PID Controller Project

Jean-François Vanreusel

Overview

This project provides an implementation of a PID Controller. A PID (Proportional – Integral – Differential) controller controls the steering and throttle of a car based on the cross-tracking error. The cross-tracking error (CTE) is the distance between the center of the car and the center of the road.

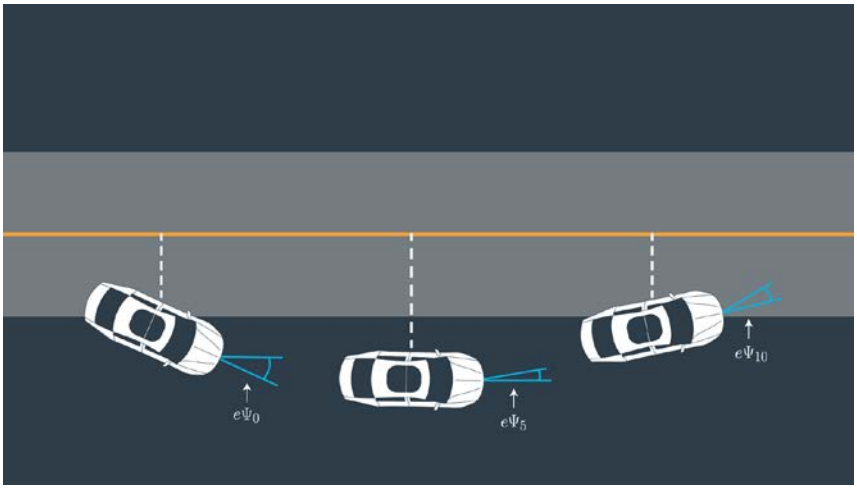
Deliverables

- main.cpp
- pid.cpp & pid.h
- **PID Controller Project.pdf** – You're reading it ☺.

Other project files provided by Udacity were not modified.

PID Controller Algorithm

The PID Controller algorithm generates a response (steering angle and throttle) based on the **Cross-Tracking Error (CTE)**. The CTE is the distance of the car to the center of the road. The white dotted lined on the image below represent the CTE.



Files

Main.cpp mainly focuses on running the simulator and maintaining the car on the road. In main.cpp, we create a PID controller object that responds to the CTEs provided by the simulator. Those responses are used as steering angle and throttle.

The main.cpp file has 2 key settings:

- **#define TWIDDLE FALSE**
  If TWIDDLE is TRUE – we only run the car for 1000 "steps". This allows us to fine tune the hyper-parameters. We compute the average CTE error on those 1000 steps to compare if a run is better than another one.

  By default, we set TWIDDLE to FALSE because we identified parameters that allow us to drive the car around the track.

- **#define RACEDRIVING FALSE**
  When RACEDRIVING is set to TRUE, the car will drive much faster reaching speed close to 80 mph. It is a bit riskier and there is a higher risk that the car crashes. If RACEDRIVING is FALSE (our default), the car moves at much safer speed.

"Twiddle" & Hyper-parameters

To run the PID Control algorithm, we need to provide 3 key parameters: **Kp**, **Ki** and **Kd**.

We identified the following parameters:

- Kp @ 0.0055
- Ki @ 0.0001
- Kd @ 0.2

We used a combination of **manual** and **automated** method to identify good parameters.

We started with Kp at 1 but the car moved with very large sinusoidals. Intuitively, we realized that Kp had to counter-balance large CTEs and picked much smaller values.

Once the car could move a little bit on the track, we fine-tuned Ki and Kd too. The integral error accumulated all the errors observed so far, so we also decided to use a small Ki to counter-balance the integral error (which tends to increase).

We tried to implement the Twiddle algorithm, but discovered that there is some randomness built into the simulator. Even if we were using the same parameters, it generated different errors. So, the Twiddle algorithm could easily take a wrong direction.

So, we basically decided to analyze results for 1000 "steps" and evaluated if the car stayed on the road or not. We started with Kp, then Kd and, finally, Ki.

By playing with the parameters, we also discovered that the car started with large sinusoidal (as the PID controller would give us). By increasing speed, we could reduce this effect a bit – so we used the throttle at 0.6 for the first 100 steps of the car.

We also created a PID controller for the throttle. This is for the RACEDRIVING mode only. The main idea here is to go fast on the straight lines and slow down in the curves.

Results

In safe (non-racing) mode, the car can drive the track using the PID controller. The throttle is 0.3 and the speed is around 30 mph.
In racing mode, the car can also complete a lap on the track but it's a bit riskier. The speed is around 60 mph and reaching even 80 mph on the straight away.