PIDDY

A Novel Controller Design for Anaesthetic Infusion Pumps

Abraham Alkhatib

Haozhen Dong

Undergraduate Interventional Control Units, Inc.

Introduction. Maintaining an adequate depth of anesthesia is always an important and challenging aspect of patient care during surgery and other critical operations. Inadequate anesthesia can lead to patients experiencing pain and distress during the procedure, while excessively deep anesthesia can result in postoperative complications, long-term side effects,and even death **[1]**. To better understand and analyze the mechanism of anesthesia's drug control, our final project for BIOE 420 will focus on designing a PID controller to provide the target level of anesthesia to a virtual patient model.

The goal of the controller is to monitor the patient's alertness level on the EEG alertness scale and maintain it within the ideal range of 45, which represents the optimal depth of anesthesia. This is a delicate balance, as an alertness score above 60 indicates the patient may still be able to feel the surgical procedure, while a score below 40 increases the risk of adverse effects, and a score below 30 carries a severe risk of long-term complications.

Moreover, patients can also exhibit varying sensitivity to anesthetic drugs, which must be taken into account by the controller to ensure the appropriate depth of anesthesia is maintained throughout the procedure **[2]**. To further determine the quality of our PID controller, we'll also test our project with virtual patients with varying sensitivity to anesthetic drugs (0.9-1.1) and our own inputs.

The Patient-Pump System. To design a successful controller, we must first thoroughly understand the system to be controlled. We've identified the plant function as the patient-pump system. To isolate it and perform system identification, we opened the loop, not allowing any feedback from the plant. We then performed a series of tests on the system to get an estimate of the delay and offset of the system, which included impulse response and step response. **Figure 1** shows the system response to these input signals. Note that the first 500 seconds are for initialization and should be discarded. From the figure, we can see that the system has an offset of 90, and a delay of roughly 400 seconds. We also noticed that the system settled into steady state after roughly 2700 seconds, discarding the first 500 seconds for initialization, we calculated `p = (2700-500) = 2200` seconds. Since a good input that characterizes all of the system's response should have a length of at least `3*p`, we used a total input length of `L = (3*2200) + 500 = 7100` seconds.

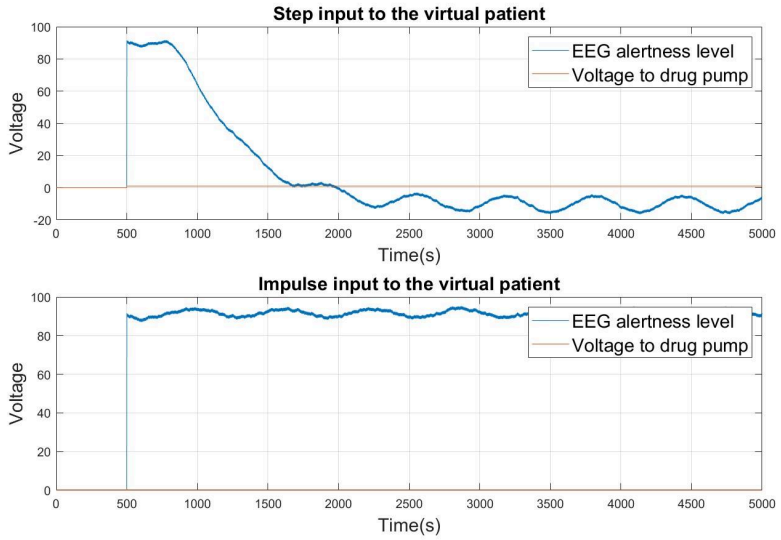

Figure 1: The Patient-Pump System's Response to a Unit Step Input

After probing the system with impulse and step signals, we decided to use parametric system identification via `lsqnonlin()` in MATLAB to estimate the plant transfer function. We chose parametric techniques as it can predict system behavior under different inputs, which is particularly useful as we tested a variety of inputs. Input design is essential to parametric system-identification, a good input should maximize the information output of the system, making the output highly variable and as random as possible, and have a length of at least `3 * p` to capture the total response duration. From step response, we know the system has p = 2200, thus we used a total input length of `L = (3*2200) + 500 = 7100` seconds. Our custom input used a summation of two square waves as the base signal, and included pulses and zeros of varying duration to excite the system as randomly as possible while keeping the response within a reasonable DA range of 20 to 70. This allows us to capture the system dynamics in its operating range of around 45. **Figure 2** shows the custom input signal, along with the system response.

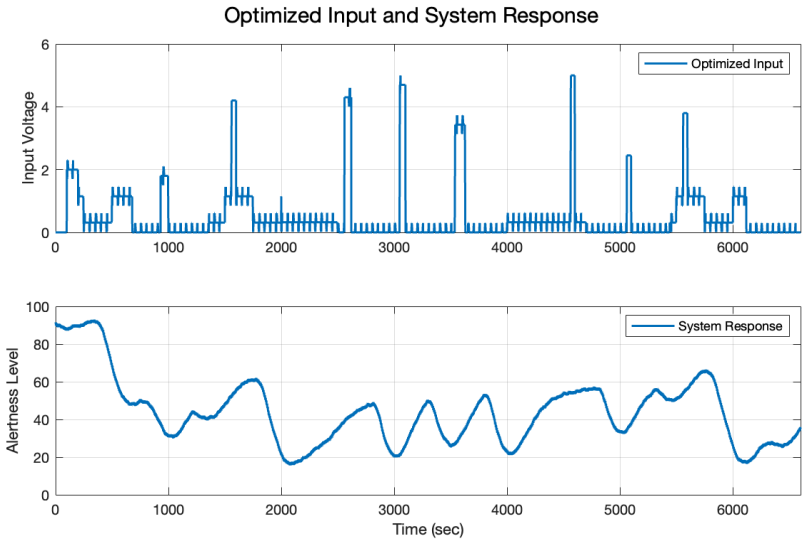

Figure 2: Optimized Input and System Response

** The first 500 seconds of input and response was chopped off for visual purposes.*

*** The code for optimized input is shown in appendix as Supplemental Figure 2.1*

After optimizing the input for an output with maximal information, we move on to perform parametric system identification to find the loop transfer function using `lsqnonlin()` in MATLAB. `lsqnonlin()` also calls a helper function called `fn_linmodel()` that generates entries to be optimized for its cost function. It also requires a pre-initialized input array theta, representing coefficient estimates for the transfer function. We chose a second order transfer function to model the system for two reasons: First, the slope of the step response at t=0 = 0. Second, there were oscillations in step response. We also initialized a delay of 400 seconds and an offset of 90, as observed in the system's step response.The final estimated loop transfer function is shown below in ***Figure 3***, while ***Figure 4*** shows its estimated output of the plant system using *lsim()*.

$$H(s) = \frac{-0.002262}{s^2 + 0.01201s + 2.262\times10^{-5}} e^{-266.5788s} + 90.592$$

Figure 3: Estimated Loop Transfer Function of the Plant System

** The exact code for lsqnonlin() is shown in appendix as Supplemental Figure 3.1.*

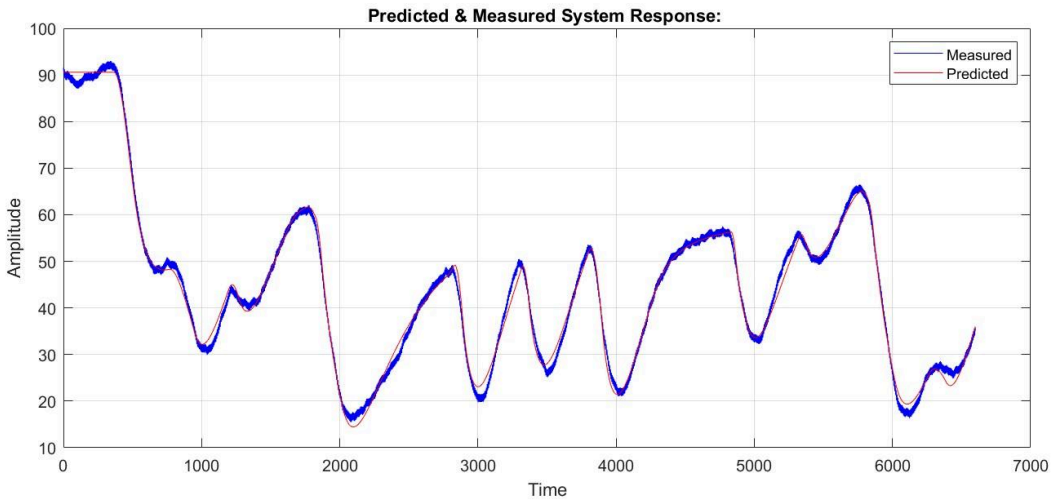

Figure 4: Predicted vs Actual System Response

** The code for lsim() and the error function is shown in appendix as Supplemental Figure 4.1.*

Controller Design. To control the system, we decided to utilize a Proportional-Intergral-Derivative (PID) design as it allows for precise tuning of the controller parameters to fit a wide range of needs. Each of the three PID components serve a different purpose, the proportional component decreases rise time, the derivative component increases overshoot, while the integral component reduces the steady state error. We used the `PIDTuner()` function in Matlab to tune the PID parameters for optimal performance. Our goal was to get into and stay in the target range of 40-60 AS as quickly and safely as possible. Several trade-offs were carefully considered during the design process, mainly pertaining to response speed, overshoot, and stability. Experimenting with `PIDTuner()` inputs revealed that our system is highly susceptible to overshoot, so we decided to sacrifice the speed of the response to keep overshoot minimal, thus enhancing safety. This means that even though it may take longer for the patient to be sedated, there is a much lower chance of over administering the drug and causing adversarial effects. Another tradeoff was made between robust or transient behavior and response speed. With safety as top priority, we chose a controller with a more robust than aggressive transient behavior, as it prioritizes stability and resilience to disturbances, this leads to slower but safer response times. We felt that a more robust transient behavior can accommodate patients with varying levels of drug sensitivity, as it also minimizes response overshoot. ***Figure 5*** shows the final controller design responding to a step input.

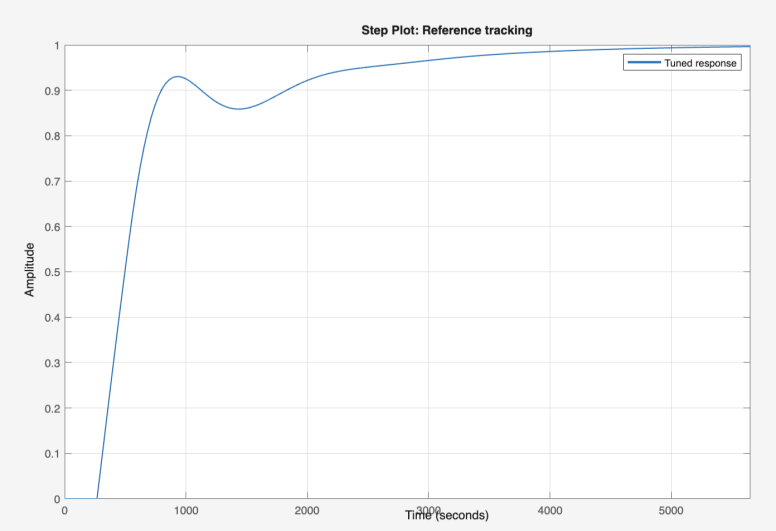

Figure 5: Step Response of the Plant paired with Controller

** Controller parameters are shown in appendix as Supplemental Figure 5.1.*

Controller Performance. Analyzing the PID-controller system, we can see that it has a Gain Margin of 8.39 dB @0.00623 rad/s, a Phase Margin of 76.5 deg @0.00195 rad/s, has a rise time of 485 seconds, nd 0% overshoot and a stable closed loop performance, as shown in Figure 5 above. To further evaluate the performance of the controller, we will connect it with the patient-pump plant system to examine settling time and overshoot for a range of patient sensitivities (±10%), as well as steady state error. ***Figure 6*** below shows the EEG Alertness Response of patients with varying levels of Sensitivity. We can see that problematic overshoot is only observed in highly sensitive patients with drug sensitivities close to 1.1, or 10% above average. For all other patients, the overshoot is within a tolerable range. For consistency purposes, we will define settling time as time at which response is last outside a 10% range from target, or 45 ± 10% (49.5 - 40.5). As seen from ***Table 7*** below, the settling time for all patients are within 3 minutes, or 120 seconds, proving that the PID controller and its specific parameters are an excellent choice for controller patient sedation.

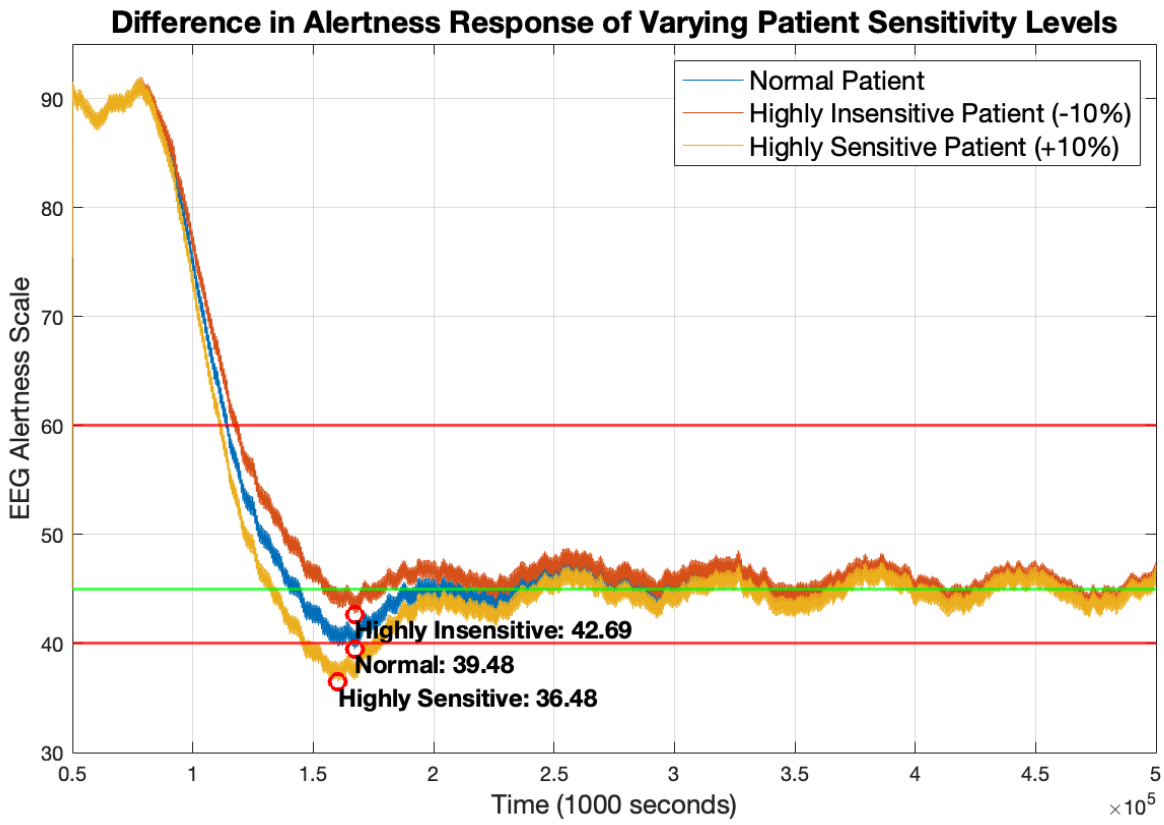

Figure 6: Patient EEG Alertness Response with Varying Sensitivity Levels

** The code for calculating overshoot is shown in appendix as Supplemental Figure 6.1.*

Patient Sensitivity	***Settling Time (sec)***	***Overshoot (% from 45)***
Normal	169.3	12.25%
Sensitive	179.9	18.9%
Insensitive	143.2	5.13%

Table 7: Settling Time and Overshoot of Patients with Varying Sensitivity Levels

** The code for calculating settling time is shown in appendix as Supplemental Figure 7.1.*

To examine steady state error, we first need to calculate the loop transfer function of the patient-pump system with the PID controller. Since the system has unity feedback, the closed loop transfer function is simply $H_{Closed-Loop} = \frac{G(s)*H(s)}{1+G(s)*H(s)}$. Notice that the loop transfer function is simply $G(s) * H(s) = H_{Loop}$, thus the equation becomes $H_{Closed-Loop} = \frac{G(s)*H(s)}{1+H_{Loop}}$. We can first calculate the loop transfer function with ***Equation 8, 9 & 10*** below.

$$G(s) = P + \frac{I}{s} + D\frac{N}{1+N\frac{1}{s}} = -0.012292 + \frac{-1.3579\times10^{-5}}{s} + (-0.96794) \times \frac{100}{1+100\times\frac{1}{s}} = \frac{-96.81s^2-1.229s-1.3579\times10^{-3}}{s^2+100s}$$

Equation 8: Calculated Transfer Function of the PID-Controller

$$H_{Loop} = G(s) * H(s)$$
$$= \left(\frac{-96.81s^2-1.229s-1.3579\times10^{-3}}{s^2+100s}\right) \times \left(\frac{-0.002262}{s^2+0.01201s+2.262\times10^{-5}}e^{-266.5788s}\right)$$
$$= \frac{0.219s^2+0.00278s+3.072\times10^{-6}}{s^4+100s^3+1.201s^2+0.002262s}e^{-266.5788s}$$

Equation 9: Loop Transfer Function of the Combined Controller and Plant System

Now, using the final value theorem, we can calculate the steady state error of the system with $H_{Closed-Loop}\frac{H_{Loop}}{1+H_{Loop}}$. ***Equations 10, & 11*** below shows the Final Value Theorem and the calculation of the steady state error to a step signal. Our PID controller design is successful in eliminating the steady state error.

$$lim_{t\to\infty} = lim_{s\to0}\, s * H(s)$$

Equation 10: Final Value Theorem

$$Error_{SS} = lim_{t\to\infty}(u(t) - y(t)) = 1 - lim_{s\to0}\, s * \left(\frac{1}{s} * \frac{H_{Loop}}{1+H_{Loop}}\right)$$

$$= 1 - lim_{s\to0}\left(\frac{\frac{0.219s^2+0.00278s+3.072\times10^{-6}}{s^4+100s^3+1.201s^2+0.002262s}e^{-266.5788s}}{1+\frac{0.219s^2+0.00278s+3.072\times10^{-6}}{s^4+100s^3+1.201s^2+0.002262s}e^{-266.5788s}}\right) = 1 - \frac{\infty}{1+\infty} \approx 0$$

Equation 11: Applying FVT on the Closed Loop Transfer Function

Controlled System Analysis. To acquire the entire controlled system performance, we evaluated its stability by gain and phase margin and Nyquist plot from the loop transfer function we calculated (***Equation 9***). According to the bode plot generated in MATLAB and parameters calculated by `margin()` function, the gain margin of the loop TF is 8.39dB at phase crossover frequency of 0.00623 rad/s, while the phase margin is 76.5 degrees at gain crossover frequency of 0.00195 rad/s. Since both gain margin in dB and phase margin in degree are larger than 0, both margin values prove the stability of the controlled system.

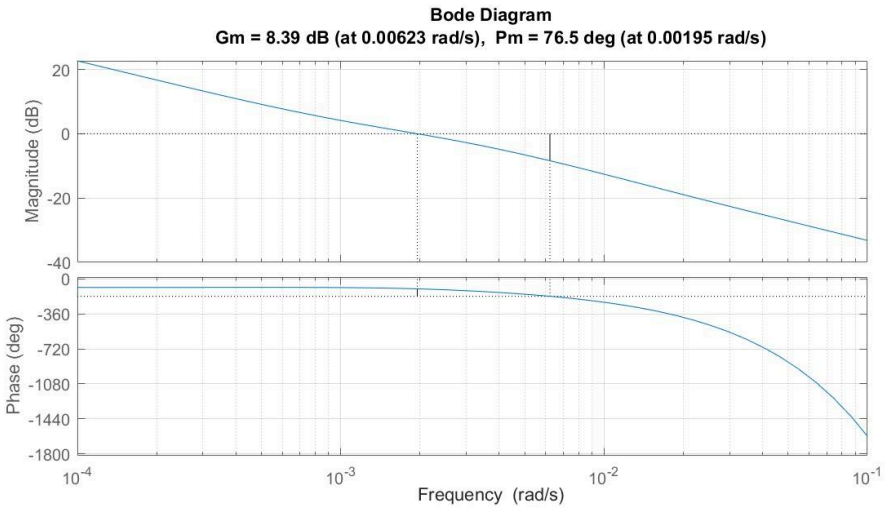

Figure 13: Bode Plot for controlled system's Loop Transfer Function

The Nyquist plot generated by the MATLAB function `nyquist()` further exhibits the stability of the controlled system. Since the (-1,0) point, which refers to the Gain 1 at 180 degrees phase, lies to the left side of the driver side window, the controlled system can be qualified as a stable system.

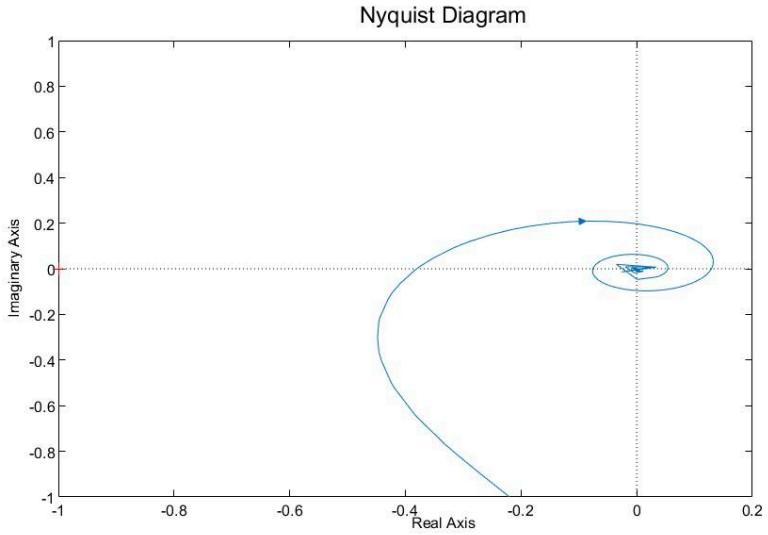

Figure 14: Nyquist Plot for Controlled System's Loop Transfer Function

Analysis for Cloud Feasibility. To determine if our controller is compatible with a cloud delay, we implemented a transport delay block between our controller and plant in the main simulation model. We then chose a delay of 70ms, which is the average cloud delay between US East and West coast, taken from the most popular cloud providers **[3]**. Our results showed that our controller is feasible for cloud deployment, keeping the average patient response above 35. We also chose to test 2x the average US E-W delay 140ms, which is the maximal delay tolerable to prevent severe risk of long-term complications. For reference, the average New York - London

delay among popular providers is 80ms, so our controller can potentially be deployed internationally, too. ***Figure 15*** plots the results below.

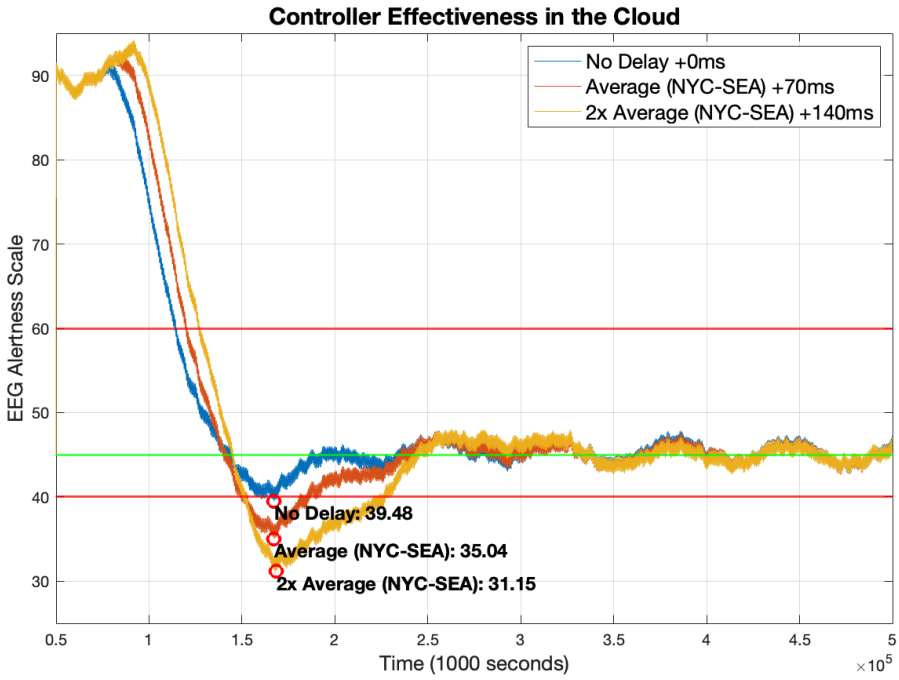

Figure 15: Controller Effectiveness when deployed on the Cloud
** Calculated in a similar manner as Overshoot in Supplemental Figure 6.1.*
***Supplemental Figure 15.1 shows where delay is added.*

Closing Remarks. After careful consideration and analysis, consultants of UICU believe that the PIDDY controller represents an excellent choice for anesthesia infusion systems. Our controller offers robust and precise control for patient sedation, working for a wide variety of patients of different drug sensitivity ranges. By leveraging advanced system identification techniques, parameter tuning, and a design focused on safety, the PIDDY controller ensures minimal steady-state error and fast settling times, saving customers time and money during prolonged use periods. The stability of the controller was verified using phase and gain margin, Nyquist analysis, to ensure its performance across a diverse set of clinical scenarios. Additionally, the controller's adaptability to cloud-based implementation allows for remote-control opportunities, ensuring scalability and accessibility in modern healthcare settings. With safety, precision, and patient outcomes in mind, the PIDDY controller represents UICU at the forefront of delivering the best tools for patient care that's currently available.

Reference.
Appendix.
Sources:
[1] Jung, Y. S., Han, Y.-R., Choi, E.-S., Kim, B.-G., Park, H.-P., Hwang, J.-W., & Jeon, Y.-T. (2015). The optimal anesthetic depth for interventional neuroradiology: comparisons between light anesthesia and deep anesthesia. Korean Journal of Anesthesiology, 68(2), 148–148. https://doi.org/10.4097/kjae.2015.68.2.148
[2] P Deepakfranklin, & M Krishnamoorthi. (2018). Monitoring Multiple Biomedical Parameter to Automate Anesthesia Injector Using FPGA. 1–5. https://doi.org/10.1109/icccsp.2018.8452862
[3] Pandey, H. (2022, August 11). Cloud Latency Shootout. Hathora. https://blog.hathora.dev/cloud-latency-shootout/

Supplemental Figure 2.1: Code for optimized input generation.

```
%% Two random square waves
u1 = 0.418 * square(2*pi*(1/500)*t);
u2 = 0.73 * square(2*pi*(1/1360)*t);
u = u1 + u2;
u = max(u, 0);
% Adding custom pulses
u(9301:10000) = 1.8 * ones(700, 1);
u(20001:50000) = 0.32 * ones(30000, 1);
u(1:2000) = 2 * ones(2000, 1);
u(15601:16000) = 4.2*ones(400, 1);
u(25001:40000) = zeros(15000, 1);
u(25601:26200) = 4.3*ones(600, 1);
u(30501:31000) = 4.7*ones(500, 1);
u(35401:36300) = 3.43*ones(900, 1);
u(45651:46000) = 5*ones(350, 1);
u(47001:50000) = zeros(3000, 1);
u(50601:51000) = 2.45*ones(400, 1);
u(55601:56000) = 3.8*ones(400, 1);
% Adding randomization
for i = 51:500:length(u)
    bias = 0.3;
    for j = 1:2:50
        u(i + j) = u(i + j) + bias;
        u(i - j) = u(i - j) - bias;
    end
end
% Adding zeros
u(1:1000) = zeros(1000, 1);
% Removing negative input
u = max(u, 0);
```

Supplemental Figure 4.1: Code for lsim() and error function used in predicting system response.

```
p = 2200;
N = 3*p;
T = 0.1;
t = tOut;

global uu tt data
uu = u;
tt = t;
data = yOut;

theta_init = [0.01; 0.01; 0.01; 90; 400];  % [a; b; c; d; e]
% Parametric curve fit
OPTIONS = optimoptions('lsqnonlin', ...
                       'Algorithm', 'Levenberg-Marquardt', ...
                       'MaxFunctionEvaluations', 150000, ...
                       'FunctionTolerance', 1e-10);
theta = lsqnonlin(@(theta) fn_linmodel(theta), theta_init, [], [], OPTIONS);
```

```
Local minimum possible.
lsqnonlin stopped because the relative size of the current step is less than
the value of the step size tolerance.

<stopping criteria details>
```

```
s = tf('s');
num = theta(1);
den = [1 theta(2) theta(3)];
Hs = tf(num, den);
% remove negative delay
if (theta(5) < 0)
    theta(5) = 0;
end
HsActual = Hs*exp(-s*theta(5));
y_hat = lsim(HsActual, uu, tt) + theta(4); % New Model with delay/offset
```

```
function err = fn_linmodel(theta)
    global uu tt data

    % Create transfer function with current parameters
    s = tf('s');
    num = [theta(1)];  % a
    den = [1 theta(2) theta(3)];  % s^2 + bs + c
    Hs = tf(num, den);


    % remove negative delay
    if (theta(5) < 0)
         theta(5) = 0;
    end
    HsFull = Hs*exp(-s*theta(5));
    ypred = lsim(HsFull, uu, tt) + theta(4); % New Model with delay/offset

    err = data(:) - ypred(:);
end
```

Supplemental Figure 5.1: Controller Parameters and Design.

Controller Parameters

	Tuned
Kp	-0.012292
Ki	-1.3579e-05
Kd	-0.96794
Tf	n/a

Performance and Robustness

	Tuned
Rise time	485 seconds
Settling time	3.6e+03 seconds
Overshoot	0 %
Peak	1
Gain margin	8.39 dB @ 0.00623 rad/s
Phase margin	76.5 deg @ 0.00195 rad/s
Closed-loop stability	Stable

Supplemental Figure 6.1: Code for calculating overshoot

```
[minValueNorm, minIndexNorm] = min(yNorm);
xMinNorm = tOut(minIndexNorm);
```

Supplemental Figure 7.1: Code for calculation of settling time.

```
% Find index and magnitude of overshoot
[minValueSens, minIndexSens] = min(ySens);
xMinSens = tOut(minIndexSens);
% Find settling time
highBound = 49.5;
lowBound = 40.5;
outsideRange = (yNotSen > highBound | yNotSen < lowBound);
normLastIdx = find(outsideRange, 1, 'last');
lastOutsideTime = (tOut(normLastIdx) - 500) / 1000
```

```
figure;
plot(tOut, yNorm, tOut, yNotSen, tOut, ySens)
grid on;
ylim([25 95])
xlim([50000 500000])

title('Controller Effectiveness in the Cloud', 'FontSize', 15)
hold on;

plot(xMinSens, minValueSens, 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Red circle for minimum
text(xMinSens, minValueSens, sprintf('2x Average (NYC-SEA): %.2f' , minValueSens), ...
     'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', 'FontSize', 12, 'FontWeight','Bold');

plot(xMinNotSen, minValueNotSen, 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Red circle for minimum
text(xMinNotSen, minValueNotSen, sprintf('Average (NYC-SEA): %.2f', minValueNotSen), ...
     'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', 'FontSize', 12, 'FontWeight','Bold');

plot(xMinNorm, minValueNorm, 'ro', 'MarkerSize', 8, 'LineWidth', 2); % Red circle for minimum
text(xMinNorm, minValueNorm, sprintf('No Delay: %.2f', minValueNorm), ...
     'VerticalAlignment', 'top', 'HorizontalAlignment', 'left', 'FontSize', 12, 'FontWeight','Bold');

yline(60, 'r', 'LineWidth', 1.5);
yline(40, 'r', 'LineWidth', 1.5);
yline(45, 'g', 'LineWidth', 1.5)
legend('No Delay +0ms', 'Average (NYC-SEA) +70ms', '2x Average (NYC-SEA) +140ms', 'FontSize', 13)
xlabel('Time (1000 seconds)', 'FontSize', 13)
ylabel('EEG Alertness Scale', 'FontSize', 13)
```

Supplemental Figure 15.1: *Insertion of the Delay block.*