ECE 171A Vehicle Steering Project

Kyle Lou

December 10, 2024

1 Introduction

In this project, we designed a simple vehicle steering control system. The input to the system is the lateral acceleration of a reference path, and the system output is the error signal between the vehicle's lateral acceleration and the reference's.

From the simple model, we can extend usages to many real-world applications, with the most obvious being self-driving vehicles. Additionally, this model can be extended to robots to help them move along any path by themselves.

Naturally, the error signal that we strive for is zero. However, a steady state value of zero introduces division by zero in performance analysis with settling time, rise time, percent overshoot, etc. As such, in designing the main system, we first aimed to optimize a preliminary system. This takes the reference acceleration as an input, and the output is the vehicle lateral acceleration. Ideally, the vehicle lateral acceleration should match the input at steady state. If this is satisfied, then the error signal is zero at steady state.

Therefore, we first design the system to match the reference signal. Then, we take the error signal as the output once this has been satisfied.

Our design process is split into three parts. We first analyze the open-loop system. Then, we introduce the PID controller to stabilize the system and bring the steady state error to zero. Lastly, we analyze it with a root locus plot and make minor adjustments with lead-lag compensation.

In the analysis, we made assumptions, such as linearizing the system and assuming constant speed. Additionally, we report the median percent error between input and output signals, as reporting the average will result in it being skewed by the short transient duration, where there is an overshoot and some oscillations. Therefore, the median is the favored measure of central tendency.

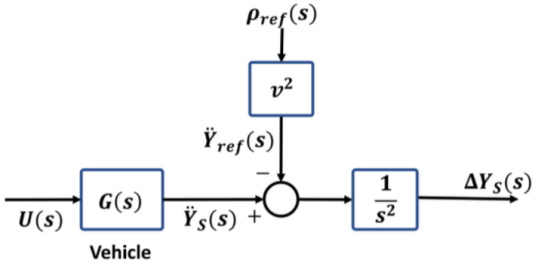

Figure 1: Open Loop System

2 Open Loop System Analysis

2.1 State Space Model

We first derived the state space model and the vehicle transfer function given the following parameters.

Variable	Value	Units	Description
v	25	[m/s]	velocity vector magnitude
M	1573	[kg]	vehicle mass
J	2873	[kg m^2]	vehicle inertia
ℓ_f	1.10	[m]	distance from center of gravity to front-wheel axle
ℓ_r	1.58	[m]	distance from center of gravity to real-wheel axle
c_f	80000	[N/rad]	nominal front-wheel cornering stiffness
c_r	80000	[N/rad]	nominal real-wheel cornering stiffness
μ	0.9		road adhesion as factor of cornering stiffness
d_s	1.96	[m]	distance from center of gravity to sensor location

The vehicle state variables are given below.

Variable	Units	Description
v	[m/s]	velocity vector magnitude
β	[rad]	sideslip angle between vehicle center line and velocity vector
ψ	[rad]	yaw angle measuring vehicle orientation with respect to local coordinate frame
$\dot{\psi}$	[rad/s]	yaw angle rate with respect to local coordinate frame

The following is the linearized state space model for the vehicle.

$$\begin{bmatrix}\dot{\beta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}-\frac{\mu(c_f+c_r)}{Mv} & -1+\frac{\mu(c_r\ell_r-c_f\ell_f}{Mv^2}\\ \frac{\mu(c_r\ell_r-c_f\ell_f}{J} & -\frac{\mu(c_f\ell_f^2+c_r\ell_r^2)}{Jv}\end{bmatrix}\begin{bmatrix}\beta\\ \dot{\psi}\end{bmatrix} + \begin{bmatrix}\frac{\mu c_f}{Mv}\\ \frac{\mu c_f\ell_f}{J}\end{bmatrix} u$$

The vehicle transfer function takes the steering wheel angle as an input, and it outputs a lateral acceleration $\ddot{y}_S$.

$$\ddot{y}_S = \begin{bmatrix}\mu\left(\frac{d_S(c_r\ell_r-c_f\ell_f}{J} - \frac{c_r+c_f}{M}\right) & \frac{u}{v}\left(\frac{c_r\ell_r-c_f\ell_f}{M}\right)\end{bmatrix}\begin{bmatrix}\beta\\ \dot{\psi}\end{bmatrix} + \mu\left(\frac{d_s c_f\ell_f}{J}+\frac{c_f}{M}\right)u$$

2.2 Open Loop Transfer Function

Using MATLAB, we loaded the state space A, B, C, and D matrices and created the transfer function $G(s) = U(s)/\ddot{Y}_S(s)$. The open loop system is shown in Figure 1. Here, we set the input $\ddot{Y}_{ref}(s)$. From the system diagram, we then integrate the output of $G(s)$ two times to get $\Delta Y_S(s)$. Thus, the total open loop transfer function is given $G_{open}(s) = G(s)/s^2$.
Using MATLAB, we computed this to be

$$G_{open}(s) = \frac{99.8s^2 + 636.1s + 3970}{s^4 + 7.377s^3 + 25.21s^2}.$$

2.3 Stability of Open Loop Transfer Function

We then calculated the open loop poles and zeros, finding the poles to be $0, -3.6886 \pm 3.4067j$ and zeros to be $-3.1866 \pm 5.4425j$. As we can see in the pole-zero plot in Figure 2, there is a pole at the origin, so the system is at best marginally stable. We can also see in the unit step response that the

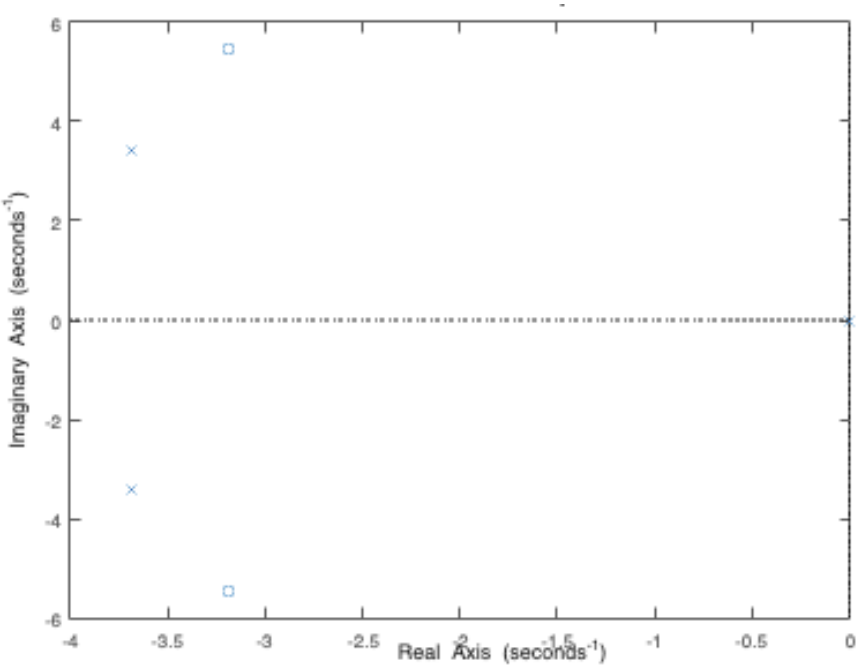

Figure 2: Pole Zero Plot of $G_{open}(s)$

system diverges in Figure 3. Thus, the open-loop transfer function itself is not enough to guarantee the stability of the system, much less than matching the reference path. As such, the task is to design a controller that will stabilize this system. We aim to move the poles into the left-hand plane and set the steady state error to zero.

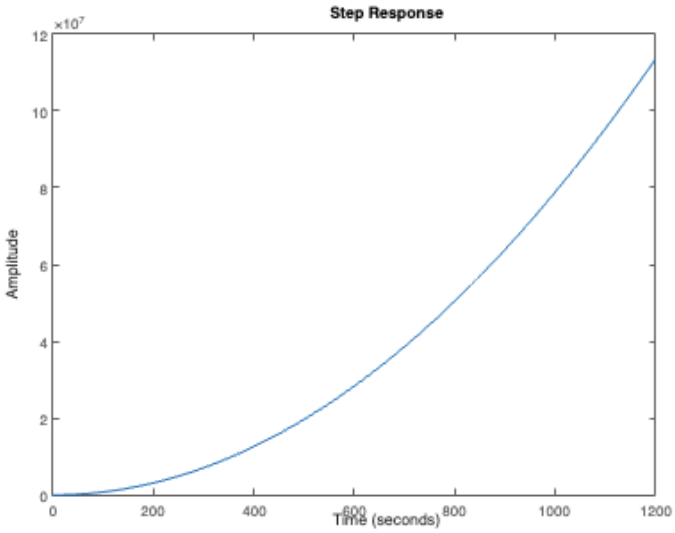

Figure 3: Open Loop Step Response

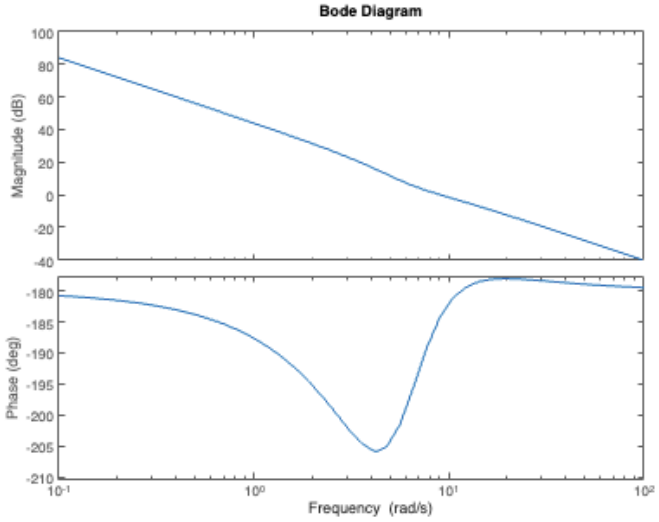

Figure 4: Open Loop Frequency Response

3 Designing PID Control

To incorporate stability and reference tracking, we introduced PID control into the system. Here, we first began with stabilizing the system with a proportional constant. Afterward, with MATLAB's PIDTuner, we began minimizing the settling time, percent overshoot, and rise time.

3.1 Proportional Control

We first began by introducing proportional control to stabilize the system. To start with, we initialized K_P to be very big and K_I and K_D to be very small. We assigned $K_P = 5$ and $K_I = K_D = 0.1$. This allowed us to test the system with mostly proportional control. We then took the transfer function from $\ddot{Y}_{ref}(s)$ to $\ddot{Y}_S(s)$ in the closed loop. As mentioned before, since this is an LTI system, setting the input to be defined as $\ddot{Y}_{ref}(s)$ does not change the analysis, we can scale it by a constant $1/v^2$ to get reference lateral acceleration $\rho_{ref}(s)$. Then, this gives us the closed-loop transfer function

$$H(s) = \frac{\ddot{Y}_{ref}(s)}{\ddot{Y}_s(s)} = \frac{FG}{s^2 + FG}.$$

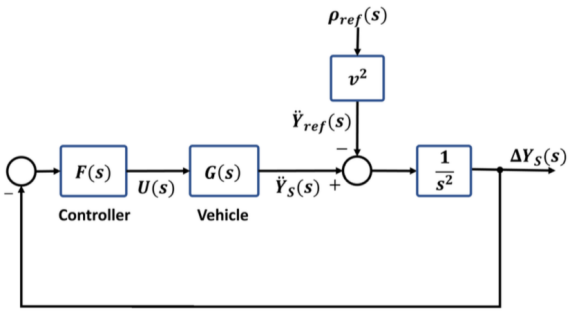

Figure 5: Closed Loop System Diagram

Using MATLAB's **pid** function, we passed in the defined constants to get the controller transfer function $F(s)$, giving us $H(s)$.

$$H(s) = \frac{9.98s^7 + 636.3s^6 + 7989s^5 + 6.056 \cdot 10^4 s^4 + 2.377 \cdot 10^5 s^3 + 5.049 \cdot 10^5 s^2 + 1.001 \cdot 10^4 s}{s^8 + 24.73s^7 + 741.1s^6 + 8361s^5 + 6.12 \cdot 10^4 s^4 + 2.377 \cdot 10^5 s^3 + 5.049 \cdot 10^5 s^2 + 1.001 \cdot 10^4 s}$$

3.2 Proportional Control Stability

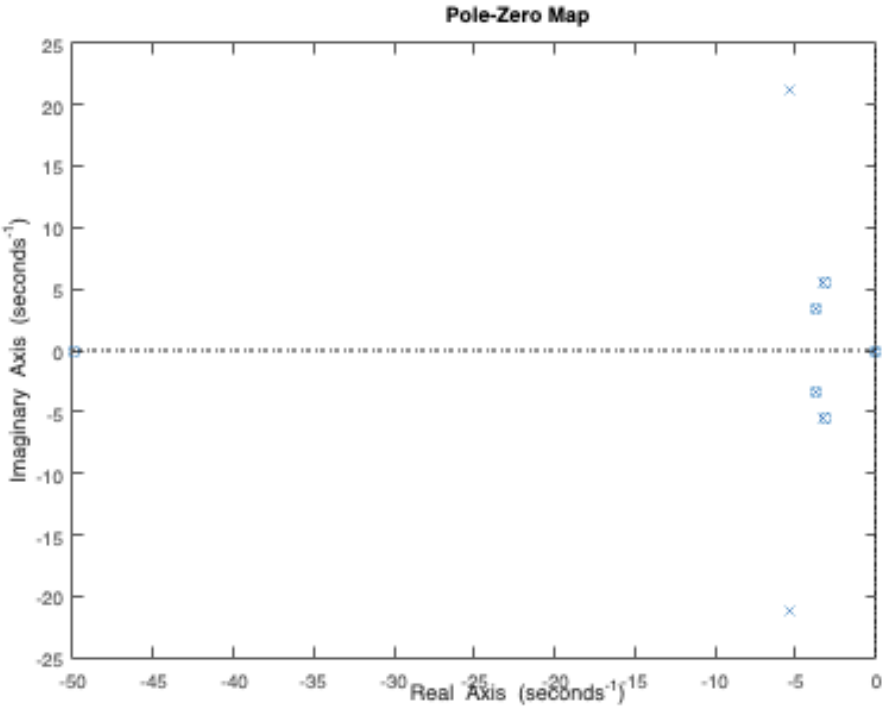

Figure 6: Pole-Zero Plot of P Controller

We then calculated the poles and zeros, finding the poles to be 0, $-5.3775 \pm 21.1187j$, $-3.2913 \pm 5.5628j$, $-3.6886 \pm 3.4067j$, and -0.0200. The zeros are 0, -49.9800, $-3.1866 \pm 5.4425i$, $-3.6886 \pm 3.4067i$, and -0.0200. The pole and zero at the origin cancel, and we are left with poles that only have negative real components. This means that our system is stable.

At the same time, we also plotted the step and impulse responses. We can see that as time goes to infinity, their responses decay to zero, showing the BIBO stability of our system.

While the system is stable, we see in the impulse and step response that there are many oscillations before the system response converges to steady state. Using MATLAB's **stepinfo**, we computed the

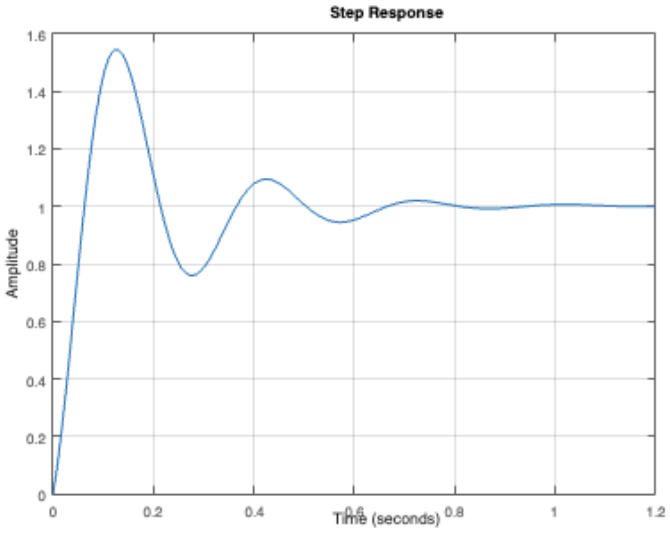

Figure 7: P Controller Step Response

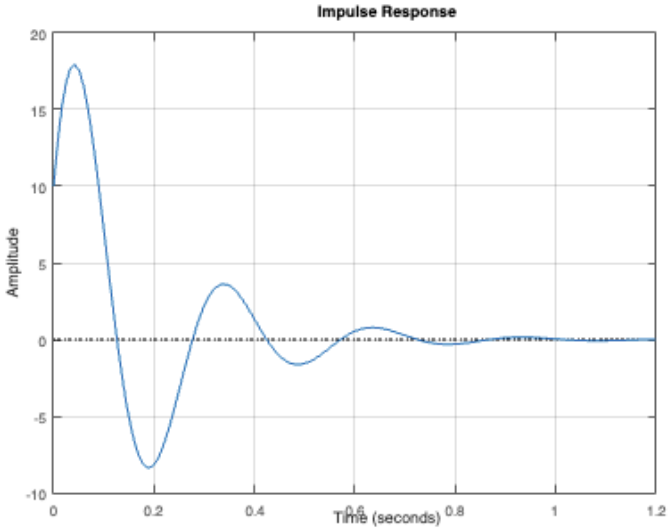

Figure 8: P Controller Impulse Response

rise time of the step response to be $0.0479s$, the settling time to be $0.6389s$, and the percent overshoot to be 54.5164-percent.
While the settling time is somewhat acceptable, the percent overshoot is too high, and we also want to decrease oscillations. As such, now that we have ensured stability, we seek to adjust K_P and K_I such that we can decrease the oscillations and percent overshoot while keeping the settling time short.

3.3 Tuning PID Coefficients

With stability ensured, we seek to tune the coefficients for the controller such that in the transfer function from $\ddot{Y}_{ref}(s)$ to $\ddot{Y}_S(s)$, the settling time is no more than $1s$, and the percent overshoot is no more than 5-percent. We also want to ensure minimal oscillations. As such, we expect to increase K_d to shorten the settling time and minimize oscillations. We also expect increase K_I to further stabilize the system and to reduce the error faster, thus decreasing the settling time. K_P may also be tuned to further increase the system response speed.
When the vehicle turns, if the wheel rotates too much, or if it oscillates, this can cause issues with the vehicle stability. We want the turning to naturally go to the desired angle and not oscillate before it

settles there. The transient response also has to be quick to allow the vehicle to respond to a series of turns.
Using MATLAB's **pidTuner**, we were able to adjust the PID coefficients to satisfy the above requirements. We ended up with $K_P = 14.1065$, $K_I = 26.9496$, and $K_D = 1.6286$. All of the coefficients increased as expected. K_I increased the most. This is due to our requirement to keep the settling time short. K_D also increased, allowing us to anticipate future turns, thus reducing the number of oscillations.
We then defined the net transfer function $H(s)$ using the previous equation, this time using the new PID controller $F(s)$ with these coefficients. To verify stability, we again plotted the poles and zeros. The pole and zero at the origin cancel out again, and there are no poles in the right-hand plane, ensuring that the system remains stable. In the frequency response, we see a gain margin of infinity,

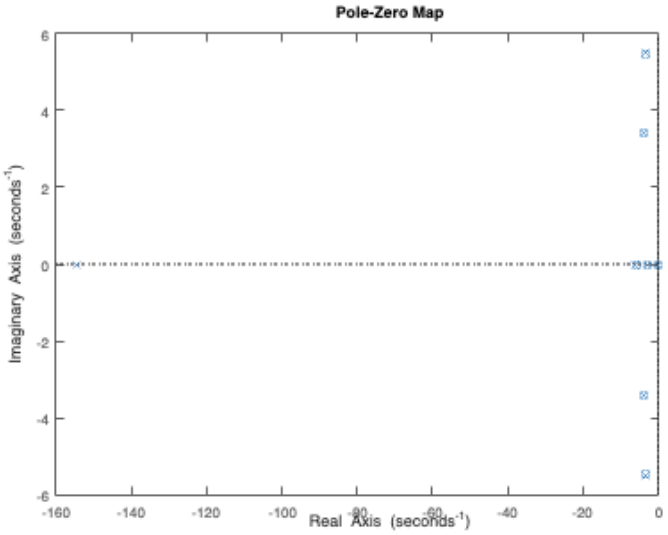

Figure 9: PID Pole Zero Map

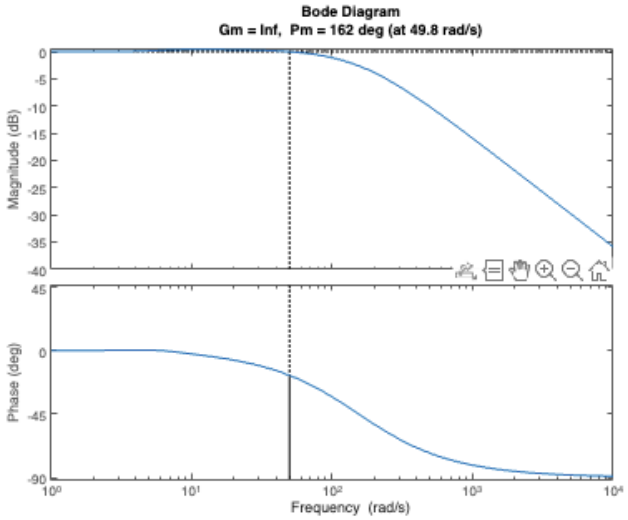

Figure 10: PID Bode Plot

indicating that the system remains stable for all gain values. The stability is also confirmed in the step and impulse responses, with the responses decaying to zero quickly.
MATLAB's **stepinfo** computed that a rise time of $0.012s$, a settling time of $0.1842s$, and a percent overshoot of 4.23-percent. This matches our specifications.

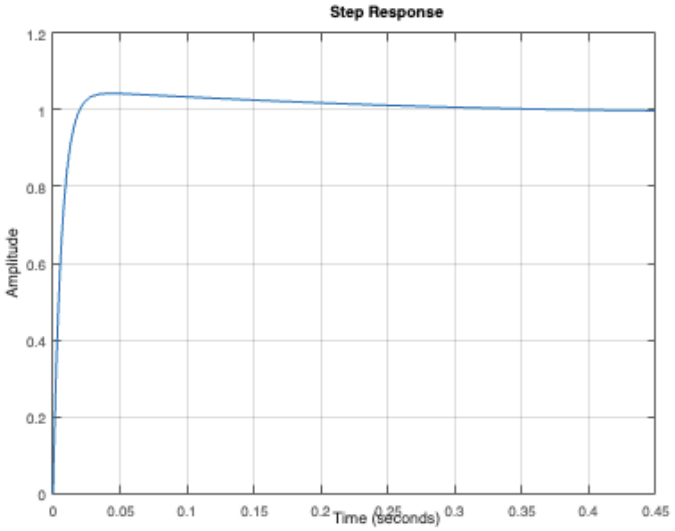

Figure 11: PID Controller Step Response

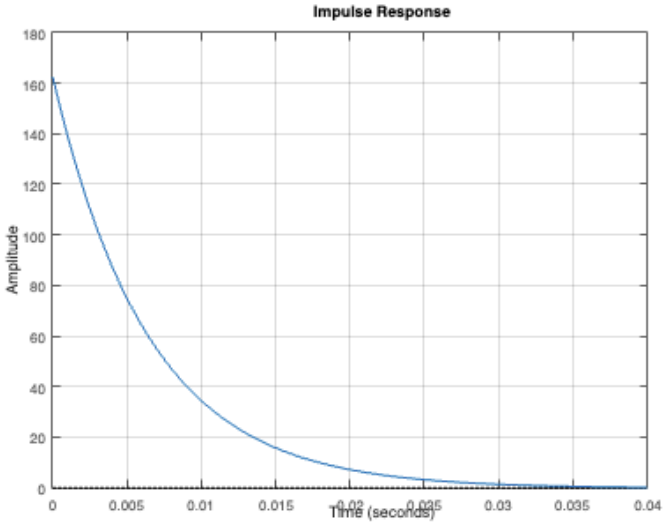

Figure 12: PID Controller Impulse Response

3.4 Error Signal Output

With this, we can now extract the error signal. For an input $\ddot{Y}_{ref}(s)$, the error signal is $\Delta Y_S(s)$. The transfer function can be derived from the system diagram to be

$$L(s) = \frac{\Delta Y_S(s)}{\ddot{Y}_{ref}(s)} = -\frac{1}{s^2 + FG}.$$

Using MATLAB, we can plot the unit step function of this. We first notice that the magnitude is extremely small, and it is displayed on the 10^{-4} scale. This already indicates that our system follows the reference signal extremely closely, and the differences are small. We also note that it goes into the negative direction at first. This is due to the transient response needing time to increase until we match the input. Using **stepinfo**, we computed the transient response time to be $1.6s$, after which it it decays and the error signal decays to zero, implying near-perfect reference tracking.

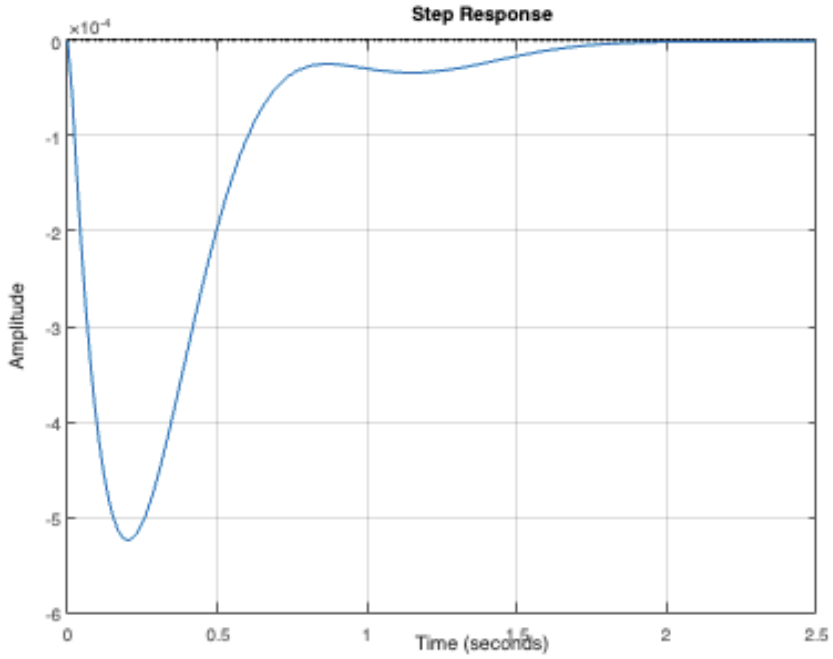

Figure 13: Error Signal Step Response

3.5 Testing With Lissajous Path

After verifying that it tracks the unit step function, which corresponds to a circular path, accurately, we tested the system against a Lissajous Path in the shape of an infinity bow. The parametric equations for such a path are given by $(r_x(t), r_y(t)) = (A\sin(at+\delta), B\sin(bt))$. To achieve the infinity bow shape, we set $A = B = a = 1$, $b = 2$, and $\delta = \pi/2$.

We then obtained the lateral acceleration function. The plot of it is shown in Figure 15.

$$\ddot{y}_{ref}(t) = v^2\rho_{ref}(t) = v^2 \cdot \frac{\dot{r}_x(t)\ddot{r}_y(t) - \dot{r}_y(t)\ddot{r}_x(t)}{(\dot{r}_x^2(t) + \dot{r}_y^2(t))^{3/2}}$$

We then input the $\ddot{y}_{ref(s)}$ into the system and obtained the output $\ddot{y}_S(t)$ with **lsim** over 100 seconds in MATLAB. We overlayed the plots on top of each other, with the latter being red and the former being blue, as shown in Figure 16. The only noticeable mismatch here occurs in the first couple of seconds, where the reference pokes above the system output. This corresponds to the transient response of the signal, as it is still growing to match the reference. Once Steady state is reached though, we can see that there is a near-perfect match between the two. This accuracy can also be seen in the output of the error signal. As we can see in the graph, the error is always bounded by ± 2.5. Considering the large magnitude of $\ddot{y}_{ref}(t)$, this error is extremely small. Considering the large magnitude of the reference signal, this error magnitude is extremely small. Using MATLAB, we computed the median percent error to be 1.32-percent. We can see that the PID controller performs extremely well for this input.

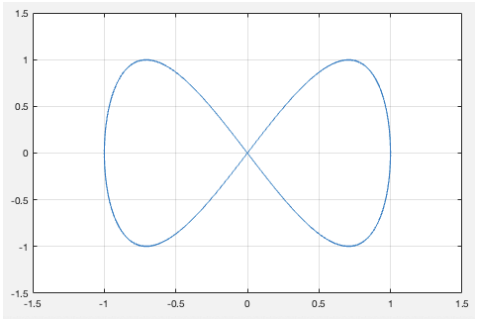

Figure 14: Lissajous Path ($a = 1$, $b = 2$, $A = 1$, $B = 1$)

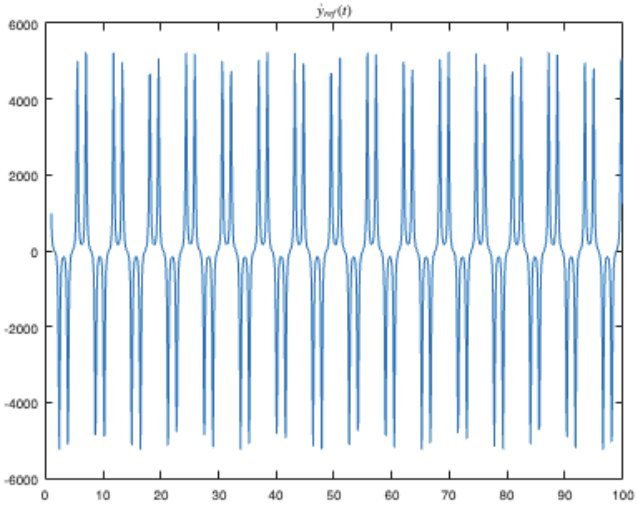

Figure 15: Lateral Acceleration of Test Path

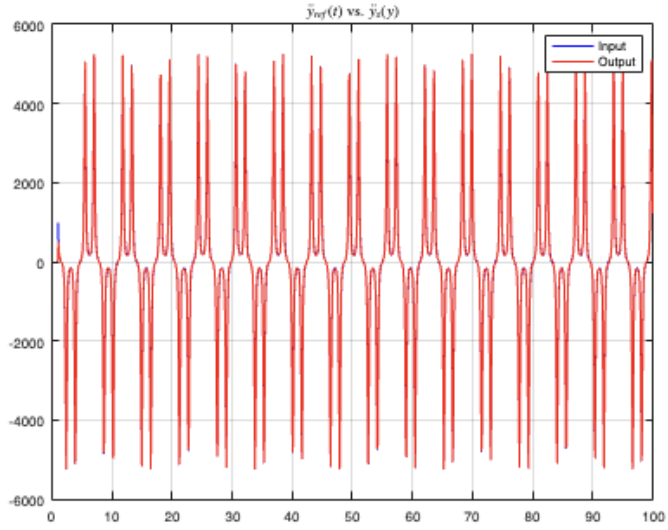

Figure 16: Input (Blue) vs Output (Red)

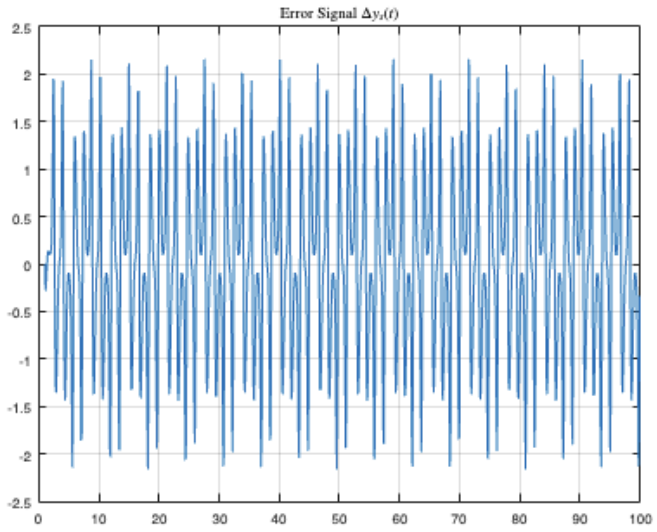

Figure 17: Error Signal

4 Root Locus and Lead-Lag Compensation

We then analyze the root locus of the system and see what improvements can be made with lead-lag compensation. This aims to further improve our system by fine-tuning it.

4.1 Root Locus

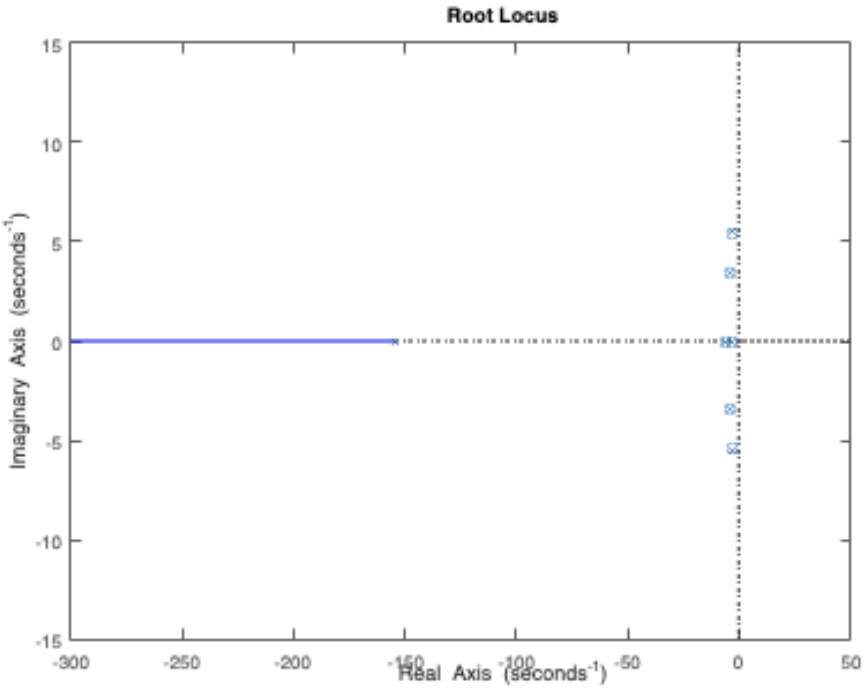

Figure 18: Root Locus of Reference Tracking System

The root locus of the system from $\ddot{y}_{ref}(t)$ to $\ddot{y}_s(t)$ is shown in Figure 18. We can see that the main cluster of poles and zeros is close to the imaginary axis, and there is one pole that rests much further

away from the rest of them. To improve the system even further, we can use lead-lag compensation to add two pair of poles and zeros to stabilize it even further.

4.2 Lead-Lag Compensation

Using lead-lag compensation, we aim to further minimize the steady-state error. With MATLAB's control system designer, we settled on a lead function with a zero at 10 and a pole at 5 and a lag function with a zero at 0.1 and a pole at 0.01. The lead compensator provides a mild phase lead to improve the phase margin without introducing significant high-frequency changes. We selected these values so that they have just enough impact on transient response but not enough to alter the overall system's dynamics.The lag compensator's primary role is to improve steady-state accuracy without altering the system's transient behavior. By placing the zero and pole in this specific range, we ensured that the compensator enhances the system's ability to handle low-frequency signals (i.e., reduce steady-state error) while leaving the transient response mostly unchanged. We then multiplied these with the PID controller to get $H_{LL}(s)$.

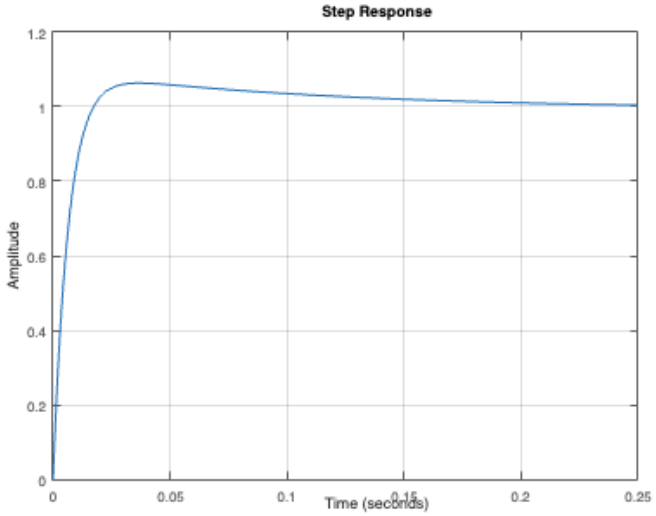

Figure 19: Unit Step with Lead-Lag Compensation

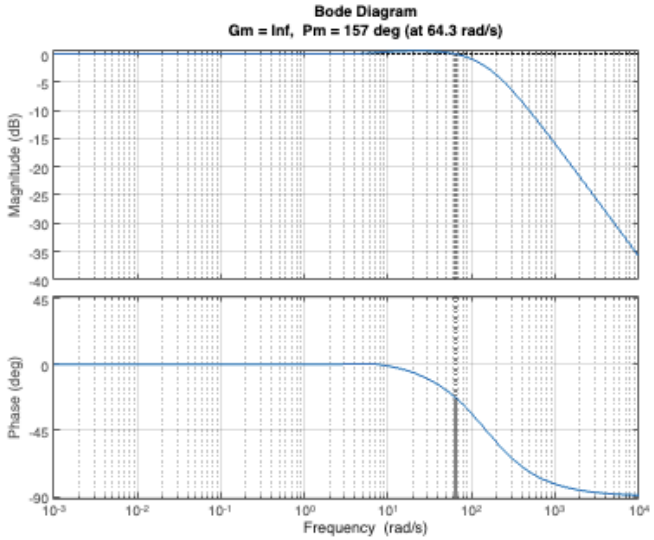

Figure 20: Frequency response with Lead-Lag Compensation

With this new system response, we immediately notice in Figure 19 that the step response decays much more quickly to the steady state value. The rise time is decreased slightly 0.0112 seconds, and the settling time decreased to $0.1479s$. The percent overshoot remained in the same range. The bode plot also indicates we have a much higher phase margin, meaning that there is better stability. At the same time, we still maintain an infinite gain margin, meaning that our system is stable for all gain values.

4.3 Testing With Lissajous Path

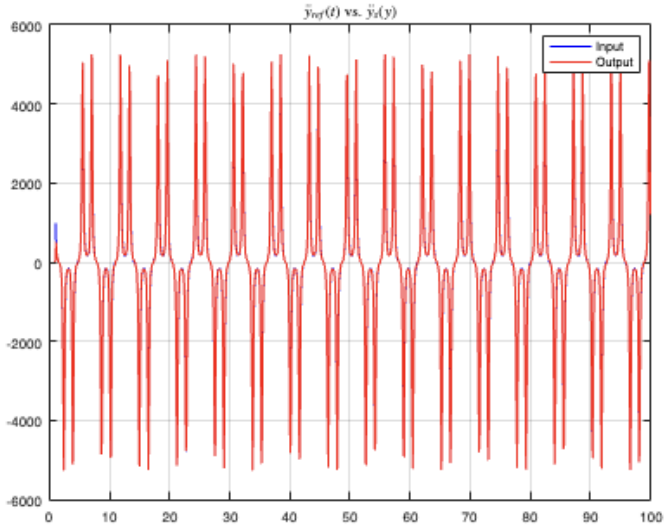

Figure 21: Input (Blue) vs Output (Red)

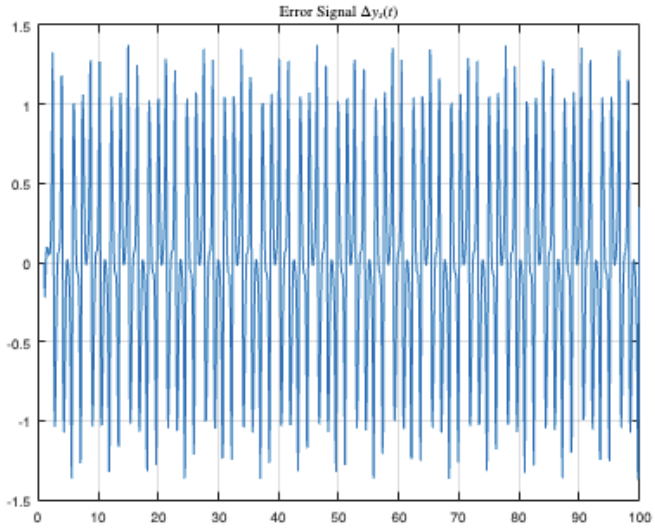

Figure 22: Error Signal

We inputted the same test path from before. Here, we see that the error signal has a lower bound, with the two signals now having a less than 1.5 magnitude difference. We computed a median percent error of 0.77-percent between the reference and output acceleration signals.
We can see that with lead-lag compensation, the steady state error is much smaller, and we have a much lower percent error on average.

5 Testing Against Different Paths

In previous sections, we tested our control systems against a circular path and a Lissajous Curve in the shape of an infinity bow. After verifying the above results, we will test the lead-lag compensated PID control system against more complicated Lissajous curves, recording the error signal and input-vs-output graphs. Each Lissajous curve is defined as $(r_x(t), r_y(t)) = (A\sin(at+\delta), B\sin(bt))$. We will change the coefficients a and $b$to produce increasingly complicated paths while fixing A and B to 1 and $\delta = \pi/2$.

5.1 $a = 3,\ b = 2$

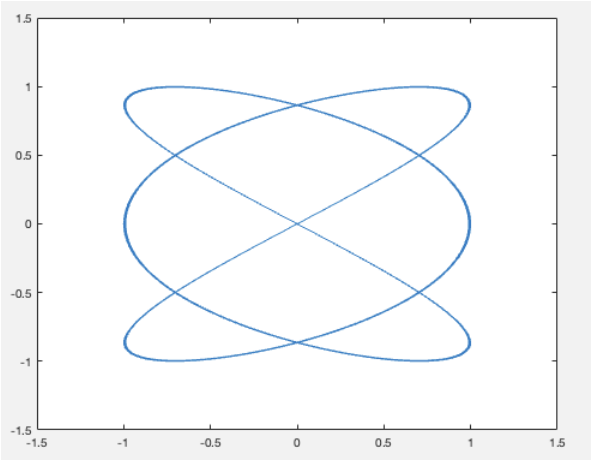

Figure 23: Lissajous Path ($a = 3$, $b = 2$)

The result of this is mostly the same as the infinity curve. The error is still bounded by 1.5, and the output signal tracks the reference signal well. The median percent error between the reference and system output is 0.77 percent.

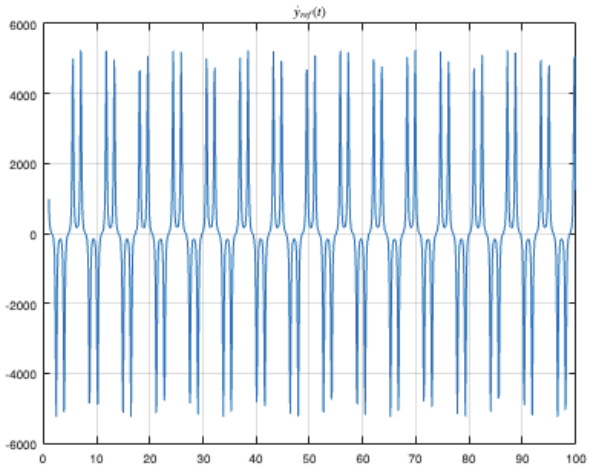

Figure 24: Reference Signal

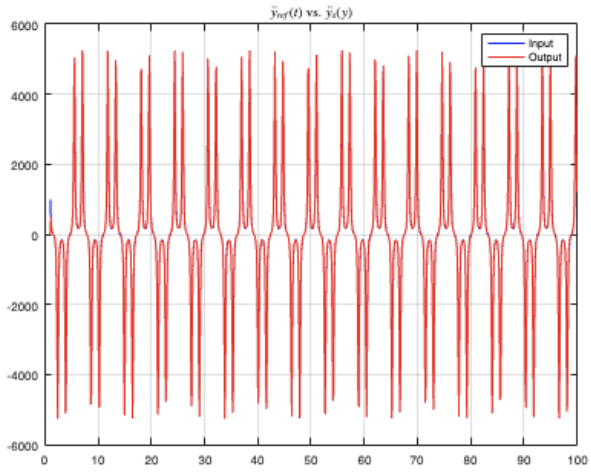

Figure 25: Input (Blue) vs Output (Red)

5.2 $a = 3,\ b = 4$

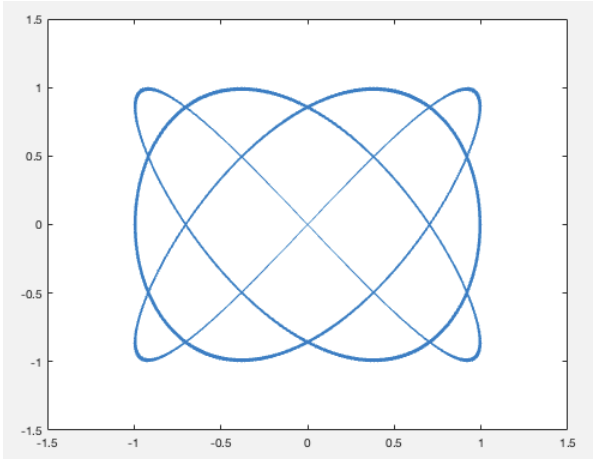

Figure 26: Lissajous Path ($a = 3,\ b = 4$)

This curve is more complicated, resulting in a slightly higher magnitude error signal. The median percent error between the reference and system output is 1.27-percent.

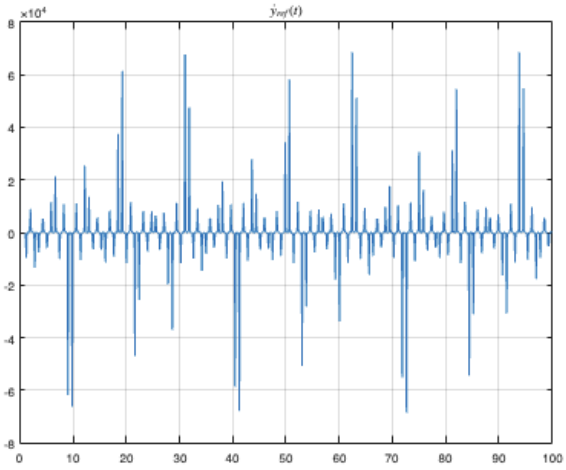

Figure 27: Reference Signal

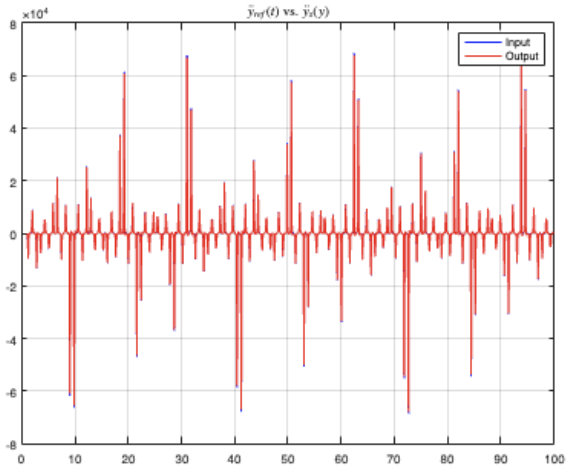

Figure 28: Input (Blue) vs Output (Red)

5.3 $a = 5,\ b = 4$

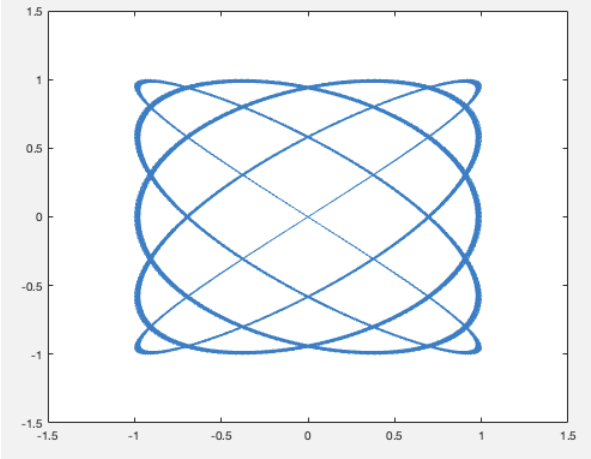

Figure 29: Lissajous Path ($a = 5,\ b = 4$)

The error is slightly higher here, as we can see from the error signal. This is due to the complicated graph having a lot more turns than the previous ones. The median percent error between the reference and system output is 1.27-percent. However, with the large magnitude of the lateral acceleration, the error signal is still relatively small, being bounded by a difference of 15.

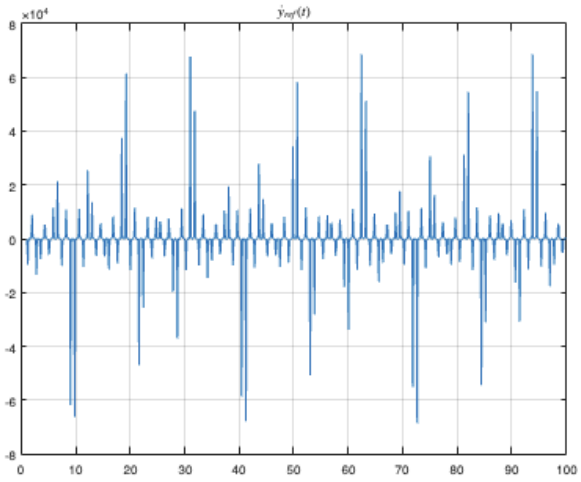

Figure 30: Reference Signal

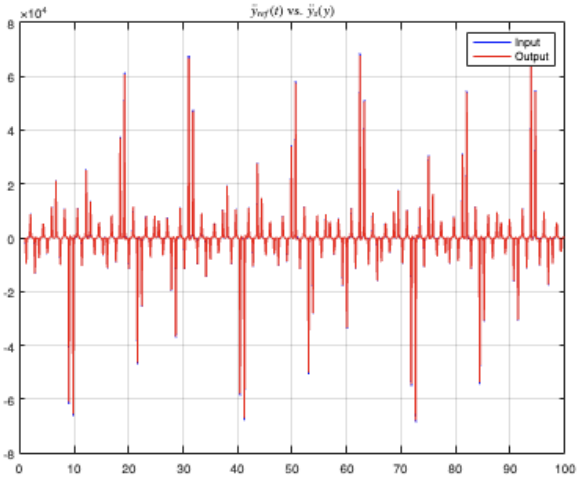

Figure 31: Input (Blue) vs Output (Red)

6 Conclusion

In this project, we built a PID controller from scratch using MATLAB simulations. The goal is to match the vehicle's lateral acceleration to a reference path's lateral acceleration. With PID coefficient tuning and lead-lag compensation, we achieved a system with zero steady-state error for step response and a small error for any arbitrary reference signal.