Control and Trajectory Tracking for Autonomous Vehicles

In this project, we build the PID controllers that make it possible for an autonomous vehicle to follow a trajectory built by a path planner. This process involved the creation of:

- **a Generic PID Controller object**
- **a PID Throttle Controller**
- **a PID Steering controller**

Step 1: Implementation of the PID controller object

The generic PID controller object provides us with a blueprint for the implementation of PID controllers. The variables for control and error handling are declared in **pid_controller.h** while the class itself is defined in **pid_controller.cpp**. As expected, the PID controller object doesn't do anything itself. As shown below, launching the Carla car simulator with only the PID controller object implemented only yields a static car that sits there forever in spite of the path being present.

Step 2: PID Controller for the throttle

To create a PID Controller for the throttle, we create an instance of the PID controller object and we initialize it with values suitable for a throttle controller. This is done in **main.cpp**. The throttle controller is responsible for delivering smooth acceleration and matching the velocity required by the path planner. To calculate deviations from the instructions provided by the path planner, we have implemented the **error_throttle** variable. This variable calculates the difference between the actual speed at a given point and the speed expected by the path planner at that particular point. The difference between the 2 is the actual error.

It's important to note that the PID throttle controller has no bearing on steering 😉 In fact, implementing the throttle controller without implementing the steering controller would result in the autonomous vehicle driving in a straight line and it would crash at the first obstacle in its path.

Step 3: PID Controller for steering

Just as we did for the throttle controller, we create an instance of the PID controller object in **main.cpp**. However, this time, it is initialized with values suitable for a steering controller. The PID steering controller is responsible for handling steering inputs matching the expectations of the path planner. As the vehicle travels along the trajectory, we continuously check if the current position and heading of the vehicle match what's expected by the path planner. The variable **error_steer** is used to calculate path deviations.

Similar to the throttle controller, the steering controller is a specialized controller. It doesn't have anything to do with throttle input. It only handles steering.

Results

Following a successful implementation of both controllers, the vehicle is able to autonomously accelerate, decelerate, avoid obstacles, and brake as shown in the screenshots below.

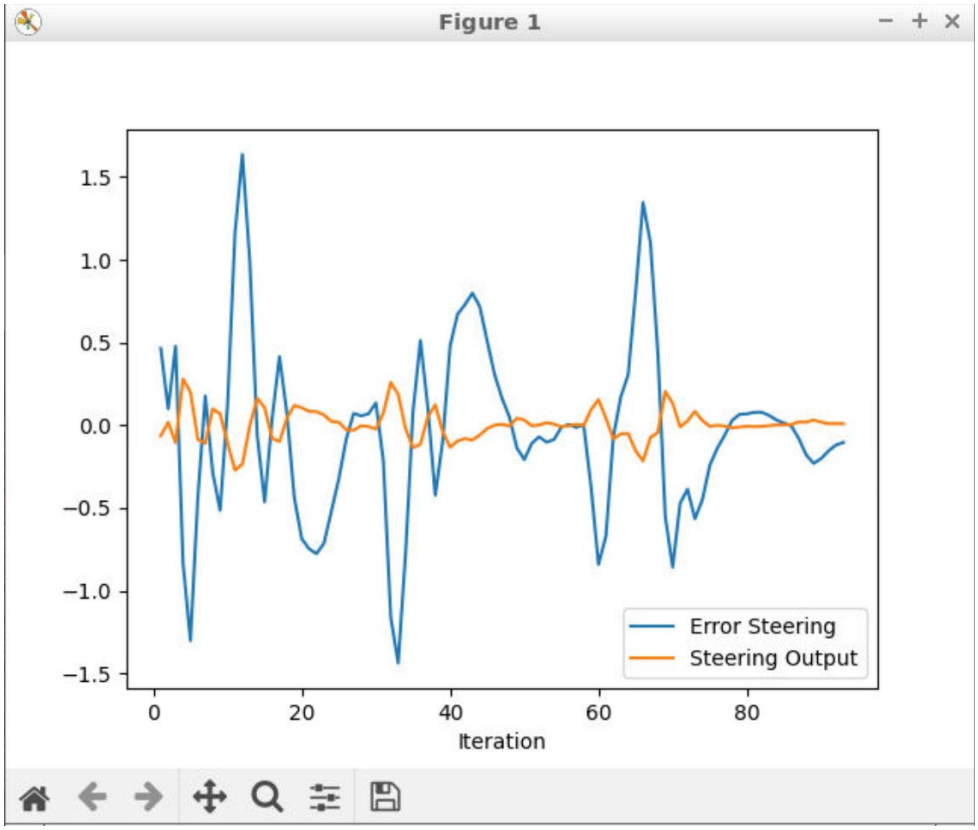
The first picture shows the vehicle at the start of the simulation as it has barely started to move. The speed is at 0.31m/s, the throttle is at 0.62, the brake is at 0.0, and the steering control is at 0.0.

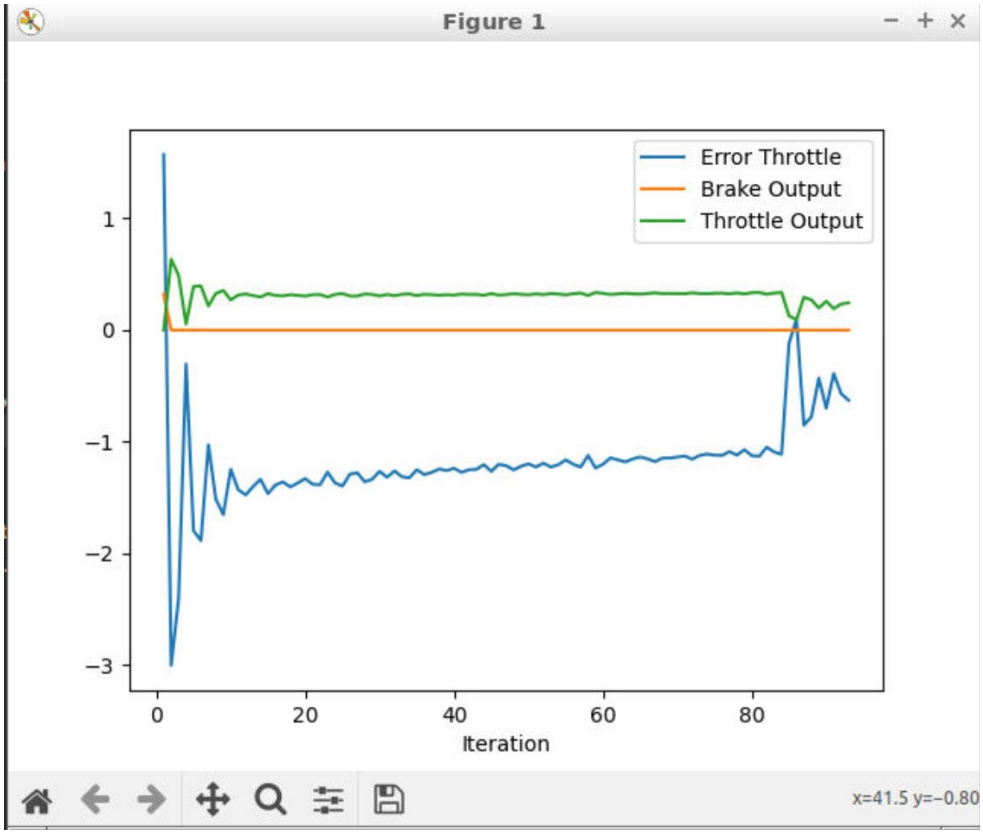
The second picture shows the car moving down the track. It is decelerating as it's approaching its first obstacle. The speed is at 3.99m/s, the throttle is at 0.0, the brake is at 0.31, and the steering control is at -0.07 as it's getting ready for passing maneuvers.

The next picture shows the autonomous vehicle overtaking another vehicle.

The final picture shows the vehicle stopped at the end of the track after completing a successful run with no collision.

Steering and Throttle Plots





The most noticeable thing in those plots is that error graphs are much more amplified than control outputs. This is to be expected since the controller minimizes the errors and the proportionate output is limited to 1.

Another pertinent observation is the fact that errors go opposite to outputs. That's also an expected behavior considering that error correction is implemented in the PDI controller object that both the throttle controller and the steering controller inherit.

What is the effect of the PID according to the plots, and how does each part of the PID affect the control command?

There are three parts to the PID controller. Each part affects the resulting output in a specific way. Let's start with the **Proportional term (kpi variable)**. It's the heavy hitter that limits and tries to correct the current error. However, being too aggressive with the Proportional term will cause it to overshoot.

To help with that overshooting tendency, the **Derivative term (kdi variable)** was introduced. It dampens the rate of error change to stabilize the control signals. Unfortunately, it introduces a residual error over time. Hence the necessity for the Integral term.

The role of the **Integral term (kii variable)** is to correct residual errors over time. If left unchecked, those errors introduce what is known as system bias.

Automatically tuning the PID parameters

The PID parameters can be automatically tuned with Twiddle. The initial deviation is high but, over time, tuning with Twiddle has been shown to give the best results.

Pros and Cons of Model-Free Controllers such as PID

- Pros: Great for quick prototyping, not tied to a specific vehicle, multi-uses
- Cons: Difficult to tune, too few variables, cannot make vehicle type specific (car, truck, watercraft, etc…)

Improving the PID Controller

This controller can be greatly improved with more control over the time interval variable. Waiting an entire second before updating the vehicle position is not a viable solution for an autonomous vehicle in real life. During that time, the vehicle has already traveled several feet and conditions may have changed. Either way, the PID controller is a great start that can be improved upon.