maneuver. The reason i used manual tuning is to understand the inside working of PID controller and how the values influence the vehicle behavior.

The Following are my conclusions :

1. *Kp* value had significant influence on vehicle behavior since it was related directly to the error. Very high value of Kp will make the car swerve wildly at curves. Lower Kp values lead to smoother steering corrections. However only *Kp* without *Kd* and *Ki* value's will result in oscillations.
2. *Kd* value is required to dampen these oscillations. A wide range of *Kd* values are suitable since its influence is dependent on error gradient and not the error itself. Whenever the vehicle navigates a sharp turn, the error increases. A combination of good *Kp* and *Kd* values reduces the error and dampen oscillations for gaining control back. If steady state error is present, then *Ki* value is required.
3. Tuning these values manually is quite challenging. Some values selected performed well in certain areas of the track but failed in other areas. Automated tuning algorithms like Twiddle which tunes by varying all the three hyperparameters in unison would be a good alternative.

Pid Controller · C Programming · Self Driving Cars · Robotics · Simulation