Content

Lab 1 Report

Pre-lab Exercise: Sinusoidal Data Generation and Plotting

1. Introduction

The main objective of this pre-lab is to generate noisy sine and cosine signals, then perform data truncation and plot the results for analysis. The following are the specific implementation steps.

2. Procedure

Step 1: Generate Time Vector

A time vector t was created, ranging from 0 to 100 seconds with a step size of 0.1 seconds.

```
% Step1
t = (0:0.1:100)'; % Time column vector
```

Step 2: Generate Sinusoidal Data

Using the sin and cos functions, two sinusoidal waveforms, y1 and y2, were generated. A uniformly distributed random noise was added to both signals.

```
% Step2
y1 = sin(0.02*pi*t)+0.4*rand(size(t))-0.2; % y1
y2 = cos(0.02*pi*t)+0.4*rand(size(t))-0.2; % y2
```

Step 3: Truncate Data

The first 200 samples, corresponding to the first 20 seconds of data, were removed from the dataset. The time vector was adjusted accordingly.

```
%Step3
t_new = t(201:end)-t(201); % Cut-off first 20s
y1_new = y1(201:end); %Cut-off sin wave
y2_new = y2(201:end); % Cut-off cos wave
```

Step 4: Create New Matrix

A new matrix data_new was created, containing the truncated time, sine, and cosine values.

```
%Step 4
data_new = [t_new y1_new y2_new];  % nx3 matrix
```

Step 5: Plot the Original Data

The original sinusoidal data was plotted on a figure with two subplots. The first subplot shows the original y1 and y2 data.

```
%Step 5
figure;
subplot(2,1,1); %First plot in a 2-row, 1-column layout, 1st fig
plot(t, y1, 'r');
hold on;
plot(t, y2, 'b');
hold on; %blue & red
title('Original Data');
xlabel('Time (sec)');
ylabel('Data');
xlim([0 140]);
ylim([-1.5 1.5]);
grid on;
legend('sin(0.02*pi*t)', 'cos(0.02*pi*t)');
```

Step 6: Plot the Truncated Data

The truncated y1_new and y2_new data were plotted on the second subplot using different colors.

```
%Step 6
subplot(2,1,2);%Second plot
plot(t_new, y1_new, 'g');
hold on;
plot(t_new, y2_new, 'color', [0.6 0.7 0.8]);
hold on;
title('Cut-off Data');
xlabel('Time (sec)');
ylabel('Data');
xlim([0 140]);
ylim([-1.5 1.5]);
grid on;
legend('sin(0.02*pi*t)', 'cos(0.02*pi*t)');
```

3. Results and Analysis

Figure 1 is the plot generated by the Pre-Lab code.

Original Data: In the first subplot, the red curve represents the noisy sine wave (sin(0.02*pi*t)), and the blue curve represents the noisy cosine wave (cos(0.02*pi*t)).

Cut-off Data: The second subplot shows the truncated data. The green and gray lines represent the sine and cosine waves after the first 20 seconds of data have been removed.

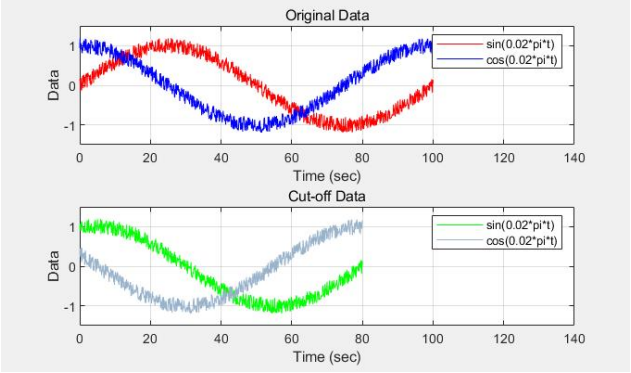


Figure 1

Lab Exercise: Identifying a Second-Order Discrete-Time Linear Model

1. Introduction

In the main experiment, a pre-collected water tank data file was used. The experiment involved removing the offset from the data and constructing a transfer function and state-space model. Afterward, the simulated output generated by the model was compared with the actual output. Here are the specific steps implemented in the experiment:

2.Processes

Data Extraction and Preprocessing

We first load the pre-collected data (SysIdenData_StudentVersion.mat) into MATLAB, which contains time (t), the actual output (y_act), and input signal (u_act). Both the noise-reduced output (y_act) and measured output (y_actm) are plotted for comparison, along with the actual input signal.

```
% Step 1
load('SysIdenData_StudentVersion.mat');
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;
```

```
% Step 2
figure;
subplot(2,1,1);
plot(t, y_act, 'b', t, y_actm, 'r');
title('Actual Output Signal');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([1 4]);
legend('Noise-Reduced Output', 'Measured Output');
grid on;

subplot(2,1,2);
plot(t, u_act, 'b');
title('Actual Input Signal');
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
legend('Actual Input');
xlim([0 700]);
ylim([1 3]);
grid on;
```

Here is the figure generated.

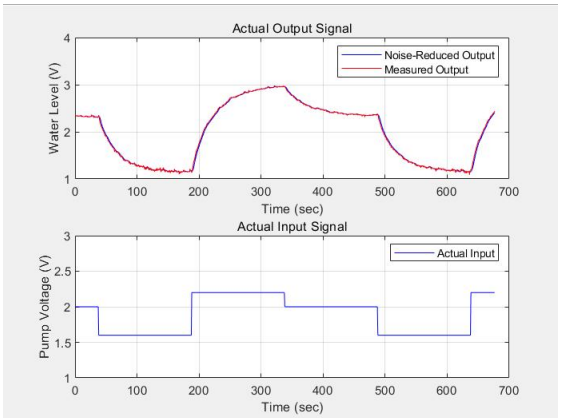


Figure 2

Removing Offset from Data

Offsets are detected and removed from both input and output signals to create offset-free data. This ensures that the model identifies the correct system dynamics.

```
% Step 3
i = 1;
while u_act(i) == u_act(1)
    i = i + 1;
end
% Output offset
u_offset = u_act(1);
u = u_act - u_offset;
% Input offset
y_offset = mean(y_act(1:i-1));
y = y_act - y_offset;
```

After removing the offsets, we plot the offset-free signals.

```
% Step 4
figure;
subplot(2,1,1);
plot(t, y, 'r');
title('Actual Offset-Free Output Signal');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-2 1]);
legend('Actual Output');
grid on;

subplot(2,1,2);
plot(t, u, 'b');
title('Actual Offset-Free Input Signal');
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
xlim([0 700]);
ylim([-0.5 0.5]);
legend('Actual Input');
grid on;
```

Here is the figure generated.

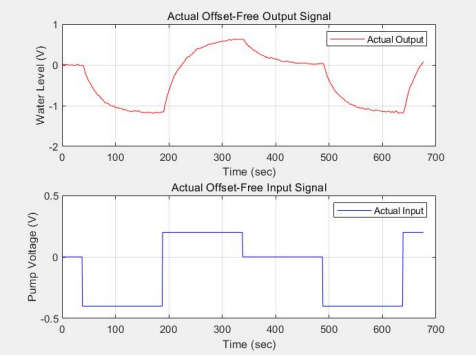


Figure 3

Second-Order Model Identification

Using the first half of the offset-free data, a second-order model is identified by constructing the matrix Φ and solving for the model parameters a_1, a_2, b_1 and b_2.

```
% Part 2 step a
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start = 3;
Y = y(k_start:N);
Phi = [y(k_start-1:N-1), y(k_start-2:N-2), u(k_start-1:N-1), u(k_start-2:N-2)];

% Part 2 step b
theta = inv((Phi' * Phi)) * (Phi' * Y);
a1 = -theta(1);  % a1
a2 = -theta(2);  % a2
b1 = theta(3);  % b1
b2 = theta(4);  % b2
```

The second-order transfer function and state-space model are derived based on the identified

parameters:

```
% Part 2 step c
% transfer function
h = 0.01;
numerator = [b1 b2];
denominator = [1 a1 a2];
Gz = tf(numerator, denominator, h);% Eq1

% state space
G = [0 1; -a2 -a1];  % matrix G
H = [0; 1];  % matrix H
C = [b2 b1];  % matrix C
D = 0;  % matrix D
sys_ss = ss(G, H, C, D, h);
```

Model Validation

To validate the model, the second half of the data is used for simulation, and the simulated output is compared to the actual offset-free output.

```
% Part3 Step a
t_half = t(N+1:end);
y_half = y(N+1:end);
t_half_shifted = t_half - t_half(1);
simulated_output = filter(numerator, denominator, u(N+1:end));

% Step b
figure;
subplot(2,1,1);
plot(t_half_shifted, y_half(),'r', t_half_shifted, simulated_output, 'b--');
title('Offset-Free Model Verification (2nd Half)');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 350]);
ylim([-2 2]);
legend('Actual Output', 'Simulated Output');
grid on;
```

The entire dataset is then used to simulate the model and compare it with the actual output across the full time period.

```
simulated_output2 = filter(numerator, denominator, u(k_start:end));
t_simulated = t(k_start:end);

subplot(2,1,2);
plot(t_simulated, y(k_start:end), 'r', t_simulated, simulated_output2, 'b--');
title('Offset-Free Model Verification (Entire)');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-2 2]);
legend('Actual Output', 'Simulated Output');
grid on;
```

Here is the generated figure.

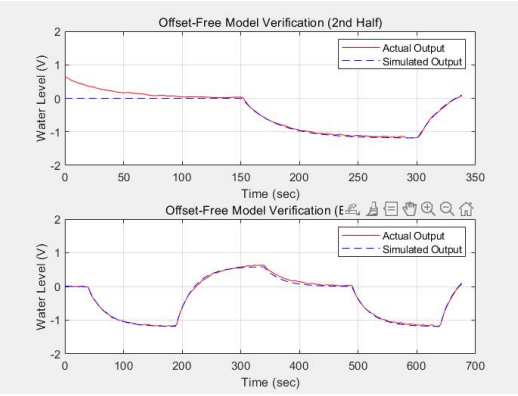


Figure 4

Post-lab Exercise: Comparison of First and Second-Order Models

1. Introduction

The goal of this post-lab exercise is to compare the accuracy of first-order and second-order discrete-time linear models in system identification. And calculate the MSE of two different order models.

2. Processes

We follow the same procedure used in the previous lab exercise to identify both first-order and second-order models using the offset-free input (u) and output (y). The two models are compared by simulating their outputs.

Step 1: Data Extraction

The first step involves loading the pre-collected data (SysIdenData_StudentVersion.mat), extracting the time, input, and output signals, and removing the offset from the signals.

```
% Step 1
load('SysIdenData_StudentVersion.mat');
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;

i = 1;
while u_act(i) == u_act(1)
    i = i + 1;
end
% Output offset
u_offset = u_act(1);
u = u_act - u_offset;
% Input offset
y_offset = mean(y_act(1:i-1));
y = y_act - y_offset;
```

Step 2: Second-Order Model Identification

Using the offset-free input-output data, the second-order model parameters are identified by creating the matrix Φ and solving the system of equations. The transfer function and state-space models are then derived.

```
% Step 2 second order model
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start = 3;
Y = y(k_start:N);
Phi = [y(k_start-1:N-1), y(k_start-2:N-2), u(k_start-1:N-1), u(k_start-2:N-2)];

theta = inv((Phi' * Phi)) * (Phi' * Y);
a1 = -theta(1);  % a1
a2 = -theta(2);  % a2
b1 = theta(3);  % b1
b2 = theta(4);  % b2

% transfer function
h = 0.01;
numerator = [b1 b2];
denominator = [1 a1 a2];
Gz = tf(numerator, denominator, h);% Eq1
```

Step 3: First-Order Model Identification

Similarly, a first-order model is identified using the same dataset, but with a simplified equation structure.

```
% Step 3 first order model
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start_2 = 3;
Y_2 = y(k_start:N);
Phi_2 = [y(k_start-1:N-1), u(k_start-1:N-1)];

theta_2 = inv((Phi_2' * Phi_2)) * (Phi_2' * Y_2);
a1_2 = -theta_2(1);  % a1
b1_2 = theta_2(2);  % b1

% transfer function
numerator_2 = b1_2;
denominator_2 = [1 a1_2];
Gz_2 = tf(numerator_2, denominator_2, h);% Eq1
```

3. Model Simulation and Comparison

After identifying both models, we simulate the system response using the entire dataset for both first and second-order models. The simulated responses are then compared with the actual output, and the MSE for each model is calculated.

```
% Simulating both models
simulated_output_1st = filter(numerator, denominator, u(k_start:end));
simulated_output_2nd = filter(numerator_2, denominator_2, u(k_start:end));
t_simulated = t(k_start:end);

% First Order MSE
MSE_1st = mean((y(k_start:end) - simulated_output_1st).^2);

% Second Order MSE
MSE_2nd = mean((y(k_start:end) - simulated_output_2nd).^2);
mse_str_1 = sprintf('%.8f', MSE_1st);
mse_str_2 = sprintf('%.8f', MSE_2nd);

figure;
subplot(1,1,1);
plot(t_simulated, y(k_start:end), 'r', t_simulated, simulated_output_1st, 'g--',t_simulated, simulated_output_2nd, 'b--');
title('Comparison of Different Offset-Free Order Models','FontWeight', 'bold');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-1.5 1]);
legend('Actual Output', '1st Order Model Response', '2nd Order Model Response');
grid on;

text(10, 0.56, ['MSE1 = ', num2str(mse_str_1)], 'FontSize', 9.5);
text(10, 0.44, ['MSE2 = ', num2str(mse_str_2)], 'FontSize', 9.5);
```

4. Results and Discussion

The comparison between the two models is visualized in the plot, with the MSE values for both models displayed on the graph. By observing Figure 5, it can be observed that the MSE of the second-order model is slightly lower than that of the first-order model, indicating a higher degree of fit between the second-order model and the actual data.

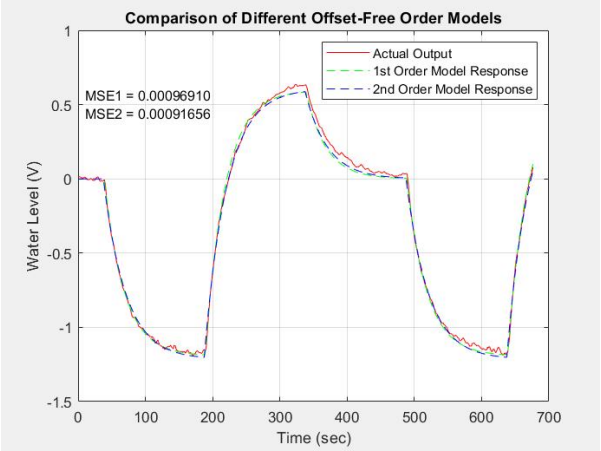


Figure 5

First-order model MSE: 0.00096910

Second-order model MSE: 0.00091656

Lab 2 Report

Lab Exercise

1. Introduction

In this experiment, the real water tank system will run under a pre-built Simulink model in MATLAB, using the collected input and water tank output data. Based on the code from Lab 1, the real water tank experimental data will be processed by removing the offset, constructing the state-space model, and performing other operations. Finally, the simulated output will be compared with the actual output.

2. Processes

Data Collection

The dataset loaded in this experiment comes from the real water tank in the laboratory. The data includes time-series measurements of the water level (y_act) and pump voltage (u_act) as they vary over time.

```
load('SysIdenData_1.mat');
```

Data Preprocessing

The raw data contains offsets and noise, which need to be removed for accurate modeling. The preprocessing steps involve the following:

1. Time Synchronization and Cropping: The data is cropped to remove irrelevant initial periods where the system is in a steady state. A cutoff time (t_cutoff) is chosen to remove this section. In this system, t_cutoff = 132s, V_max = 3V, V_min = 2.1V. Hence, the code can calculate:

```
u_offset = (V_max + V_min)/2;
t_cutoff = 132/0.75;
```

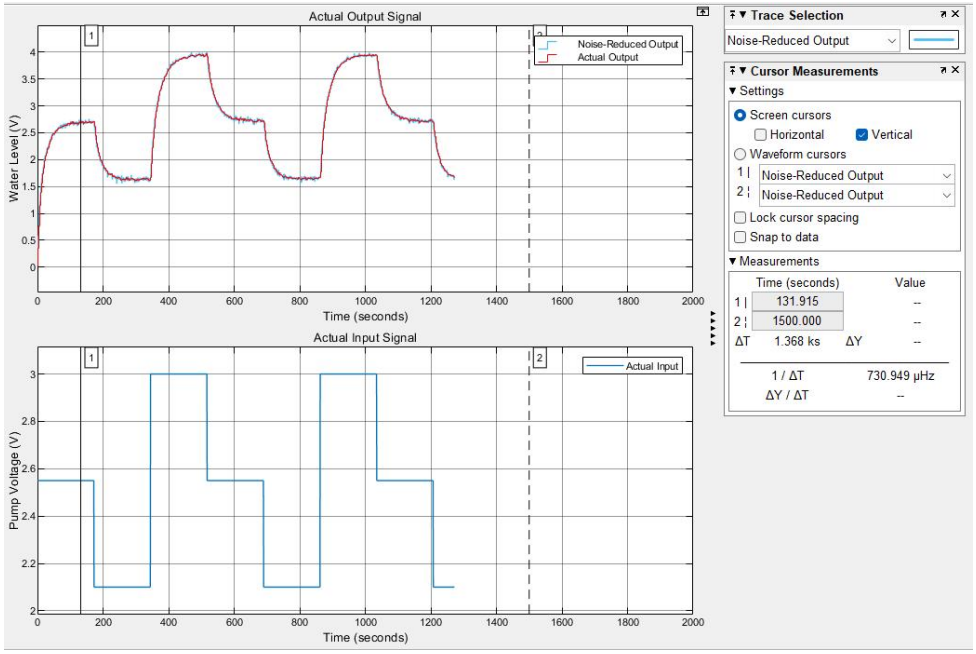


Figure 1: Collected Data from Real W-T

2. Offset Removal: By subtracting the average value during the first steady-state period, the system is ensured to start from a zero-state response.

```
t = LogData.time(t_cutoff:end)-LogData.time(t_cutoff);
y_act = LogData.signals(1).values(t_cutoff:end,2);
y_actm = LogData.signals(1).values(t_cutoff:end,1);
u_act = LogData.signals(2).values(t_cutoff:end,1);
```

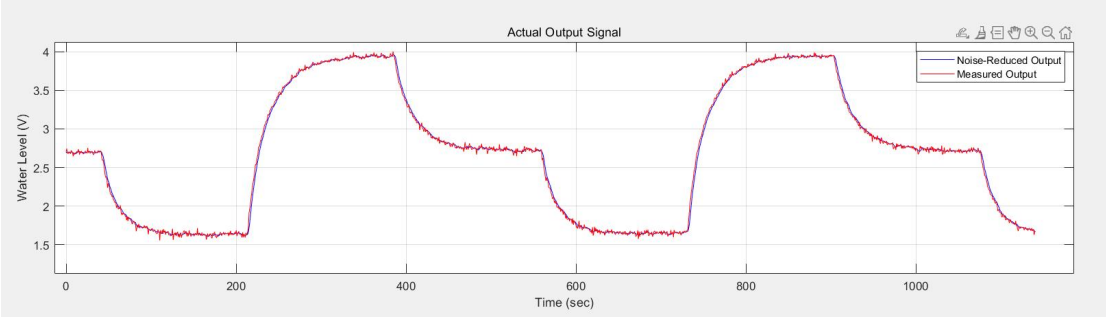


Figure 2: Comparison of noise-reduced output signal and measured output signal

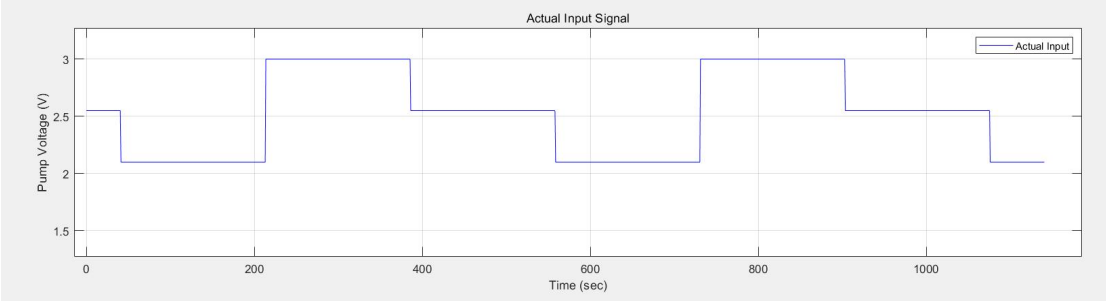


*Figure 3: Offset-free actual input signal (*u_act*).*

System Identification

Once the data is preprocessed, we proceed with system identification using the following steps:

1. Data Division: The data is divided into two parts, with the first half used for identifying the system parameters, and the second half used for model validation.

```
% Part 2 step a
N = floor(length(y) / 2); % the first half of the offset-free input-output data
```

2. Model Structure: Using state space form and transfer function to describe the system

```
% Part 2 step a
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start = 3;
Y = y(k_start:N);
Phi = [y(k_start-1:N-1), y(k_start-2:N-2), u(k_start-1:N-1), u(k_start-2:N-2)];
theta = inv((Phi' * Phi)) * (Phi' * Y);

% Part 2 step b
a1 = -theta(1);  % a1
a2 = -theta(2);  % a2
b1 = theta(3);  % b1
b2 = theta(4);  % b2

% Part 2 step c
% transfer function
h = t(2)-t(1);
numerator = [b1 b2];
denominator = [1 a1 a2];
Gz = tf(numerator, denominator, h);% Eq1

% state space
G = [0 1; -a2 -a1];  % matrix G
H = [0; 1];  % matrix H
C = [b2 b1];  % matrix C
D = 0;  % matrix D
sys_ss = ss(G, H, C, D, h);
```

Model Verification

The identified model is validated by comparing its simulated output with the actual output. The following steps are involved in simulation:

1. **Simulation for the Second Half of Data:**

   The second half of the dataset is used for model validation.

   (1) Model Verification (Second Half) (Figure 3):

   This plot shows the comparison of the actual output with the simulated output for the second half of the data. The actual output is plotted in red, and the simulated output is plotted in blue.

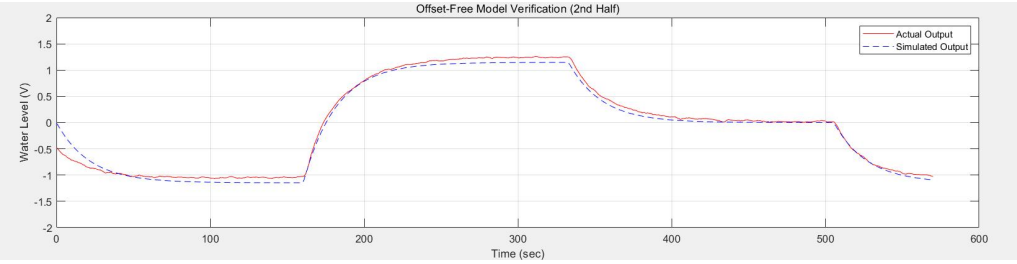


Figure 4: Actual output vs. simulated output for the second half of the dataset.

2. Simulation for the Entire Dataset:

Simulation is also performed over the entire dataset to compare the actual output with the predicted output.

(2) Model Verification (Entire Dataset) (Figure 4):

This plot compares the actual output and the simulated output over the entire dataset.

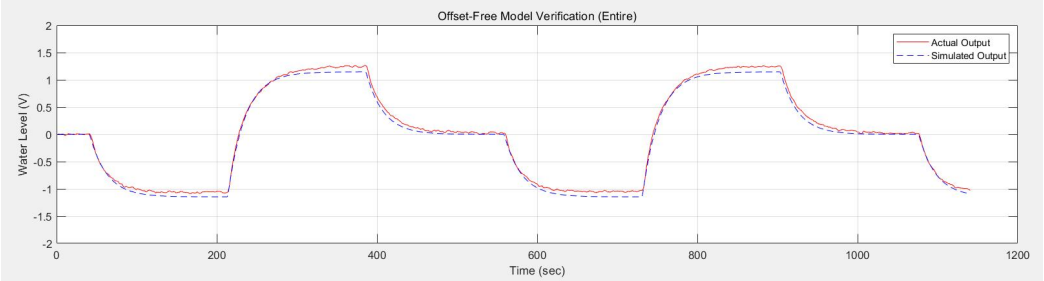


Figure 5: Actual output vs. simulated output for the entire dataset.

4. Results

4.1 Data Preprocessing

The preprocessing steps successfully removed noise and offsets from the original signals. The following plots show the raw and offset-free signals:

Actual Output Signal (Noise-Reduced):

As shown in Figure 2, the noise-reduced output signal (y_act) is compared with the measured output signal (y_actm), demonstrating the effectiveness of noise reduction.

Actual Input Signal:

As shown in Figure 3, the actual input voltage (u_act) over time is displayed.

4.2 System Identification

The identified transfer function and state-space model were successfully derived. The transfer function representing the system is:

$$G(z) = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2}$$

where a_1, a_2, b_1 and b_2 are the estimated parameters.

4.3 Model Verification

The simulated output based on the identified model was compared with the actual output. The results indicate that the model's response closely matches the actual system behavior, confirming the validity of the identified model.

Figures 4 and 5 show the comparison between actual and simulated outputs for both the second half and the entire dataset, verifying the model's accuracy.

Discussion

Since the **Repeating Sequence Stair** block outputs zero during its first period, the initial input voltage in the simulation is zero. However, the actual system's initial voltage is u_{offset}, which is a segment we extracted from the actual waveform and is not starting from zero. Therefore, as shown in **Figure 4**, the simulated and actual outputs do not match at the initial moment.

5. Conclusion

This experiment demonstrated the process of system identification using input-output data. We successfully preprocessed the data, identified the system parameters, and derived both a transfer function and a state-space model. The model was validated through simulation, showing good agreement with the actual system response.

Lab 3 Report

Pre-lab Exercise

1. Introduction

This pre-lab exercise is part of a larger experiment to design a state feedback control system with an observer for both regulation and set-point control (output feedback control) of a W-T system model. The goal is to become familiar with the system identification process and the properties of the resulting state-space representation.

2. Objective

The primary objective of this pre-lab is to recreate the transfer function and state-space models of the identified system from Lab 2, determine if the open-loop system is stable or not, verify if the identified system is a minimum phase system, investigate the reachability (controllability) and observability of the system, and derive the canonical observable form of the state-space model using the duality property.

3. Procedure

To achieve the objectives, the following steps were performed:

The provided system identification data from Lab 2 was first loaded and preprocessed. The input and output offsets were removed by subtracting the mean values from the respective signals. The first half of the offset-free input-output data was then used for the analysis.

```
% LAB_02_DATA
load('SysIdenData_1.mat');
V_min = 2.1;
V_max = 3;
u_offset = (V_max + V_min)/2;
t_cutoff = 132/0.75;
t = LogData.time(t_cutoff:end)-LogData.time(t_cutoff);
y_act = LogData.signals(1).values(t_cutoff:end,2);
y_actm = LogData.signals(1).values(t_cutoff:end,1);
u_act = LogData.signals(2).values(t_cutoff:end,1);
i = 1;
while u_act(i) == u_act(1)
    i = i + 1;
end
% Output offset
u = u_act - u_offset;
% Input offset
y_offset = mean(y_act(1:i-1));
y = y_act - y_offset;
```

Next, the system parameters a1, a2, b1, and b2 were estimated using a least-squares method. These parameters were then used to construct the discrete-time transfer function G(z) and the state-space representation (G, H, C, D).

```
N = floor(length(y) / 2); % the first half of the offset-free input-output data
k_start = 3;
Y = y(k_start:N);
Phi = [y(k_start-1:N-1), y(k_start-2:N-2), u(k_start-1:N-1), u(k_start-2:N-2)];
theta = inv((Phi' * Phi)) * (Phi' * Y);
a1 = -theta(1);  % a1
a2 = -theta(2);  % a2
b1 = theta(3);  % b1
b2 = theta(4);  % b2

% transfer function
h = t(2)-t(1);
numerator = [b1 b2];
denominator = [1 a1 a2];
Gz = tf(numerator, denominator, h);% Eq1

% state space
G = [0 1; -a2 -a1];  % matrix G
H = [0; 1];  % matrix H
C = [b2 b1];  % matrix C
D = 0;  % matrix D
sys_ss = ss(G, H, C, D, h);
```

The stability of the open-loop system was determined by computing the eigenvalues of the state matrix G. If all the eigenvalues had magnitudes less than 1, the system was considered stable. The zeros of the transfer function G(z) were also found to check if the system is minimum phase, which requires all zeros to be inside the unit circle.

```
% Step 1_a: Calculate eigenvalues of the state matrix G & determie whether stable
eig_G = eig(G);
disp('Eigenvalues of the system:');
disp(eig_G);
if all(abs(eig_G) < 1)
    disp('The open-loop system is stable.');
else
    disp('The open-loop system is unstable.');
end

% Step 1_b: Find zeros of the transfer function G(z)
zeros_Gz = zero(Gz);
disp('Zeros of the transfer function:');
disp(zeros_Gz);
% Check if the system is minimum phase: A minimum phase system has all its zeros inside the unit circle.
if all(abs(zeros_Gz) < 1)
    disp('The system is a minimum phase system.');
else
    disp('The system is not a minimum phase system.');
end
```

The reachability (controllability) and observability of the system were evaluated by computing the ranks of the controllability and observability matrices, respectively. If the ranks were equal to the dimension of the state-space model, the system was considered reachable and observable.

```
% Step 2: Check reachability (controllability) of (G, H)
controllability_matrix = ctrb(G, H);
rank_ctrb = rank(controllability_matrix);
disp('Rank of the controllability matrix:');
disp(rank_ctrb);

if rank_ctrb == size(G, 1)
    disp('The system is reachable (controllable).');
else
    disp('The system is not reachable (controllable).');
end

% Check observability of (G, C)
observability_matrix = obsv(G, C);
rank_obsv = rank(observability_matrix);
disp('Rank of the observability matrix:');
disp(rank_obsv);

if rank_obsv == size(G, 1)
    disp('The system is observable.');
else
    disp('The system is not observable.');
end
```

Finally, the canonical observable form of the state-space model was derived using the duality property. This involved transposing the state matrix G and swapping the input and output matrices H and C, respectively.

```
% Step 3: Find the canonical observable form using duality property
G_T = G';
H_T = C';  % Transpose of C becomes the new input matrix
C_T = H';  % Transpose of H becomes the new output matrix
D_T = D';

% Create the state-space representation in the canonical observable form
sys_obs = ss(G_T, H_T, C_T, D_T, h);
```

4. Results and Analysis

The open-loop system was found to be stable, as all the eigenvalues of the state matrix G had magnitudes less than 1. However, the system was determined to be non-minimum phase, as one of the zeros of the transfer function G(z) was located outside the unit circle. The system was confirmed to be both reachable (controllable) and observable, as the ranks of the controllability and observability matrices were equal to the dimension of the state-space model. The canonical observable form of the state-space model was successfully derived using the duality property.

```
Eigenvalues of the system:
    0.2995
    0.9655

The open-loop system is stable.
Zeros of the transfer function:
   -1.6786

The system is not a minimum phase system.
Rank of the controllability matrix:
     2

The system is reachable (controllable).
Rank of the observability matrix:
     2

The system is observable.
```

5. Conclusion

The pre-lab exercise has provided a solid foundation for the upcoming lab work. By recreating the system identification results from Lab 2 and analyzing the properties of the state-space model, the groundwork has been laid to design the state feedback control system with an observer. The understanding of system stability, phase properties, reachability, and observability will be crucial in the next stages of the experiment.

Lab Exercise

1. Regulation by state feedback

According to the requirements in the manual, it is necessary to ensure that y(0) = Cx(0) = 0.3.Then, we can calculate the value of x(0).

```
% Q1
% Given y(0) = C * x(0) = 0.3
y0 = 0.3;  % Initial offset-free output

% Assume y(k) = x2(k) in the observable form, so we set:
x2_0 = y0;  % Set x2(0) directly to y(0)

% Initial conditions for x(0)
x0 = [0; x2_0];  % Assuming x1(0) = 0
```

```
Chosen initial conditions for state vector x(0):
         0
    0.3000
```

Then, calculate the state feedback gains L_{db} and L_{ndb} for deadbeat control and non-deadbeat control, respectively. Use the MATLAB function *acker* to obtain these feedback gains, and select appropriate closed-loop pole positions as suggested.

```
% Deadbeat control
% Choose desired closed-loop poles for deadbeat control (e.g., both at the origin)
desired_poles_db = [0 0];
L_db = acker(G_T, H_T, desired_poles_db);  % Calculate deadbeat control gain
disp('Deadbeat control gain (L_db):');
disp(L_db);

% Non-deadbeat control
% Choose desired closed-loop poles for non-deadbeat control (e.g., close to open-loop poles)
% Adjust these values based on your system's requirements
desired_poles_ndb = [0.4 0.9]; % You may need to adjust these for stability and input limits
L_ndb = acker(G_T, H_T, desired_poles_ndb);  % Calculate non-deadbeat control gain
disp('Non-deadbeat control gain (L_ndb):');
disp(L_ndb);
```

Below are the calculated values of L_{db} and L_{ndb} for the system.

```
Calculated water level (ml): 179.748
Deadbeat control gain (L_db):
   20.0854   21.3674

Non-deadbeat control gain (L_ndb):
   -1.0784    0.2872
```

Next, build the responses of these closed-loop control systems in Simulink and observe the performance of deadbeat and non-deadbeat control.

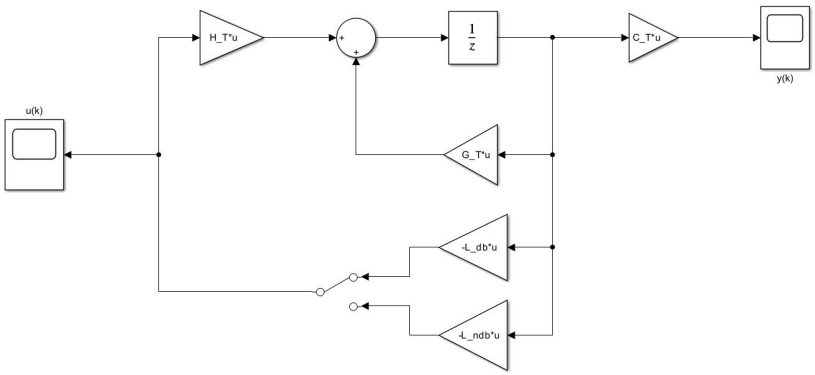


Figure 1. Two closed-loop control model

From Figure 2 and Figure 3, it can be observed that deadbeat control responds faster than non-deadbeat control. However, deadbeat control exhibits overshoot, far exceeding the set threshold of ±0.5V. The overshoot could cause significant damage to the system.

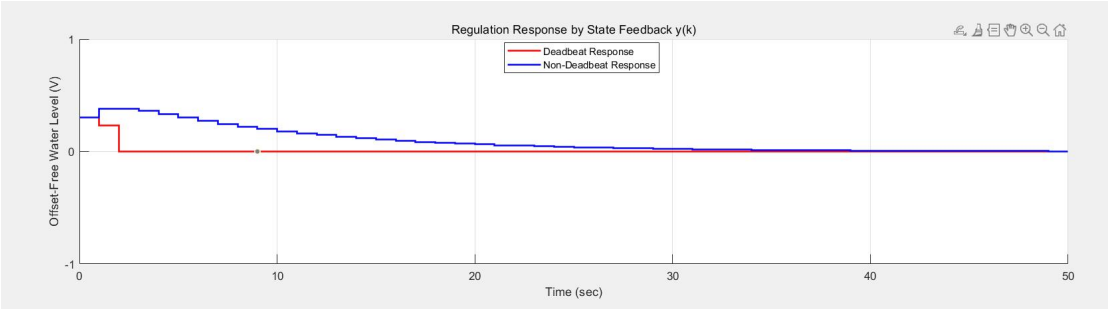


Figure 2. regulation response by state feedback

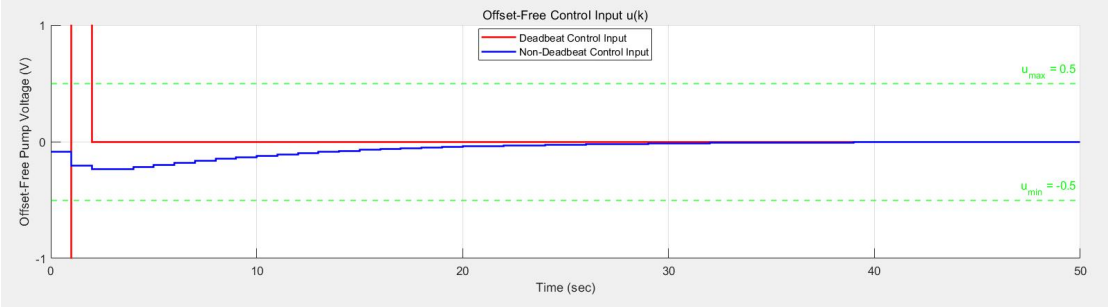


Figure 3. Offset-Free control input

2. Set-point control

According to the requirements in the manual, the reference signal for the system is set as {0, 0.7, -0.2, 0.5, 0}, with a period of 140×0.75 = 105 seconds for each level.

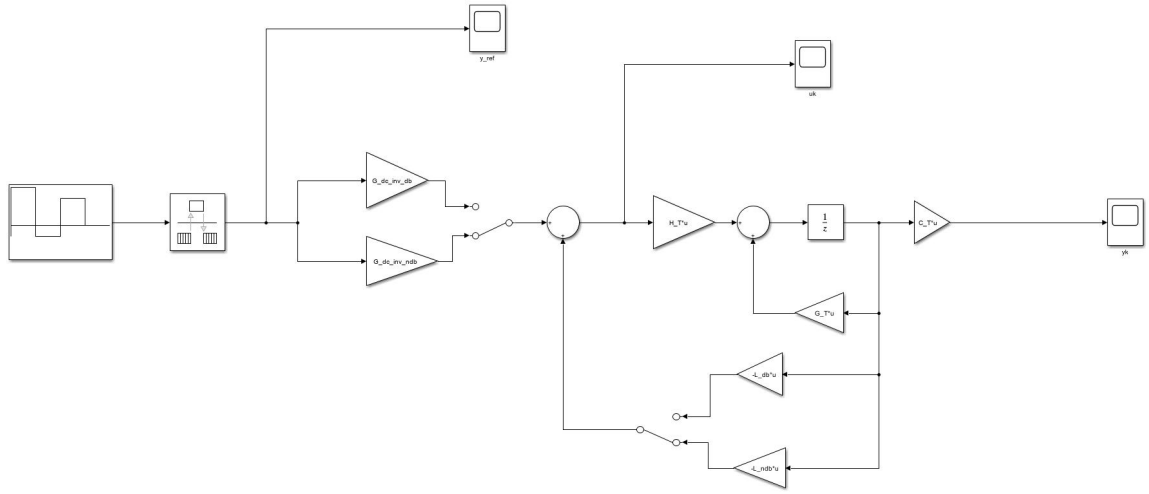


Figure 4. Set-Point Control Model

From Figure 5, it can be observed that the simulated output successfully tracks the reference output, and the control input remains stable within the ±0.5V range.

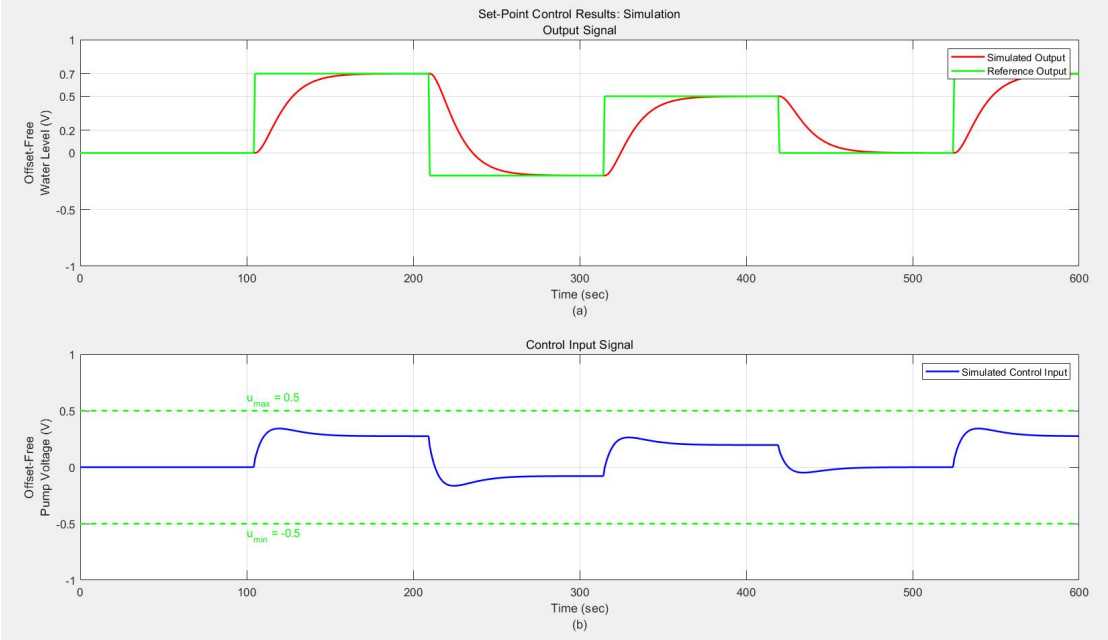


Figure 5. Set-Point Control Model

3. Output feedback control (Set-point control with observer)

The following is the simulation model of the output feedback control, and the observer gain is to be determined.

```
Observer gain L_ndb:
  -12.6739   -3.7617
```

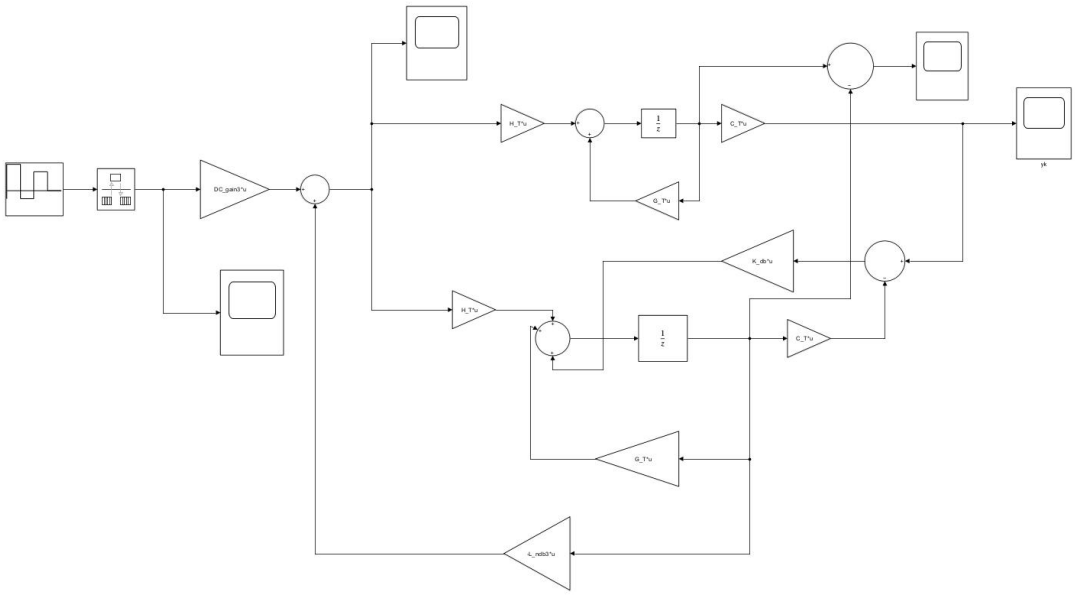


Figure 6. Output feedback control Model

It can be observed that the output response tracks the reference output well, and the input voltage remains within the voltage threshold of ±0.5V. There is a difference in the initial values between the observer and the actual system, but after one sampling period, both converge to the same value.

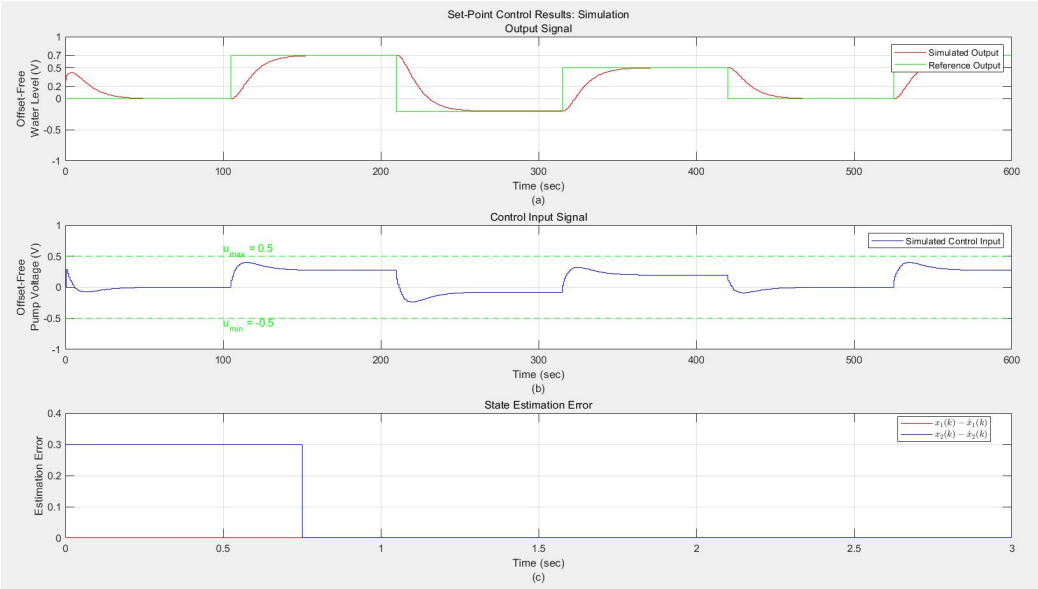


Figure. 7. Set-point control response with observer and non-zero initial conditions, (a) Output response compared with reference output, (b) Control input, and (c) State estimation error.

Conclusion

In this experiment, we learned and practiced the process of designing a closed-loop control system using the state feedback approach for a water tank system. First, we calculated the state feedback gains for deadbeat and non-deadbeat control using the Ackermann function and MATLAB's acker() function, and analyzed the system's dynamic response under different control strategies. Next, we used Simulink to build models for the closed-loop control system, set-point control, and output feedback control system with an observer. During the design process, we calculated the observer gain using the duality principle and optimized the control system's performance by combining state feedback and the observer. The results showed that the designed control system successfully tracked the reference output, and the control input remained within the ±0.5V limit, proving the feasibility and stability of the system under the given constraints.

Lab 4 Report

Lab Exercise

1. Introduction&Objective

The main objective of this experiment is to implement the non-deadbeat output feedback control system, designed in Experiment 3, onto the actual water tank system (W-T) and evaluate its performance. Due to changes in the experimental setup (such as valve positions and system characteristics), it is necessary to repeat the system identification process and adjust the previously designed control algorithm for the new system model, ensuring that the system operates stably and meets the performance requirements.

2. Process

(1) System Identification

Repeat Lab2 and here is the new data of the W-T system.

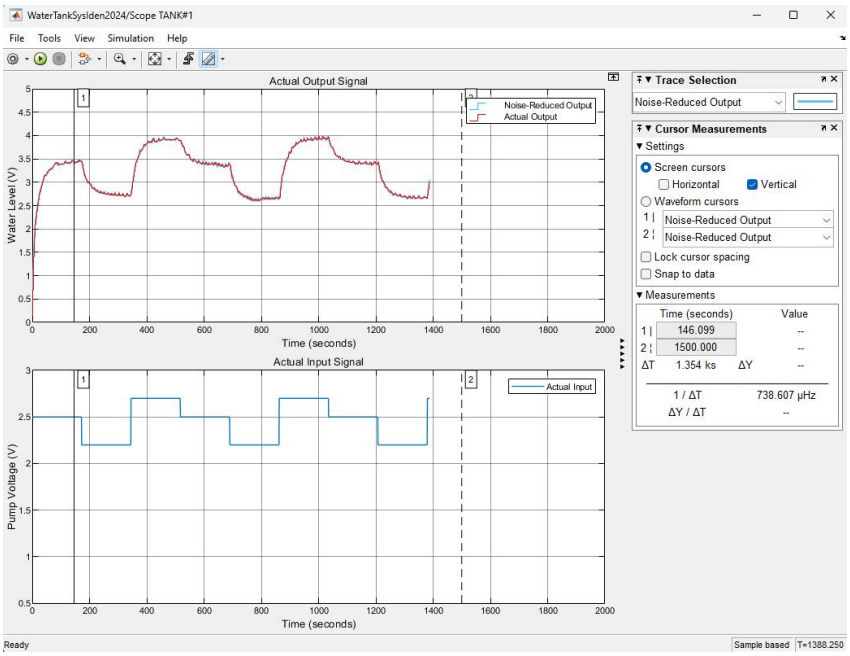


Figure 1. Collection Data

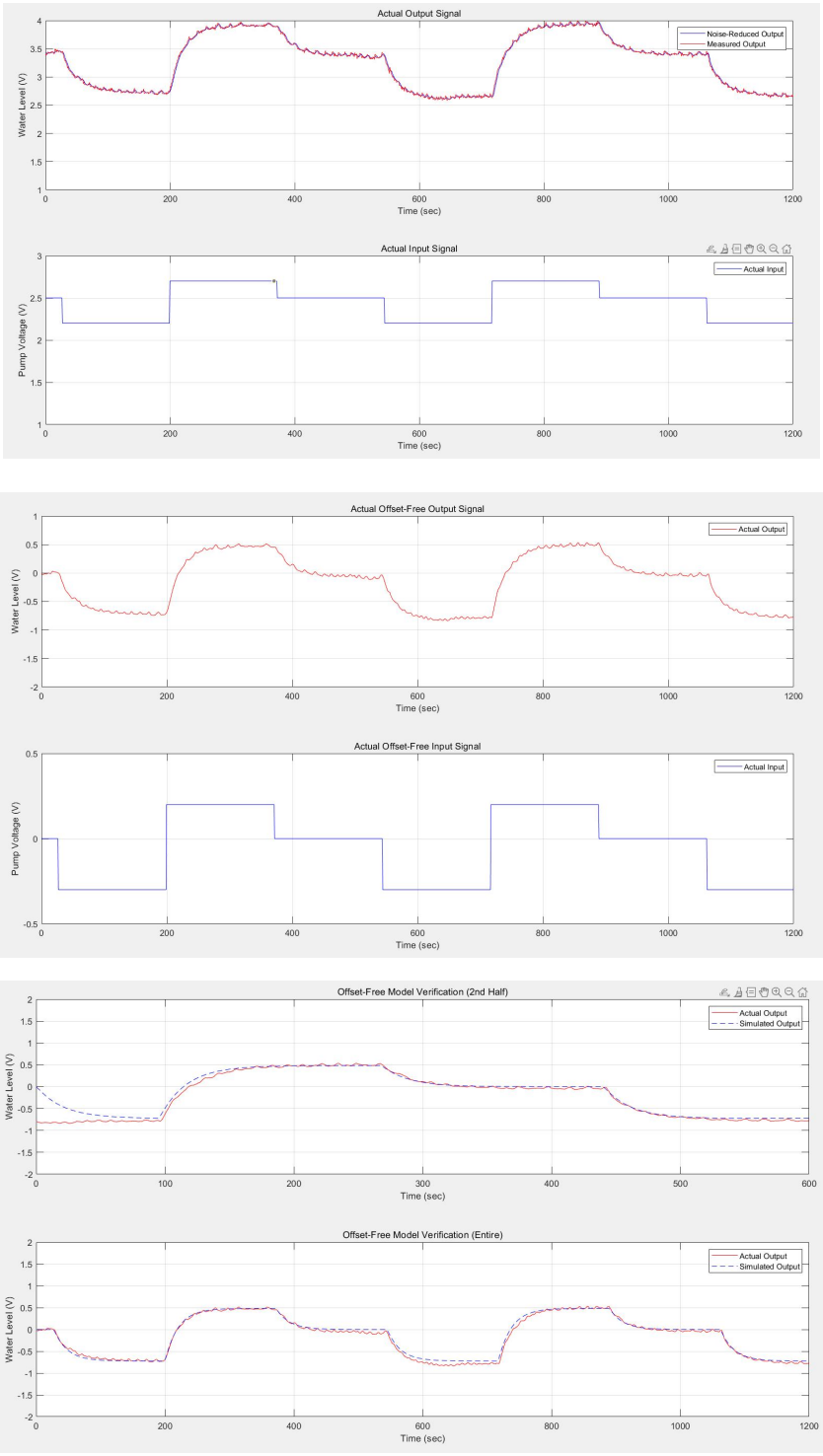


Figure 2. Results repeat Lab 2

(2) Output Feedback Control Design

Then repeat the Lab 3, the goal of the control system is to track the reference signal y{k} = {0,0.7,−0.2,0.5,0} as quickly as possible, with each level period being 140×0.75=105s.

These parameters are specified in the MATLAB workspace for the Simulink model.

```
System Model Matrices:
G (State Matrix):
         0    1.0000
   -0.5539    1.5385

H (Input Matrix):
     0
     1

C (Output Matrix):
    0.0106    0.0265

Input and Output Offsets:
u_offset (Input Offset):
    2.5000

y_offset (Output Offset):
    3.4406

Initial Conditions:
x0 (Initial State):
     0
     0

y0 (Initial Output):
    0.3000
```

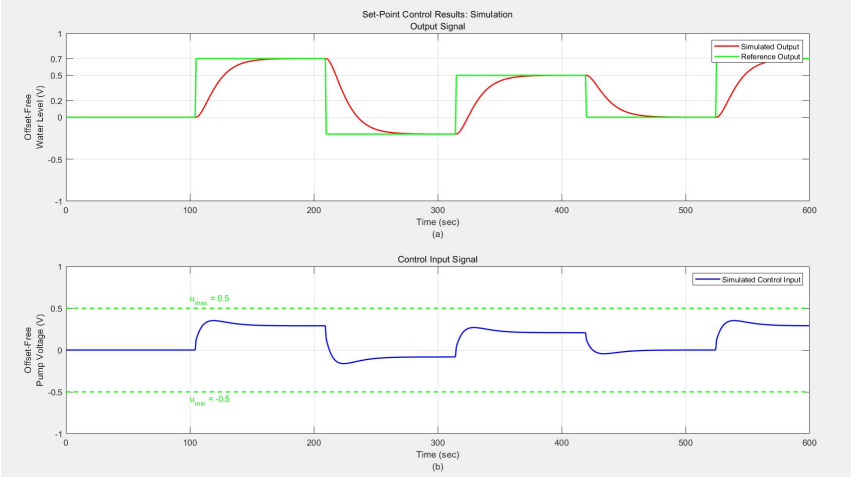
```
State Feedback Gains:
L_db (Deadbeat Feedback Gain):
   33.1393   44.7041

L_ndb (Non-Deadbeat Feedback Gain):
  -11.7053   -6.6742

Inverse of the DC Gain:
G_dc_inv_db (Deadbeat Inverse DC Gain):
   26.9079

G_dc_inv_ndb (Non-Deadbeat Inverse DC Gain):
    0.1722
```

From the figure, it can be observed that the simulated output tracks the reference voltage well, and the simulated control input is also kept within the ±0.5V voltage threshold range.



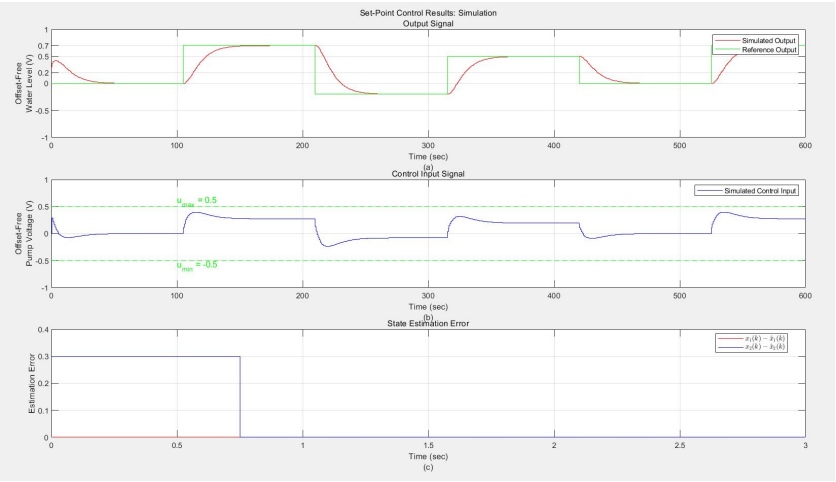


Figure 3. Results repeat Lab 3

(3) Real-time implementation of the control system

After the real-time implementation of the control system is completed, the next step is to extract data from both the simulation and the actual experiment, and perform a detailed comparison of the results.

These graphs show the time-varying output feedback control results and control input signals. The actual output (red) tracks the reference output (green), indicating that the control system is running as expected. However, the actual control input experienced a brief overshoot at the front end of the water tank startup, exceeding the set voltage threshold.

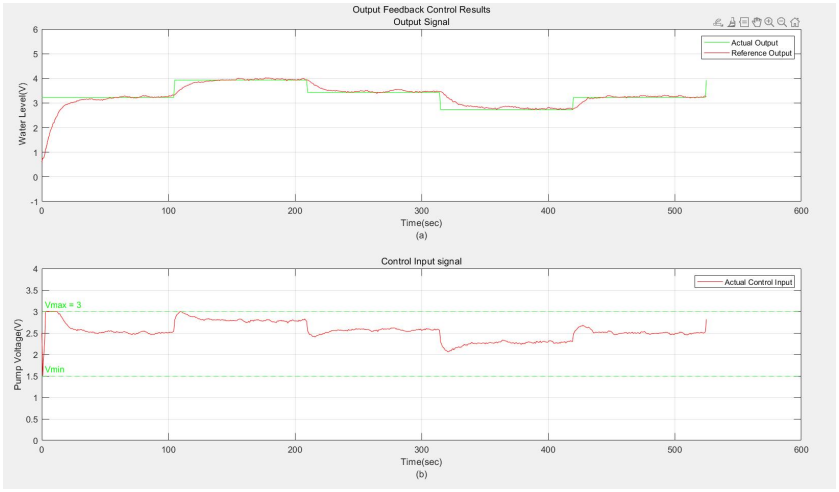


Figure 4. . Comparison between simulated and actual output feedback control

Optional as Bonus: Because a high voltage is initially required to drive the water tank, there will be a clipping effect exceeding the voltage threshold at the beginning of the input stage. If high voltage is undesirable, a certain amount of water can be pre-stored in the tank.

Conclusion

The main objective of this experiment is to implement the non-deadbeat output feedback control system onto the actual water tank system and evaluate its performance. Using the water tank model in the experiment, system identification was first performed to obtain a new system model, and based on this, the output feedback control system was designed.

Lab 5 Report

Pre-lab Exercise

Question 1.

ZOH Transfer function:

$$G_h(s) = \frac{1-e^{-hs}}{s}$$

$$G_{DT}(z) = Z\left[\mathcal{L}^{-1}\left(G_{CT}(s) \times G_h(s)\right)\right]$$
$$= Z\left[\mathcal{L}^{-1}\left(\left(K_p + \frac{K_i}{s}\right) \times \frac{1-e^{hs}}{s}\right)\right]$$
$$= Z\left[\mathcal{L}^{-1}\left(\frac{K_p(1-e^{-hs})}{s} + \frac{K_i(1-e^{hs})}{s^2}\right)\right]$$

According to Laplace Inverse Transform.

$$\Rightarrow G_{DT}(z) = K_p \frac{z}{z-1}(1-z^{-1}) + K_i \frac{hz}{(z-1)^2} \times (1-z^{-1})$$
$$= K_p \frac{z}{z-1}\left(\frac{1-z}{z}\right) + K_i \frac{hz}{(z-1)^2}\left(\frac{1-z}{z}\right)$$
$$= K_p + K_i \frac{h}{z-1}$$

Question 2.

According Eq 1,

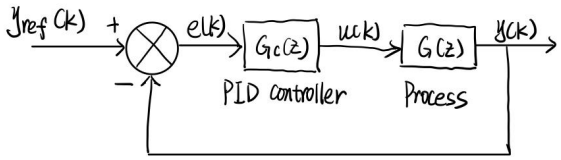
$$G_{DT}(z) = \frac{U_z}{E_z} = K_p + \frac{K_i h}{z-1}$$
$$= K_p + K_i h \cdot \frac{z}{z-1} \cdot \frac{1}{z}$$
$$\downarrow \quad \downarrow$$
$$1 \quad \delta(t-kt)$$

$$\Rightarrow \frac{U(k)}{e(k)} = K_p + K_i h\, \delta(t-kt)$$

$$\Rightarrow U(k) = (K_p + K_i h\, \delta(t-kt))e(k)$$

Question 3.

The block diagram



$$y(k) = e(k) \times G_c(z) \times G(z)$$

$$= [y_{ref}(k) - y(k)] \times G_c(z) \times G(z)$$

$$= y_{ref}(k) \times G_c(z) \times G(z) - y(k) \times G_c(z) \times G(z)$$

$$y_{ref}(k) \times G_c(z) \times G(z) = y(k)[1 + G_c(z) \times G(z)]$$

$$\Rightarrow \frac{y(k)}{y_{ref}(k)} = \frac{G_c(z)\,G(z)}{1 + G_c(z)\,G(z)}$$

$$u(k) = e(k) \times G_c(z)$$

$$= [y_{ref}(k) - y(k)] \times G_c(z)$$

$$= [y_{ref}(k) - u(k)\,G_c(z)] \times G_c(z)$$

$$[1 + G(z)\,G_c(z)]\,u(k) = G_c(z)\,y_{ref}(k)$$

$$\Rightarrow \frac{u(k)}{y_{ref}(k)} = \frac{G_c(z)}{1 + G(z)\,G_c(z)}$$

Lab Exercise

1. System Identification

Repeat the steps of Lab 2 and obtain the following parameters.

```
V_min = 2.2;
V_max = 2.7;
u_offset = (V_max + V_min)/2;
t_cutoff = 138/0.75;
```

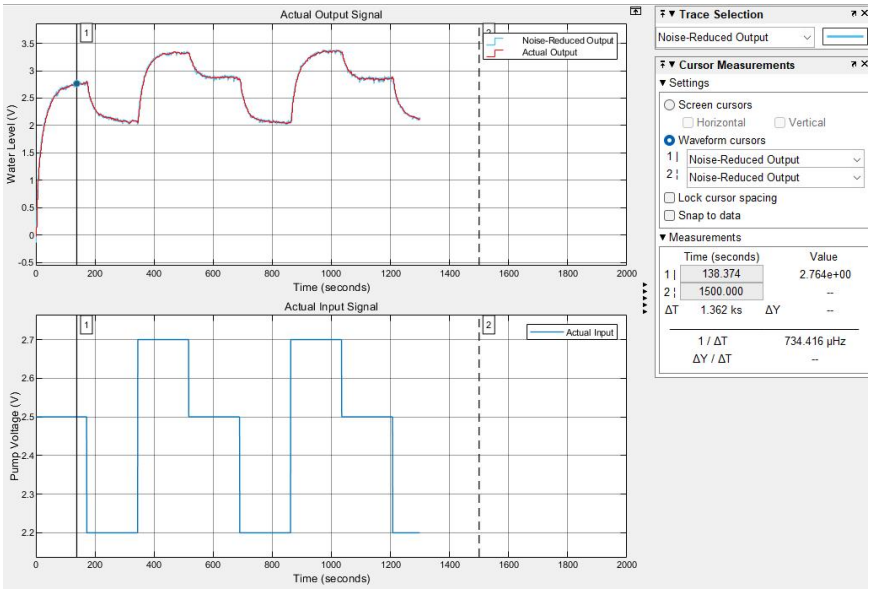
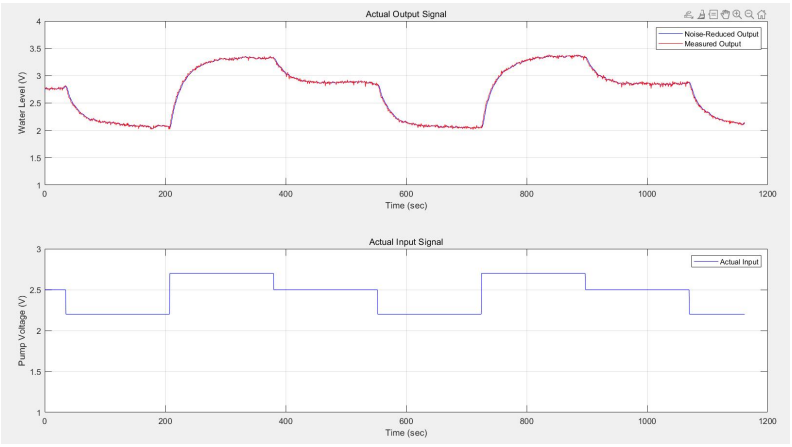
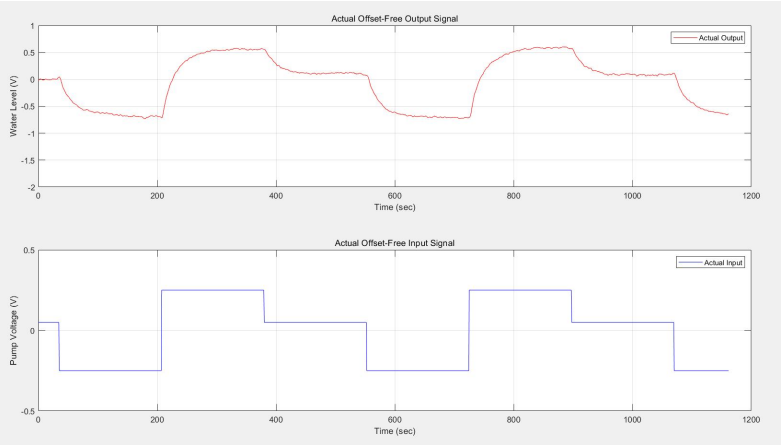


Figure 1. Collected Data

It can be observed from the generated plots that the simulated output tracks the actual output very well.





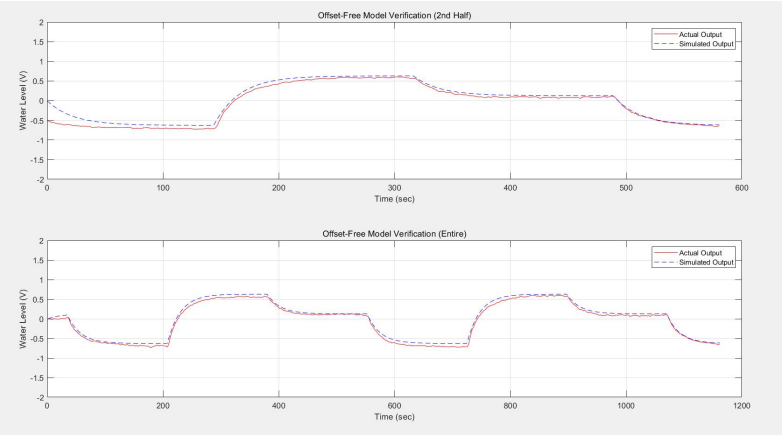


Figure 2. Results repeat Lab 2

2. PI control design

Here is the simulink model of PI control.

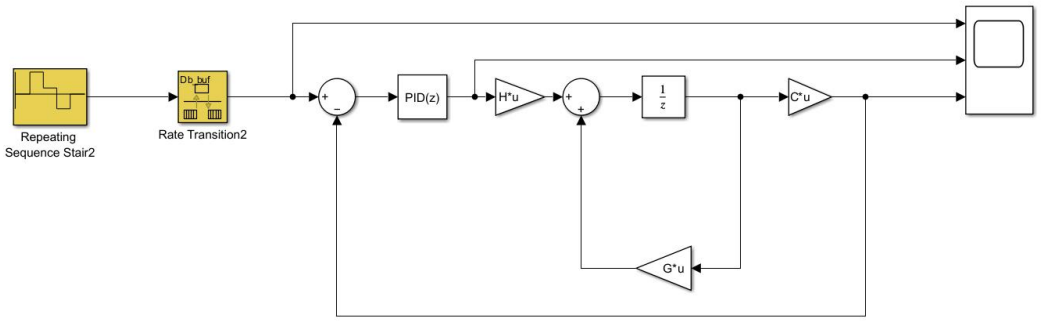


Figure 3. PI control model

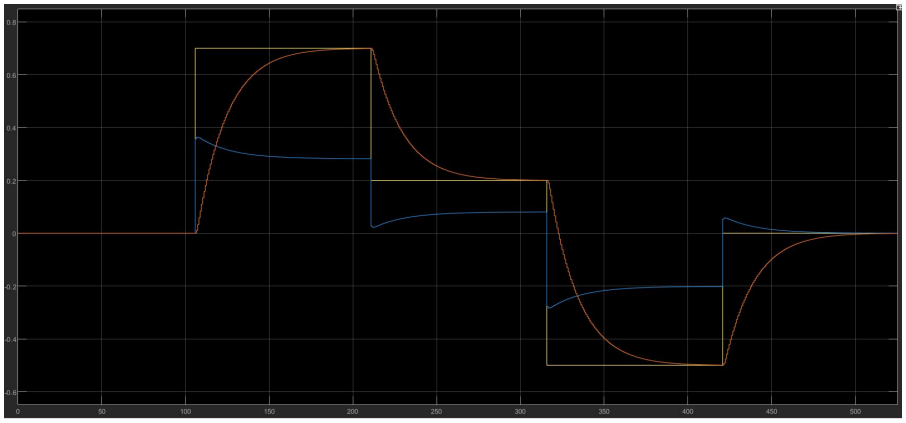


Figure 4. Input and Output of PI Controller

In our design, we used the trial-and-error approach to determine the values of Kp and Ki. After several adjustments, we obtained Kp = 0.51 and Ki = 0.022, which made the simulated output closely follow the reference signal.

3. Real-time implementation of the control system

After applying the designed control parameters and PI controller settings to the model and running the simulation, Figure 5 can be obtained. It can be observed that there is a significant deviation between the actual and simulated outputs in the initial stages. However, around 144s, the actual output begins to track the simulated output, proving that the PI controller can effectively control the system.

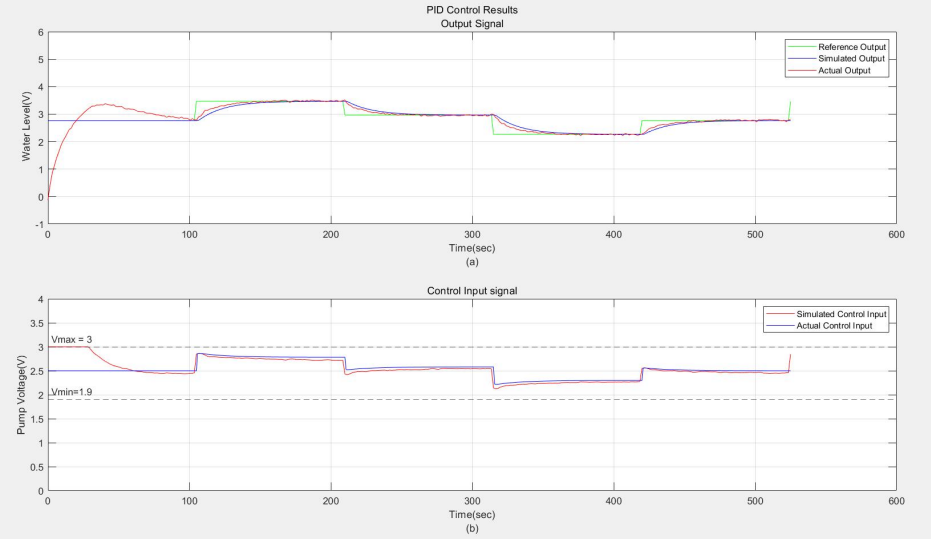


Figure 5. The control result of W-T system

Optional as Bonus:

At the beginning, there may be a significant deviation between the system's output signal and the reference signal, which is typically referred to as the initial transient behavior. This could be because, at the start of control, the system's output has not yet fully responded to the changes in the control input.

The control input saturation at the beginning occurs because the system needs to make a large adjustment in the early stage, and the controller's output signal exceeds the controller's output limit.

Conclusion

In this experiment, the PI controller was successfully designed using the trial-and-error method, with the gains Kp = 0.51 and Ki = 0.022. The system's output closely tracked the reference signal, and the overshoot remained below 2%, indicating effective control. Initially, the control input was saturated due to large adjustments required, but the system stabilized as the output began to align with the reference. Overall, the PI controller demonstrated good performance, proving its ability to manage the system effectively.