
Simulation time:
Speed: 3.07 m/s
Location: (87.4, 7.8)
Steer: -0.20
Throttle: 0.38
Brake: 0.00
Collision:
Number of vehicles: 4
Behavior: FOLLOW LANE
Prev Junction: -1
Num waypoints: 19
Spirals: 5 / 7

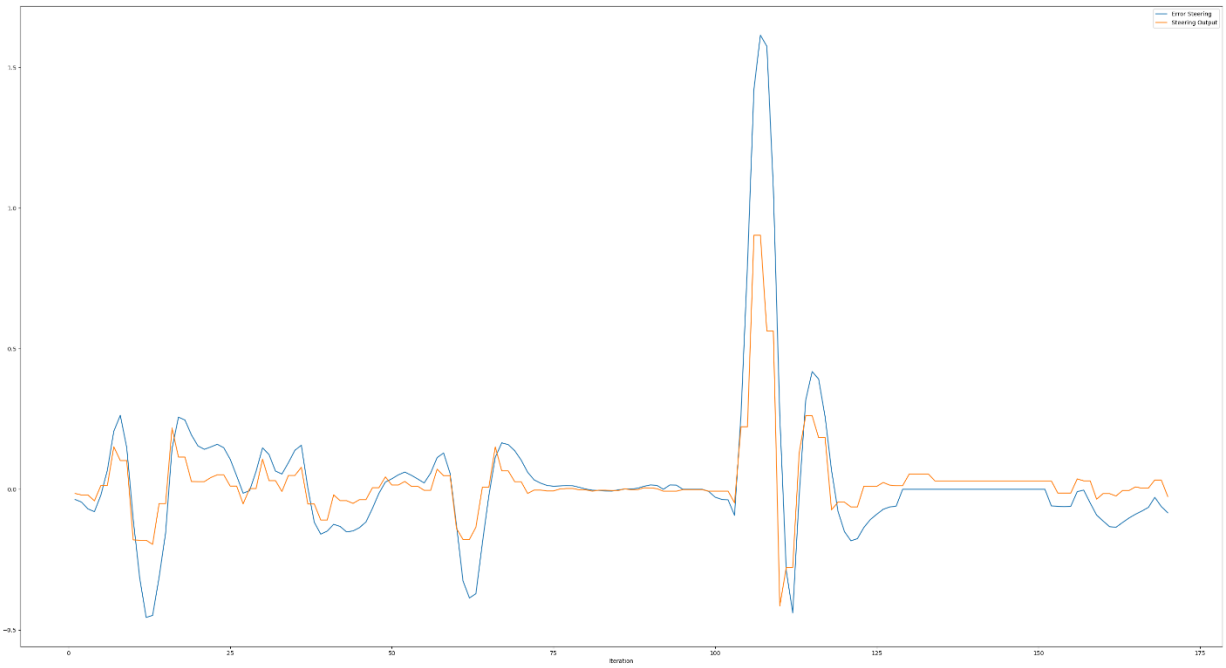

Figure 1 Steer Plot.

The plot signals two lines: the Error Steering in blue and the Steering Output in orange. The Error Steering shows the difference between the current steering and the desired steering direction, which is based on a vector field that I explain below. The vector field suggests that the car should steer in the direction of flow from the average waypoint to the first (or last) waypoint. It also creates a set of reference directions (an orthonormal base) and adjusts the steering if the car's projected position is to the left or right of that base.

The Error Steering stays mostly low, with a few spikes in certain parts of the video: when the waypoints fall behind the car at the beginning, and when the car turns with a slight offset. The Steering Output follows the Error Steering because of the proportional part of the PID controller. However, they aren't the same because the derivative part of the PID controller helps prevent the car from overcompensating for the error. An integral part of the PID controller is small, so we don't see much of its effect in the graph. However, if the car drifts slightly due to damage or bumps, the integral part helps correct it. If there's no problem, the integral error tends to cancel out, and it has little effect.

The large spike around iteration 100 happens when the car turns right and has an offset. The car adjusts for the offset but slightly overcompensates. It keeps adjusting and overcompensating a little until it regains control. Throughout the rest of the journey, there are small bumps when the car makes slight turns. When driving straight, the car fully regains control.

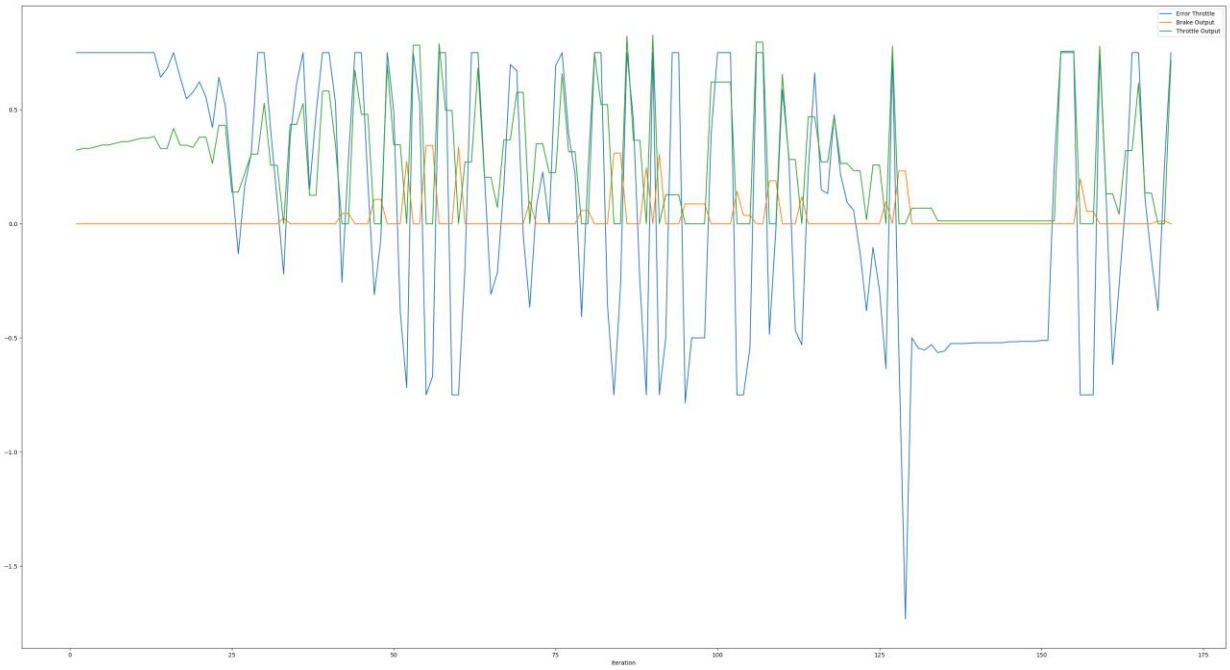

Figure 2 Throttle Plot.

The Throttle Plot shows three lines: the Error Throttle in blue, the Throttle Output in green, and the Brake Output in orange. The Error Throttle shows the difference between the current speed and the desired speed, which is based on a vector field I describe below. Essentially, the vector field suggests that the car should aim for the average speed of the average waypoint. It also creates a reference frame (an orthonormal base) from the flow direction and adjusts the speed if the car's projected position is ahead or behind this reference frame.

The Error Throttle remains low, with small fluctuations (up to 0.5) along the route. I admit that the PID controller for throttle and brake isn't perfectly tuned. This is because the vector field I used is based on averages, which makes it somewhat imprecise. However, overall, the system works well.

The Throttle Output follows the Error Throttle, due to the proportional part of the PID controller. However, they aren’t the same because the derivative part of the PID controller prevents the car from overreacting to the error. An integral part of the PID controller is small, so its effect is not noticeable in the graph. However, if the car drifts slightly due to bumps or damage, the integral part helps correct it.

One important point is that both the Throttle Output and Brake Output are positive. The Error Throttle can be either positive or negative. When the Error Throttle is positive, the Throttle Output increases proportionally and remains positive, while the Brake Output is zero. When the Error Throttle is negative, the Brake Output increases proportionally, though more mildly, and the Throttle Output is zero. The Brake Output is smaller because the brakes are more powerful than the accelerator.

Effect of PID on the Control Command:

The PID controller adjusts the control outputs based on the error (difference between the current and desired state). The **Proportional (P)** term makes the output proportional to the error, the **Derivative (D)** term helps reduce overshooting by reacting to how fast the error changes, and the **Integral (I)** term corrects for small, accumulated errors over time. In the plots, you can see the proportional term directly adjusting the steering or throttle based on the current error, while the derivative smooths out sharp corrections, and the integral only has a noticeable effect in cases of consistent drift.

How to Automatically Tune PID Parameters:

PID parameters can be automatically tuned using methods like **Ziegler-Nichols**, which involves increasing the proportional gain until the system oscillates, or **optimization-based methods** such as gradient descent, which adjusts parameters to minimize errors. These methods aim to find the optimal balance of PID gains for the system, either by trial and error or algorithmic adjustments.

Pros and Cons of PID Controllers (Model-Free Controllers):

The main advantage of a PID controller is its simplicity and ability to work with any system as long as feedback is available. It doesn't require a detailed model of the system, making it adaptable and easy to implement. However, it can struggle with non-linear systems or those with complex dynamics, and tuning the PID parameters can be tedious, often requiring trial and error or advanced tuning methods.

Improving the PID Controller:

To improve the PID controller, you could implement an **adaptive PID** that changes its parameters based on real-time performance or add **feedforward control** to predict necessary adjustments in advance. Additionally, **anti-windup** techniques for the integral term and **hybrid approaches** combining PID with simple models or deep learning could improve stability and performance, especially in complex environments.