

PID Controller

Student: Guido Rezende de Alencastro Graça

Project Goals:

This is the report of the “PID Controller” project of the Self Driving Vehicles course on Udacity. This project goal is to build a code that is able to drive the autonomous vehicle in a simulator course, calculating the steering of the vehicle at every timestep. This calculation uses a PID controller and the Cross Track Error (CTE) given by the simulator. The criterias for passing are the following, as detailed on the *Project Rubric*:

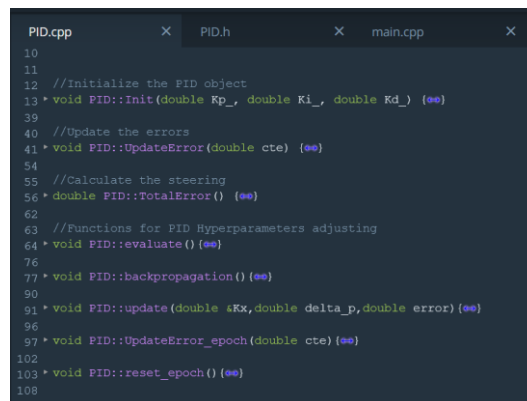
1. The code should compile.
2. The PID procedure follows what was taught in the lessons.
3. Describe the effect each of the P, I, D components had in your implementation.
4. Describe how the final hyperparameters were chosen.
5. The vehicle must successfully drive a lap around the track.

Code Pipeline

The PID class is structured in four parts: The object initialization, the error updating, the error calculation, and the hyperparameter tuning. The main view of the PID.cpp is shown in Figure 1. On the init(), the PID parameters are set and the errors are initialized as 0. The variables for the tuning are also initialized. On the UpdateError(), each of the p,i,d errors are updated, whereas on TotalError() the total error is calculated.

As for the hyperparameter tuning, it was used the Gradient Descent algorithm, based on the student antevís’ project¹. In this algorithm, the Root Square Mean (RSM) of the CTE is calculated for a given epoch size and used to change each of the hyperparameters. When the current epoch RSM is smaller than the tolerance, the code stops training.

The PID control algorithm was implemented in the main.cpp as shown in Figure 2.



```
PID.cpp      X  PID.h      X  main.cpp      X
10
11 //Initialize the PID object
12 void PID::Init(double Kp_, double Ki_, double Kd_) {**}
13
39 //Update the errors
40 void PID::UpdateError(double cte) {**}
41
54 //Calculate the steering
55 double PID::TotalError() {**}
56
62 //Functions for PID Hyperparameters adjusting
63 void PID::evaluate() {**}
64
76 void PID::backpropagation() {**}
77
90 void PID::update(double *Kx, double delta_p, double error) {**}
91
96 void PID::UpdateError_epoch(double cte) {**}
97
102 void PID::reset_epoch() {**}
103
108
```

Figure 1: PID.cpp functions

```

main.cpp      X
77
78     double steer_value;
79
80     //Update the p,i,d errors
81     pid.UpdateError(cte);
82
83     //Calculate and adjust the steering value
84     steer_value = pid.TotalError();
85     if (steer_value>1){
86         steer_value = 1;
87     }
88     else if (steer_value<-1){
89         steer_value = -1;
90     }
91
92     //Training
93     std::cout << "Timesteps: " << timesteps << " Epoch size: " << pid.epoch_size << endl;
94     if (need_training) {
95         //std::cout << "True" << endl;
96         if (timesteps % pid.epoch_size == 0){
97             //Evaluate
98             std::cout << "Time for evaluate" << endl;
99             pid.evaluate();
100            std::cout << "Evaluated, must train is: " << pid.need_training << endl;
101            if (pid.need_training && need_training) {
102                std::cout << "Training " << endl;
103                //Pass
104                pid.backpropagation();
105            }
106            pid.reset_epoch();
107        }
108    }
109    timesteps++;
110    ///END OF CODE/////

```

Figure 2: PID implementation on main.cpp

Hyperparameters calibration

The initial values of the PID control's hyperparameters were set manually. Each parameter was set at 1 and were manually changed until the vehicle managed to complete a lap. Using only a P Controller, the vehicle oscillates on the strait part of the lane, and on laps it zigzags until going out of the track. It was observed that a too big P parameter led to strong sharp turns of the vehicle (Figure 3a), so it was set to 0.1. Upon adding the D Controller, the vehicle stopped wobbling on the lane, but at some sharp turns it went over the lane line (Figure 3b). This was solved using an I Controller with a small value of 0.002.

The optimization used epoch of 200 timesteps with a learn rate of 1e-2. The initial and optimized parameters are on Table 1.

Table 1 – PID Parameters at start and after optimization

Parameters	Initial values	After optimization
K_p	0.1	0.100789
K_i	0.002	0.00238509
K_d	1	1.00036



Figure 3: Issues on other controllers. a) P control, oscillating too sharp on the track. b) PD control, going over the lane line.

References

¹Link for antevis project on github: https://github.com/antevis/CarND_T2_P4_PID