PID Control

Overview

In this project I implemented a PID controller to control a car in Udacity's simulator. The simulator sends cross-track error, speed and angle to the PID controller using WebSocket and it receives the normalized steering angle and the throttle, in order to drive the car.

Dependencies

The project has the following dependencies (from Udacity's seed project):

- cmake >= 3.5
- make >= 4.1
- gcc/g++ >= 5.4
- Udacity's simulator.

To run the project select the corresponding PID project from the simulator. Build the PID project using cmake (cmake .. and make) and run it ./pid.

The effects of the PID components in my implementation

- The proportional component of the controller tries to steer the car toward the center line (against the cross track error). The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by constant Kp, called the proportional gain constant. If the proportional gain is high, there will be a large change in the output and the system may become unstable. If the gain is small, the control action may be too small when responding to disturbances in the system. The proportional gain contributes to the massiveness of the output change.
  The expression for the proportional gain is given bellow.
  P = Kp * crosstrack_error
- The integral error is the integral of the error over the observation period. In our case the interval between observations is constant. The integral term is proportional to the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have previously been corrected. The accumulated error is then multiplied by the integral gain (Ki) and added to the controller output. The expression for the integral term is:
  I = Ki * integral_error

The integral term accelerates the movement of the process toward the goal point and eliminates the errors caused by a pure proportional controller. Due to the fact that the integral error responds to accumulated errors from the past, it can cause the present value to overshoot.

- The differential term helps counteract the proportional trend to overshoot by smoothing the approach to it. The derivative of the process is calculated by determination the slope of the error over time and multiplying it by the derivative gain Kd.

  D = Kd * derivative_error

The method the parameters were chosen

The parameters were chosen manually. At the beginning some initial values were set. Afterwards based on the result observed in the simulator the parameters were adjusted. I have added extra decimals to somehow increase the precision of the result. First the proportional term was used, the car should follow the road, but it started overshooting and the vehicle moved out of the driving area. The integral part was added but since it sometimes caused the car to move out of the road, I kept a low value for it for this reason. The differential term was then added to overcome the overshooting.

The final values for the parameters are: 0.128, 0.002, 1.623

Although the approach used is not scientific, it proved to work and the results converged to a local minimum.

Simulation

I attached the link of a short video of the vehicle driving in the simulator.