Introduction

The goal for this project is to build a PID controller in C++ for use in Udacity's car simulation. In this project we revisit the lake race track from the Behavioral Cloning Project. The simulator provides the cross track error (CTE) and the velocity (mph) in order to compute the appropriate steering angle.

The project is graded according to these Rubric Points, which have been addressed in my project. A table of contents for the rubric point is below:

Rubric Point Table of Contents

i) Compilation

The code compiles correctly.

Code compiles without errors with cmake and make

ii) Implementation

The PID procedure follows what was taught in the lessons

The PID procedure is built in accordance with what was taught in the lessons. In fact, I have implemented 2 PID controllers. The first for steering angle control (PID) and the second for throttle/speed control (PID_Speed).

The PID implementation is done in PID.cpp. The *PID::UpdateError* method calculates proportional, integral and derivative errors, while the *PID::TotalError* method calculates the total error using the given coefficients.

```
}

void PID::UpdateError(double cte) {
  /**TODO: Update PID errors based on cte***/
  d_error = cte-p_error;
  p_error = cte;
  i_error += cte;
  iter += 1;
}

double PID::TotalError() {
  /** TODO: Calculate and return the total error****/
  return -Kp * p_error - Kd * d_error - Ki * i_error;
}
```

iii) Reflection

Describe the effect each of the P, I, D components had in your implementation.

Proportional Component
This component of the PID controller steers the ego vehicle towards the center of the lane using the provided cross-track error. With just the P component, the ego vehicle will overshoot the central line and drive off the road.

Integral Component
This component eliminates any bias on the controller that could prevent the cross-track error to be completely corrected over time. With just the Integral component, the ego vehicle would just go to go in circles. Luckily, in the Udacity simulator, there is no bias in the controller.

Differential Component
This component mitigates oversteering from the proportional component. The differential component smoothens the steer towards the center line by taking into account the delta in steering angle from the previous timestep. As a result, it will add back an ‘understeer’ force to counteract the proportional component. \

Describe how the final hyperparameters were chosen.

I chose the PID controller parameters manually through trial and error.

First, we check that the car actually drives (i.e., in a straight line) with all parameters as zero. Then, we add the proportional component. Here, the ego vehicle will follow the road but it starts oversteering and quickly drives off the road. Then, we add a differential component to counteract the oversteering. We kept the integral as zero as the simulator had no bias. Any other value would introduce a bias to the vehicle. The final parameters for the PID controller is [P: 1.3, I: 0.0, D: 1.0].

iv) Simulation

The vehicle must successfully drive a lap around the track.

The vehicle drives multiple laps without leaving the track