# PID Controller for Infusion Pump for Anesthesia

Edward Romero, Yousuf Mohideen, Arun Palaniappan

## Control System Design and System Identification

We are interested in developing a PID controller that can provide adequate depth of anesthesia to a patient undergoing surgery based on measured EEG signals of alertness. Developing this controller would eliminate the need for monitoring a patient's alertness during surgery, allowing for automated anesthesia delivery and potentially reducing post-operative complications. The alertness of a patient is measured on the alertness scale (AS). For a normal patient, an AS above 60 indicates the subject may feel the surgical procedure, and an AS of below 40 indicates increased risk to the patient. An AS below 30 indicates severe risk of long-term side effects of anesthesia. The ideal alertness for surgery is an AS of 45.

Figure 1 presents the Simulink model we developed, which inputs a voltage to an infusion pump. This pump releases precise amounts of anesthesia to a virtual patient. The virtual patient's alertness changes in response to the anesthesia, and the virtual patient model feeds this alertness to a controller block. The controller block measures the deviation of the patient's alertness from a target value of 45 on the alertness scale and adjusts the voltage to the infusion pump accordingly, either increasing or decreasing the dosage of anesthesia until the target alertness is achieved and maintained. This model also accounts for patient sensitivity, which can vary due to factors such as age, weight, metabolic rate, and is defaulted to a value of 1.

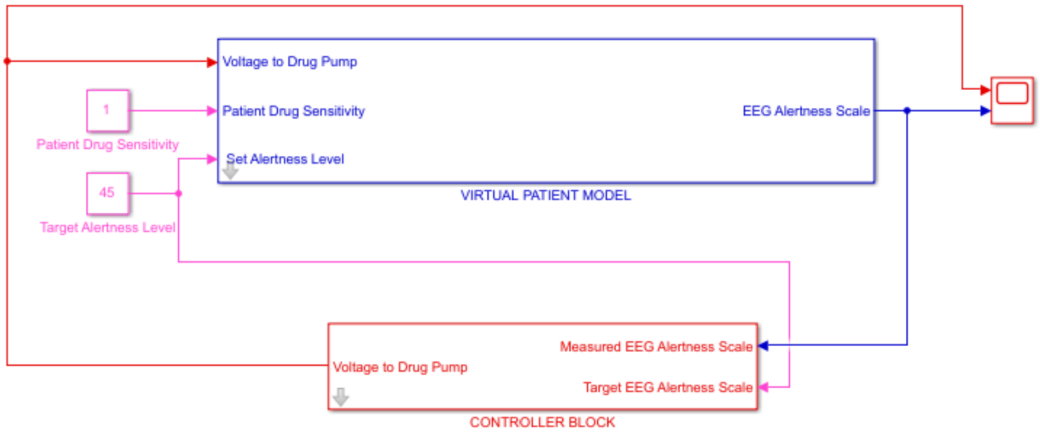


Figure 1: Control System Design

To implement a PID controller into the controller block, we first need to determine the transfer function of the virtual patient model, as it quantifies the dynamics of a patient's response to anesthesia. We started by isolating the virtual patient block by removing the measured EEG alertness connection and the voltage-to-drug pump connection to and from the controller block, as shown in Figure 2, to conduct black box testing of the virtual patient model.

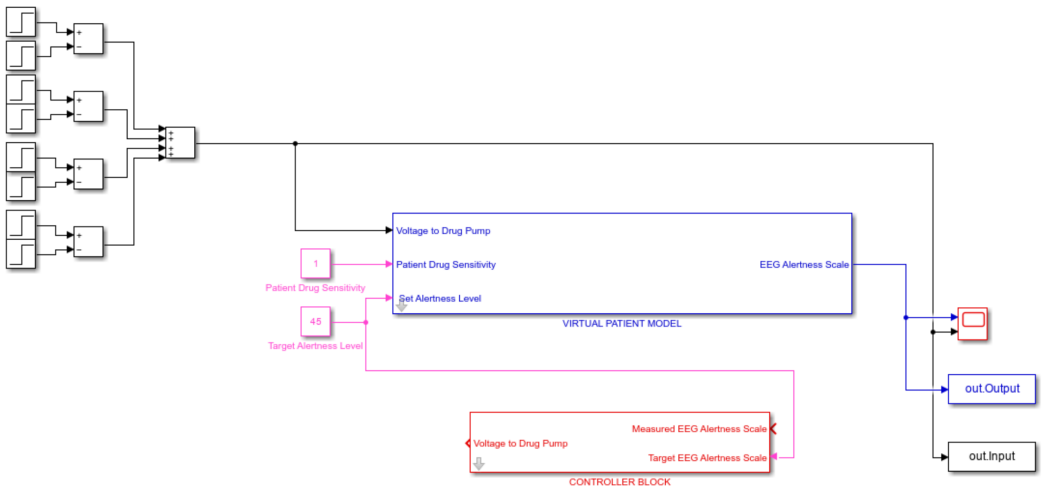


Figure 2: Modified Simulink Model for Parametric System Identification

To achieve a condition number of the observation matrix close to 1, we produced as much variation as possible in the measured output data from the virtual patient block by strategically creating and combining four step inputs for the voltage-to-drug pump. The step inputs used were:

- Step input of 1 from 1200 to 1400 seconds
- Step input of 0.5 from 2000 to 2300 seconds
- Step input of 0.8 from 2500 to 2800 seconds
- Step input of 0.3 from 3300 to 4000 seconds

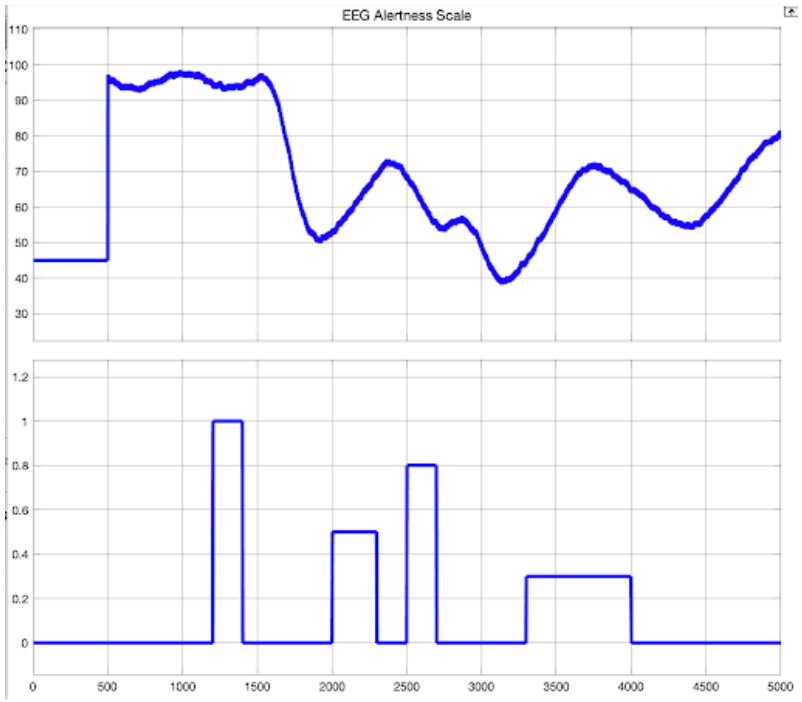


Figure 3: Input and Output Data from Virtual Patient Model

As seen in Figure 3, the patient's alertness fluctuated from values as high as 98 AS to as low as 38 AS over the course of 5000 seconds. We exported the input and output data from the virtual patient model to the workspace. Analyzing this data, we observed that the output data had an offset of 95, corresponding to the initial patient alertness. Additionally, there was a delay of 349 seconds, which is the time taken to observe changes in patient alertness in response to the voltage-to-drug pump input.

Using the Levenberg-Marquardt algorithm, we developed code to estimate the parameters of a transfer function model, and we fit this transfer function to the input and output data. The code we developed can be found in the appendix section of this report. Figure 4 shows the predicted transfer function in red and the measured transfer function in blue. The transfer function obtained from the parametric system identification code fits the measured transfer function perfectly and written below Figure 4.

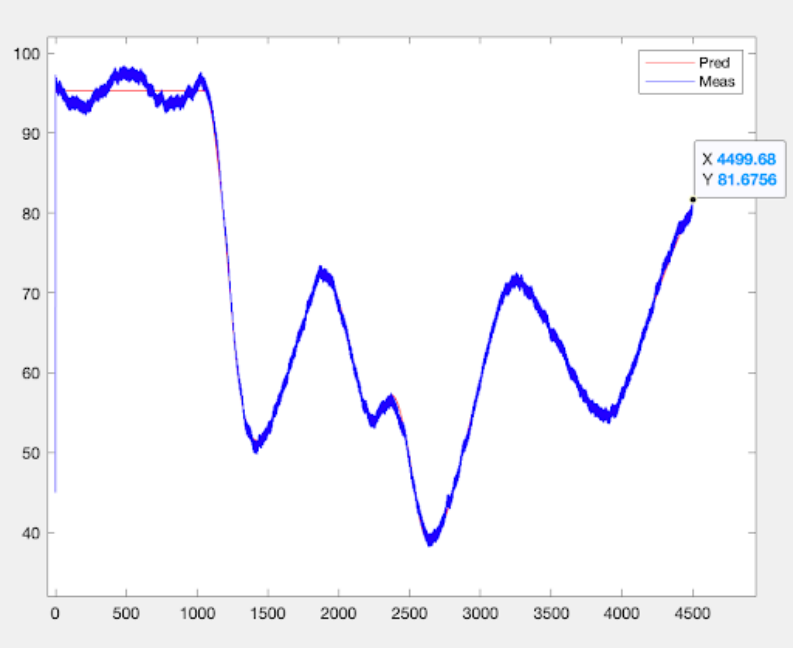


Figure 4: Predicted vs. Measured Transfer Function from Parametric System Identification

$$H(s)_{\text{new}} = e^{-349s}\left(\frac{-0.002357}{s^2 + 0.007896s + 1.39 \times 10^{-5}}\right)$$

## Controller Design to Maintain Alertness at a Specified Level

We modified the controller block in the main model to implement the transfer function within a PID controller block, as illustrated in Figure 5.

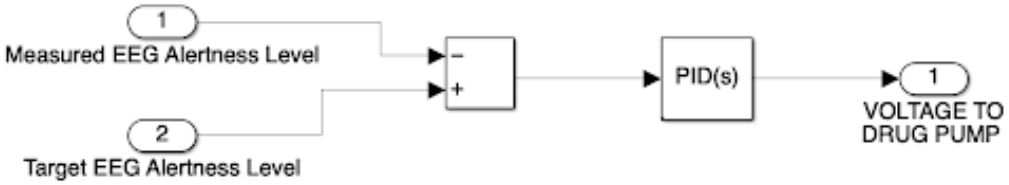


Figure 5: Controller Block

We input the transfer function into MATLAB's pidTuner and adjusted the response time and transient behavior sliders to optimize the system's rise time, settling time, and overshoot. We iteratively fine-tuned these sliders until we found a combination of $K_p$, $K_i$, and $K_d$ gains that brought the patient to the target alertness level quickly, with minimal overshoot. This ensures the patient is no longer alert, allowing doctors to begin the procedure as soon as possible, and helps avoid post-operative complications.

Figures 6 and 7 demonstrate the performance of our controller and the selected PID gains, which excel in achieving a short rise time, quick settling time, and minimal overshoot. This ensures that the controller administers anesthesia at a rate that rapidly brings the patient's alertness to the target of 45 AS and maintains it. The controller overshoots the target alertness by 8%, which remains within the safe range. The minimum alertness score it reaches is 41, and any score below 30 indicates long-term health risks.

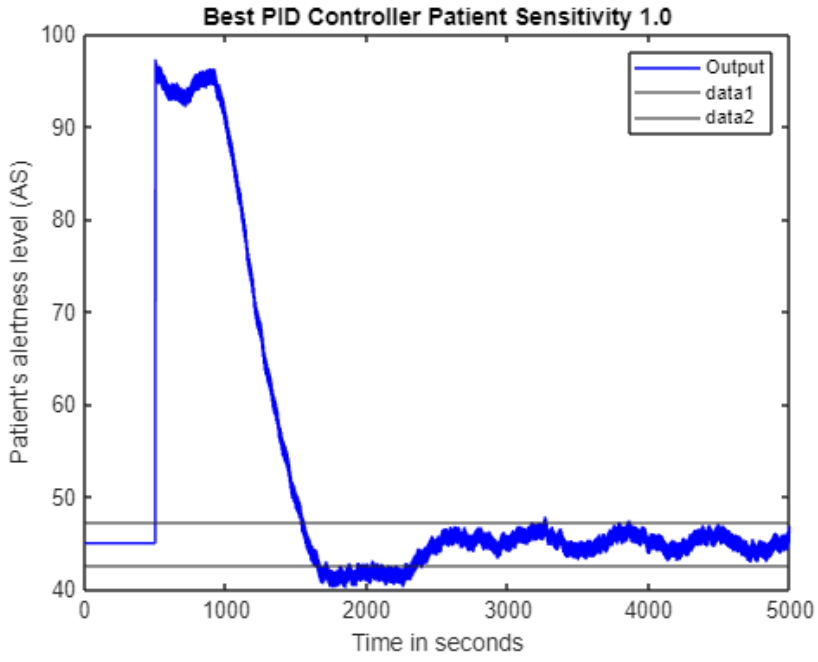


Figure 6: PID Controller Performance for Patient Sensitivty of 1.0

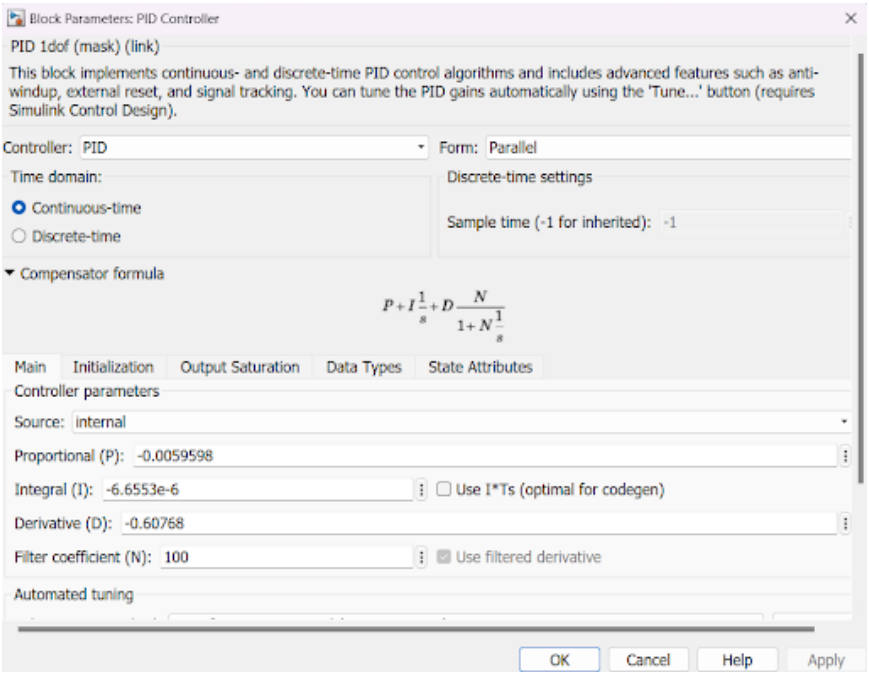


Figure 7: PID Gains

## Controller Performance with a Variety of Patient Sensitivities

To test the robustness of the PID gains selected from Figure 7, we adjusted the patient sensitivity in the Simulink Main Model to +/- 0.5 from the standard sensitivity of 1. We examined patient sensitivities of 1.05 and 0.95 to account for patients who may be hyper-responders to anesthesia or have a tolerance to it.

Figure 8 shows a patient sensitivity of 0.95. Compared to a patient sensitivity of 1, the settling time is slightly longer, but the overshoot is smaller, which is advantageous. This is expected because the patient has a higher tolerance to anesthesia.

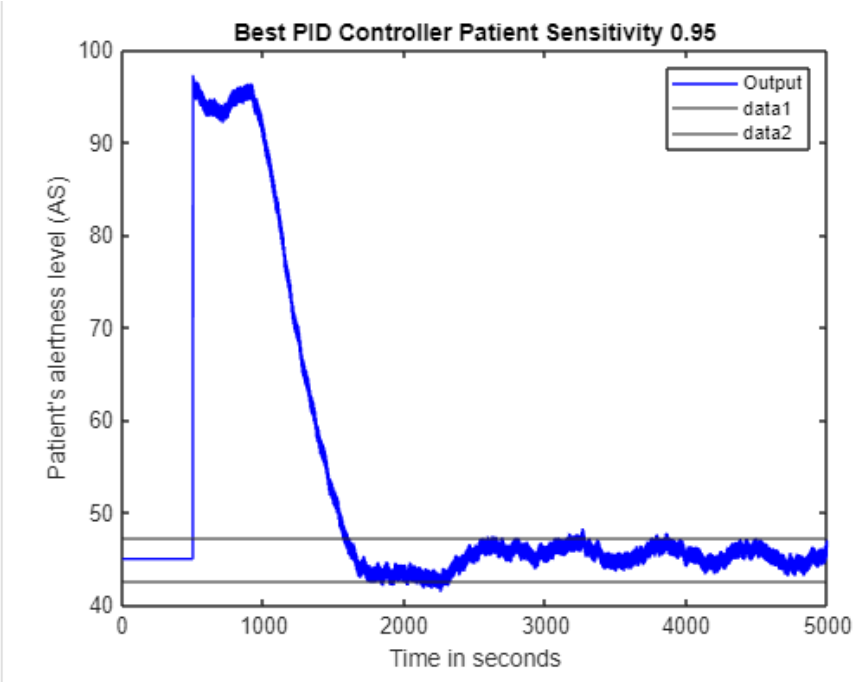


Figure 8: PID Controller Performance for Patient Sensitivty of 0.95

Figure 9 shows a patient sensitivity of 1.05. Compared to a patient sensitivity of 1, the settling time is shorter, but the overshoot is significantly higher. This is expected because the patient hyper-responds to anesthesia. However, this is tolerable because the AS remains around 40 for a short period of time and never dips below 30.

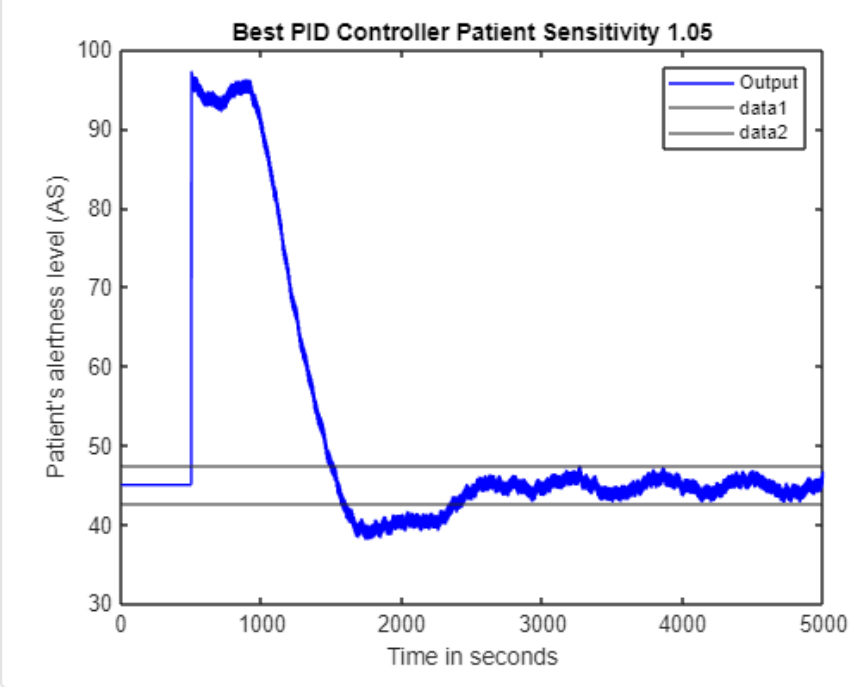


Figure 9: PID Controller Performance for Patient Sensitivty of 1.05

## Controller Stability

Analyzing the controller's stability is crucial because an unstable controller can lead to errors in the anesthesia dosage, which could be fatal for the patient. We used the `rlocus` and `margin` commands in MATLAB to generate root locus and Bode plots for the loop transfer function of our Simulink model.

The loop transfer function of our system was calculated by multiplying the controller block, $C$, with our transfer function block, $P$. Our controller block has negative $k_{PID}$ constants, and our transfer function was also found to be negative. When multiplying $C \cdot P$, we expect the loop transfer function to be positive, since negative $\times$ negative yields a positive result. With this in mind, we removed the negative signs from the controller block equation and transfer function, which yielded the equations $C = k_p + \frac{k_I}{s} + k_D s$ and $P = e^{-349s}\left(\frac{0.002357}{s^2+0.007896s+1.39\times10^{-5}}\right)$. Multiplying $C$ and $P$ with our PID gains, we found the loop transfer function below.

$$\text{Loop Transfer Function} = e^{-349s}\left(\frac{0.001432s^2 + 1.405 \times 10^{-5}s + 1.569 \times 10^{-8}}{s^3 + 0.007896s^2 + 1.39 \times 10^{-5}s}\right) \tag{1}$$

We generated the Bode plot for the loop transfer function and found the gain margin to be 8.16 dB and the phase margin to be 74.7003°, as seen in Figure 10. The Bode plot showed that our system is stable for our PID gains because both the gain margin and phase margin are positive. Thus, we do not have to worry about dosage errors, since the controller is always stable.

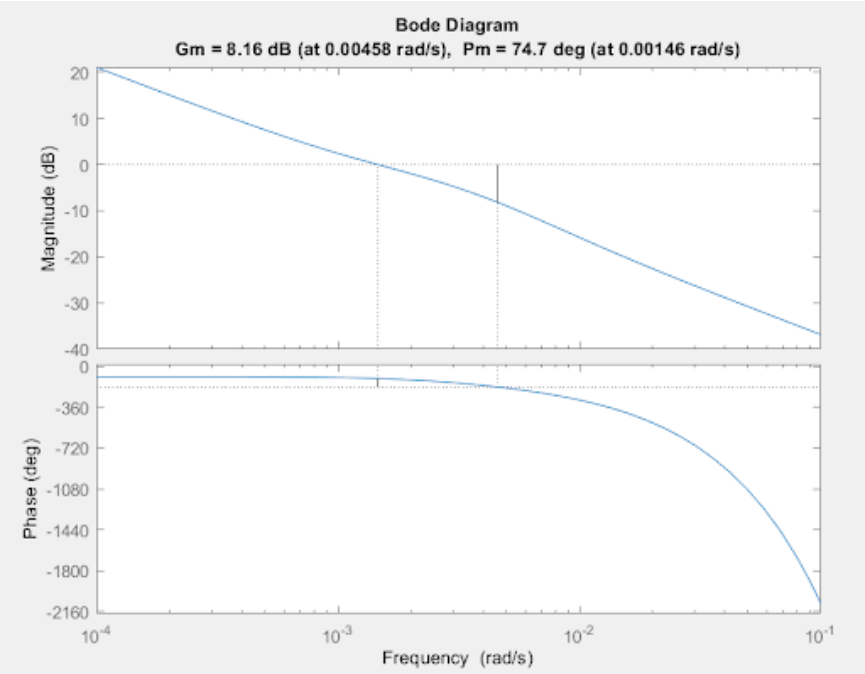


Figure 10: Bode Plot for Loop Transfer Function

Likewise, using `rlocus` on the transfer function below showed the system is stable for our PID gains because the roots of the denominator all lie to the left of the real axis, as seen in Figure 11.

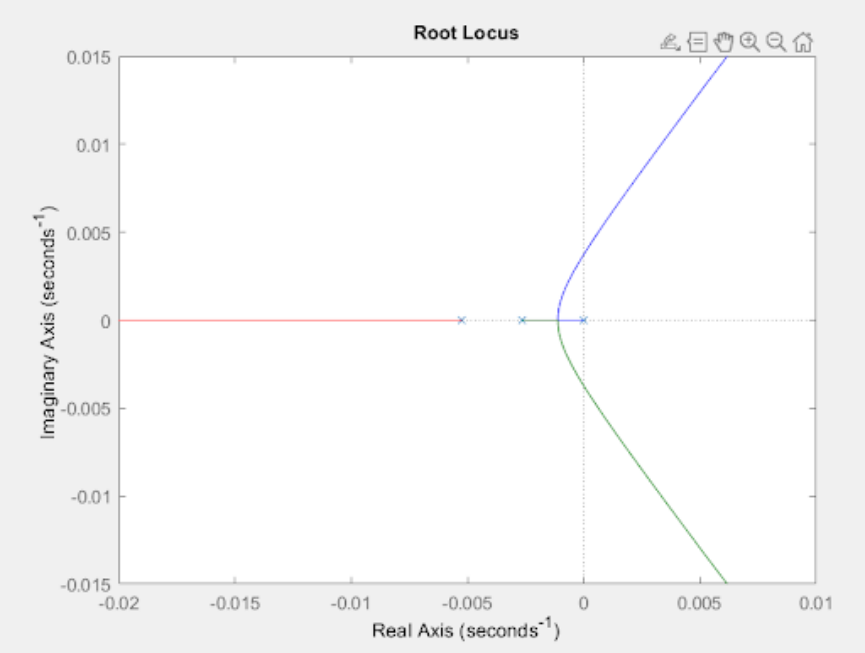


Figure 11: Root Locus Plot for Denominator of Loop Transfer Function

$$\text{Denominator of Loop TF for } \texttt{rlocus} = \frac{1}{s^3 + 0.007896s^2 + 1.39 \times 10^{-5}s} \tag{2}$$

## Cloud Computing Platform Assessment

We are interested in determining if control of this device can be accomplished through a cloud computing platform. Therefore, we analyzed the maximum communication lag that is tolerable with the controller. The phase margin of our loop transfer function is $74.7°$ at $\omega_{gc} = 0.00146$ rad/sec, as shown in the Bode plot in Figure 9.

Time delay $T_D$ is given by:

$$T_D = \frac{(\text{Phase Margin}^\circ) \cdot (2\pi/360^\circ)}{\omega_{gc}}$$

Substituting the values:

$$T_D = \frac{(74.7^\circ) \cdot (2\pi/360^\circ)}{0.00146 \text{ rad/sec}} = 892.99 \text{ seconds}$$

We find that the maximum additional delay the system can tolerate before becoming unstable is 892.99 seconds. Given that typical communication delays in cloud computing can vary widely but often range from milliseconds to several seconds depending on network conditions, it is critical to ensure that the total delay, including both control loop processing time and network latency, remains well below 892.99 seconds.

If the simulation indicates that delays approach or exceed this threshold, as suggested by the simulation running for over 5000 seconds in total, it may be impractical to use a cloud-based controller due to the risk of instability.

## Appendix

**Code for Parametric System Identification and Plotting:**

```
T = 0.001;
u = input.Data;u = u(500000:end);
t = [0:length(u)-1]'*T;
p = 500/T;
y = output.Data;y = y(500000:end);
plot(t,u,'k',t,y,'r')  % Plotting input, u, and output, y, against t.
legend('Input','Output')
N = min(length(y),length(u))
%% Prep data for estimation
global uu
```

```
global data
global tt
uu = u(:);
tt = t(:);
data = y(:);
%Parametric Fit
theta_init(1) = 1;
theta_init(2) = 1;
theta_init(3) = 1;
theta_init(4) = 95;   % for modeling signal offset
theta_init(5) = 500;
theta = zeros(size(theta_init));
%OPTIONS = optimoptions('lsqnonlin','Algorithm','Levenberg-Marquardt','MaxFunEvals',1200);
OPTIONS = optimoptions('lsqnonlin','Algorithm','Levenberg-Marquardt','MaxFunEvals',1200,'FunctionTolerance',1e
    -10);
theta = lsqnonlin(@(theta) fn_linmodelq2(theta),theta_init,[],[],OPTIONS);
%% MODEL HERE HAS TO MATCH MODEL IN lsqnonlin line above (fn_linmodel)
num = [theta(1)];
den = [1 theta(2) theta(3)];
Hs = tf(num,den);
s = tf('s');
%ypred = lsim(Hsnew,u,t);
% DELAY
if (theta(5)<0) % NOTE: CANNOT HAVE NEGATIVE DELAY
theta(5)=0;
end
Hsnew = Hs*exp(-s*theta(5));
ypred = lsim(Hsnew,u,t) + theta(4);   % signal model with offset
plot(t,ypred,'r'); hold on;
plot(t,y,'b'); hold off;
legend('Pred','Meas')
```

**Code for LinModelq2:**

```
function error = fn_linmodel(theta)
global data
global uu
global tt
num = [theta(1)];
den = [1 theta(2) theta(3)];
Hs = tf(num,den);
%ypred = lsim(Hs,uu,tt);
%% USE WHATEVER MODEL MATCHES YOUR NEEDS - here is one with offset (theta4) and delay theta(5)
%
s = tf('s');
% num = [theta(1)];
% den = [1 theta(2) theta(3)];
% Hs = tf(num,den)
% DELAY
if (theta(5)<0) % NOTE: CANNOT HAVE NEGATIVE DELAY
theta(5)=0;
end
Hsnew = Hs*exp(-s*theta(5));
ypred = lsim(Hsnew,uu,tt)+theta(4);  % Model with signal offset.
error = data - ypred;
```