
UNSW
AUSTRALIA

**SCHOOL OF ELECTRICAL ENGINEERING AND TELECOMMUNICATIONS**

# ELEC4632

# Laboratory Report

By Yuanzhao Chen (z5041686)

# Table of contents



# Introduction

As the ELEC4632 laboratory guide specified, the task in this report aims at designing a water-tank control system from scratch to control the water level. The water-tank system consists of two water tanks (Q1, Q2) and they are connected by a single valve (valve#12). Inflows of the tanks are fed by two individual pumps which can be controlled by the computer. Water inside the tanks can flow back to sink via two valve controlled tubes. In this task, we control only one water tank's water level (Q1) instead of two. Additionally, valves' positions are fixed after calibration.

In order to achieve this goal, the whole task is split into 5 sub-labs for us to accomplish step by step. In lab 1, we explore the method of identifying a suitable dynamic model (second-order regression model) for a given data set. After system identification process is done, we verify the obtained dynamic model to inspect its accuracy. In the post lab session, we also compare the effect of different model orders in the accuracy of system identification. In lab 2, we basically perform the same technique as we use in lab 1, except that this time we're identifying real-time data's model from the water system. In lab 3, we explore the method of designing state feedback control with observer for both regulation and set-point control. Disturbance rejection and robustness analysis of the system will be discussed in the post lab session. In lab4, we implement the technique in lab 3 on the water-tanks system in real-time to control the water level. In lab 5, we design and implement a PID control system to control the water tank.

# Chapter 1: Use linear least squares method to conduct system identification

**Lab description:** We explore the method of identifying a second-order regression model for a given data set (SysIdenData_StudentVersion.mat). After system identification process is done, we verify the obtained dynamic model to inspect its accuracy. In the post lab session, we also compare the effect of different model orders in the accuracy of system identification to see which order better fits the control system.

## Lab 1 methodology

### Least Square method

We assume the water tank system can be approximated by a second-order transfer function. In general, transfer function in Z domain of a second-order discrete time model is as below:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b_1 z + b_2}{z^2 + a_1 z + a_2} = \frac{b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

The state space form is:

$$\begin{cases} x(k+1) = Gx(k) + Hu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

$$G = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix}, \quad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = [b_2 \;\; b_1], \quad D = 0$$

Differential equation:

$$y(k) + a_1 y(k-1) + a_2 y(k-2) = b_1 u(k-1) + b_2 u(k-2)$$

$$\text{so, } y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_1 u(k-1) + b_2 u(k-2)$$

$$y(k) = \underbrace{\begin{bmatrix} y(k-1) & y(k-2) & u(k-1) & u(k-2) \end{bmatrix}}_{\varphi^T(k)} \underbrace{\begin{bmatrix} -a_1 \\ -a_2 \\ b_1 \\ b_2 \end{bmatrix}}_{\theta} \quad \Rightarrow \quad y(k) = \varphi^T(k)\theta.$$

From above we can see in order to figure out the second-order discrete time model we need to find out the values of parameters $a_1, a_2, b_1, b_2,$ which are called regression variable. We notice that in our model, y(k) depends on two previous data and we can list them out:

$$\begin{cases} y(1) = \begin{bmatrix} y(0) & y(-1) & u(0) & u(-1) \end{bmatrix} \theta \\ y(2) = \begin{bmatrix} y(1) & y(0) & u(1) & u(0) \end{bmatrix} \theta \\ y(3) = \begin{bmatrix} y(2) & y(1) & u(2) & u(1) \end{bmatrix} \theta \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ y(N) = \begin{bmatrix} y(N-1) & y(N-2) & u(N-1) & u(N-2) \end{bmatrix} \theta \end{cases}$$

$$\Rightarrow \underbrace{\begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ \vdots \\ y(N) \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} y(0) & y(-1) & u(0) & u(-1) \\ y(1) & y(0) & u(1) & u(0) \\ y(2) & y(1) & u(2) & u(1) \\ \vdots & \vdots & \vdots & \vdots \\ y(N-1) & y(N-2) & u(N-1) & u(N-2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} -a_1 \\ -a_2 \\ b_1 \\ b_2 \end{bmatrix}}_{\theta}$$

$$\Rightarrow \quad Y = \Phi\theta.$$

Look at the above equation only $\theta$ is un-known to us, however we can use Least Square Method to find out an estimator of $\theta$:

$$\hat{\theta} = (\Phi^T\Phi)^{-1}\Phi^T Y$$

Notice that in this lab we use half of the data y to find out $\hat{\theta}$ (i.e., N = floor(0.5*length(y_act))) and then compare the simulation result with the second half of the actual data to see whether it fits the actual system, although in practice using all of the given data to find out $\hat{\theta}$ will yield a more accurate approximation of $\theta$.

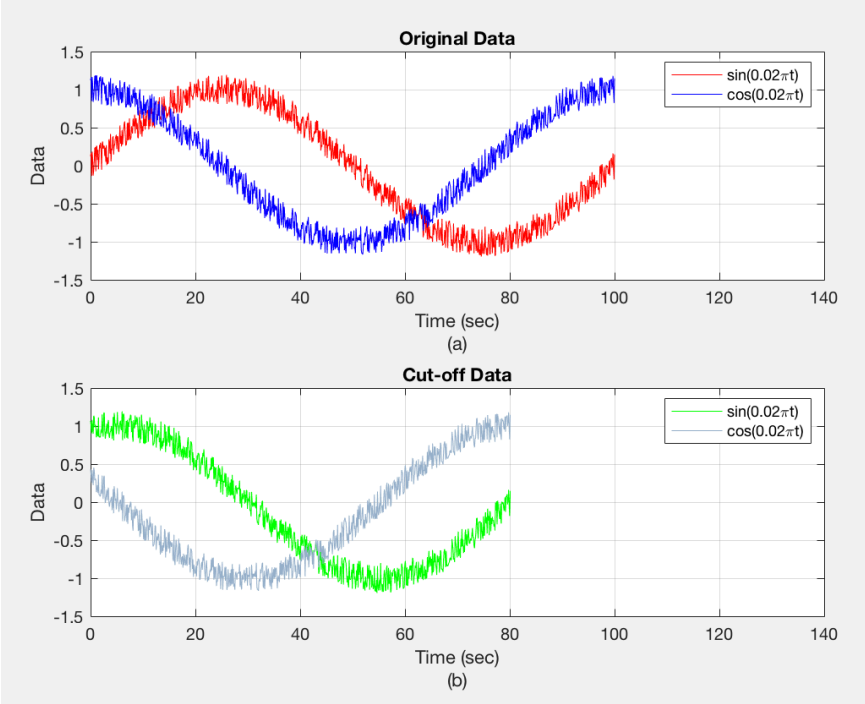
**Offset compensation**

Keep in mind that all variables in the control system theory are offset free (i.e. fluctuate around 0), however in real life this is not the case. Hence before we perform any modern control technique, we must find variables' offset and subtract them from the actual value. Specifically, in this task we need to find out the offset of output signal y_act and the offset of input signal u_act. We name these two offsets y_offset and u_offset. We only apply (y_act – y_offset) and (u_act – u_offset) to our control theory instead of y_act and u_act. After every calculation is done, we need to add those offsets back to the processor.

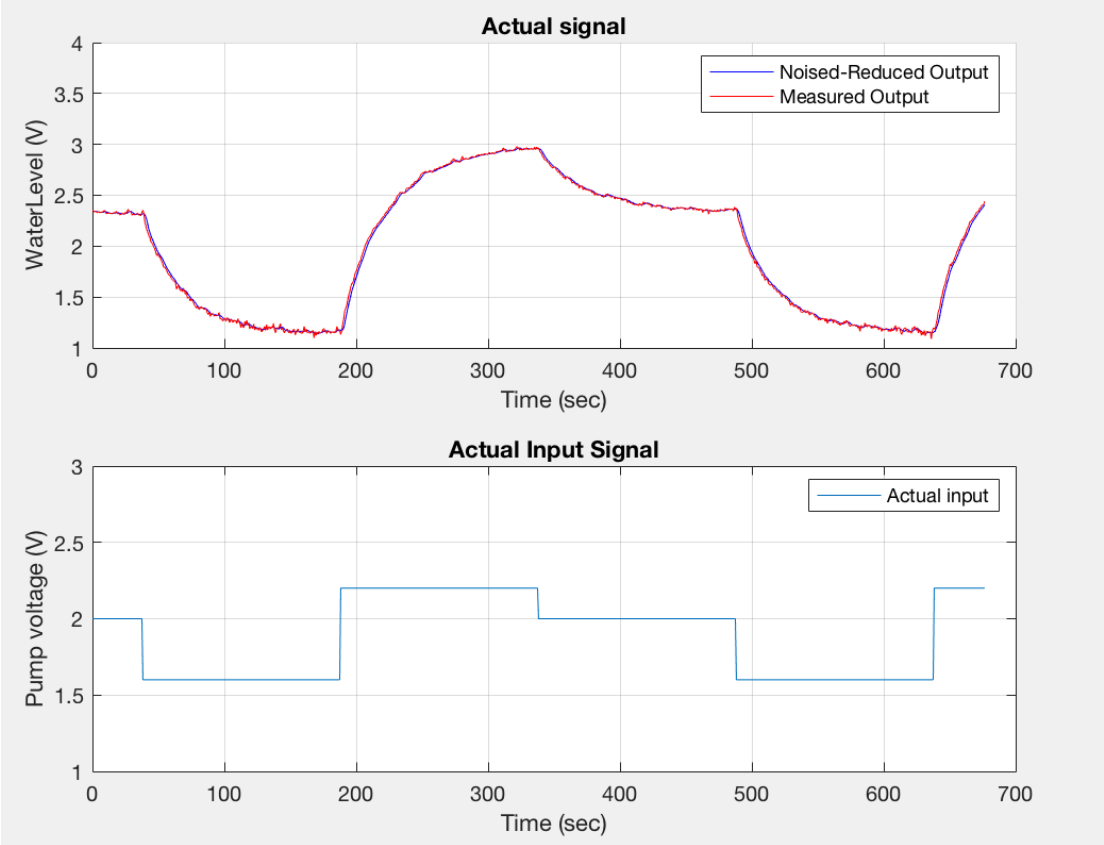
Conventionally, u_offset is the first value of u_act because at t = 0 sec we assume there is no offset-free input and everything is at rest. The best approach to estimate the closest value to actual output offset is by taking average form the first period of output data which corresponds to the first period of input data before the first change.

# Lab 1 result

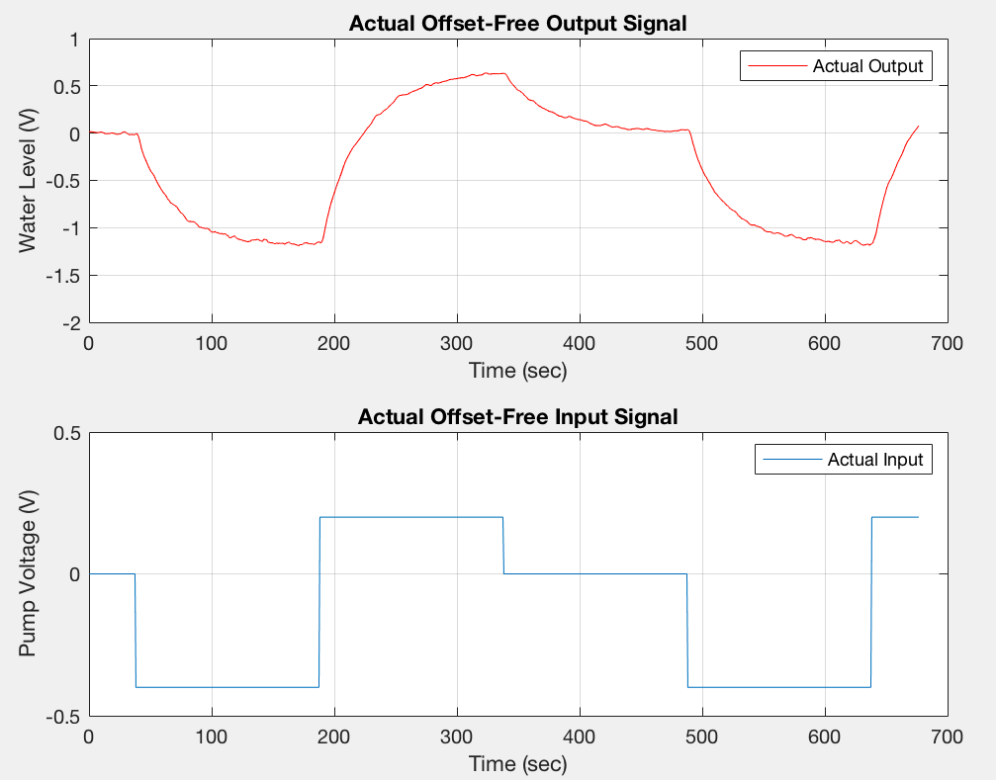
Prelab:



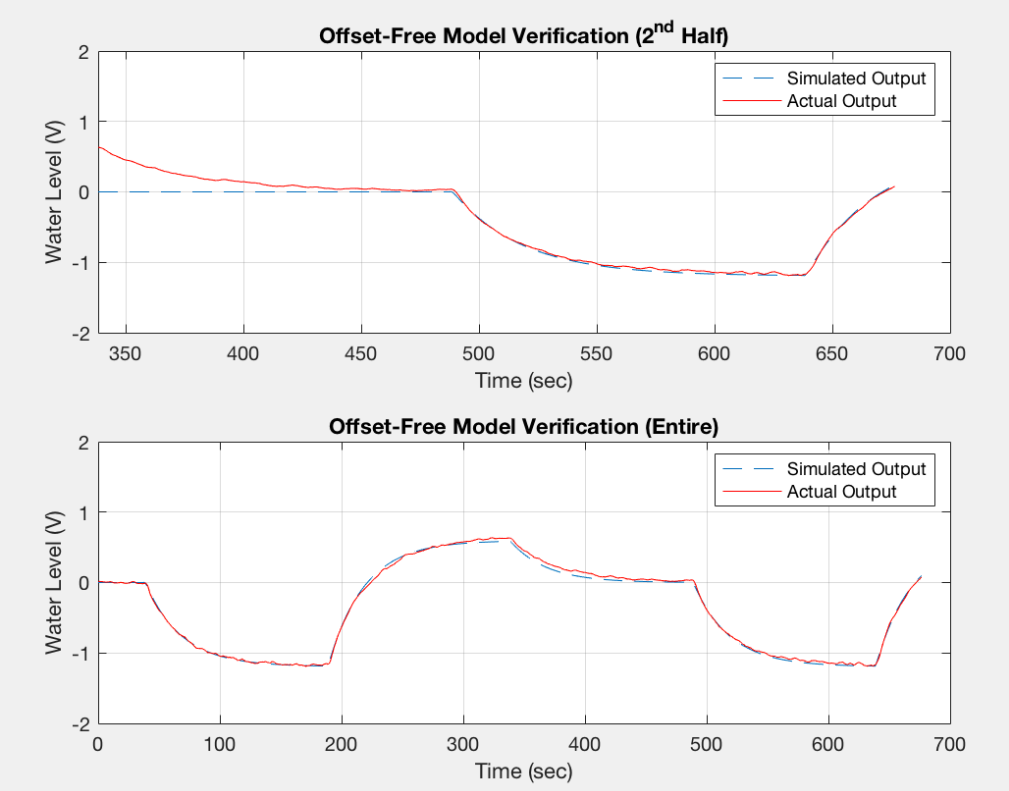

Plotting of non-offset free input, output:



Plotting of offset-free output, input:

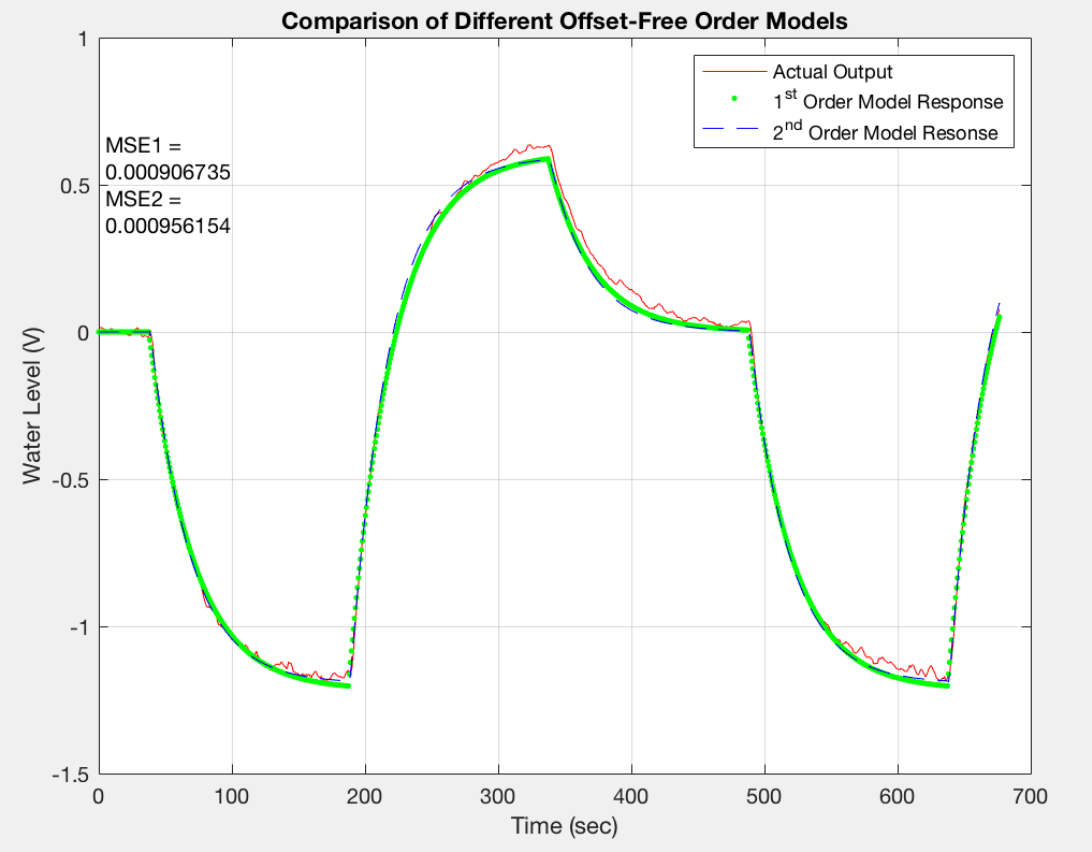



System identification simulation vs actual result:



**Comment:** From figure above we can see that although data of the second half was not used to calculate $\hat{\theta}$, the simulation is still pretty close to the actual output, which is very good. The reason why simulation and actual result differ so much at the beginning is that in the dynamic model we assume the system is initially at rest (i.e., u (0) = 0, y (0) = 0), however, the second half of the actual data is not initially 0 (activated by first half's input).

Post lab:




**Comment:** To calculate first order regression model, the matrix $\theta$ is a bit different here:

$$\begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ . \\ . \\ . \\ y(N) \end{bmatrix} = \begin{bmatrix} y(0) & u(0) \\ y(1) & u(1) \\ y(2) & u(2) \\ . & . \\ . & . \\ . & . \\ y(N-1) & u(N-1) \end{bmatrix} \begin{bmatrix} -a_1 \\ b_1 \end{bmatrix}$$

For first order regression model, y only depends on 1 previous data instead of 2. Except that, every procedure is the same as calculating second-order model. We find out in this case first order model is better than second order model as it has a smaller Mean Square Error (MSE), which is not very usual.

# Chapter 2: Identify a linear model for a water-tank system

**Lab description:** In lab 2, we basically perform the same technique as we did in lab 1, except that this time we're identifying real-time data collected from the water system. The input of the water tank system is the pump voltage of Q1 and output is voltage which is associated with the water level. Valves' positions are fixed after calibration.

## Lab 2 methodology

Methodology is the same as lab 1, only difference is the lab procedure.

### Lab procedure

1. Run the Simulink model (WaterTankSysIden.slx)
2. Find linear voltage range: Set gain block for step input to 2.5 V. Adjust valve position such that to water level goes up to somewhere between 240 ml and 260 ml. After water level remains stable for one minute, reduce gain block so that water level comes down to somewhere between 60 ml and 80 ml, water level should stay steady for one minute as well. We kept a record of the data in this step:

| | |
|---|---|
| Vmin = 1.5 V | Water Level min = 75 ml (1.3V) |
| Vmax = 2.5V (fixed) | Water Level max = 245 ml (4.22V) |

3. Preparation for data collecting:

$$u\_offset = (Vmax + Vmin)/2 = 2\ V$$

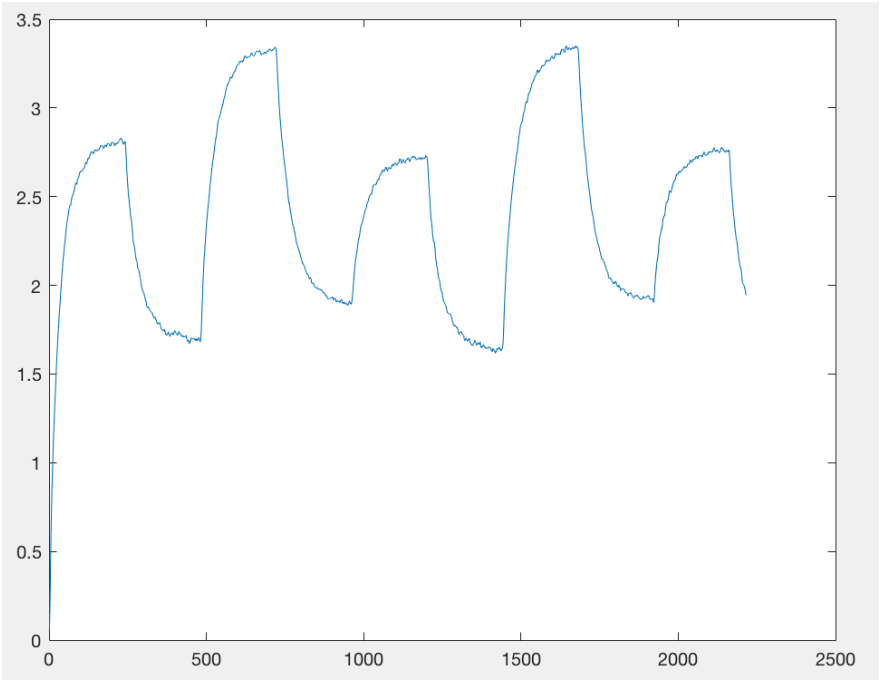
$$Vmin - u\_offset < u(k) < Vmax - u\_offset, \text{ hence}$$

$$-0.5 < u(k) < 0.5$$

   The default value for the signal period is $200 \times 0.75 = 150$ sec, we find out in our system $380 \times 0.75 = 285$ sec is a better value.
4. Real-time input-output data collection
5. Identifying the linear second-order model (modify from lab 1 code)

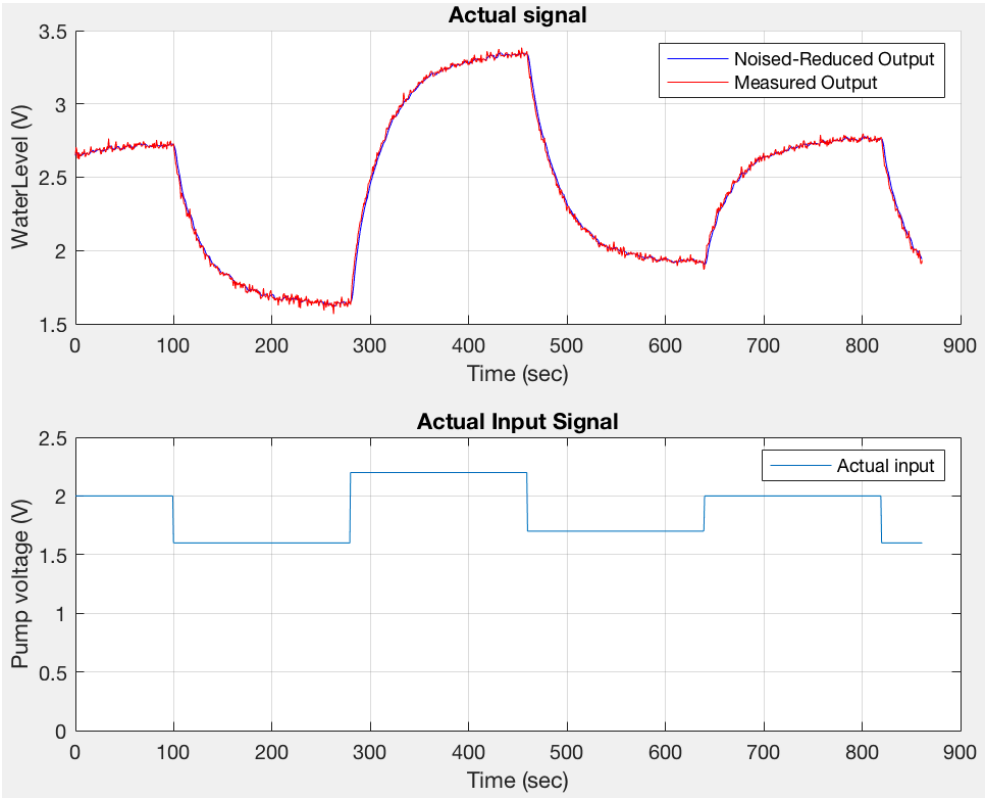
## Lab 2 result

Raw data from Simulink:

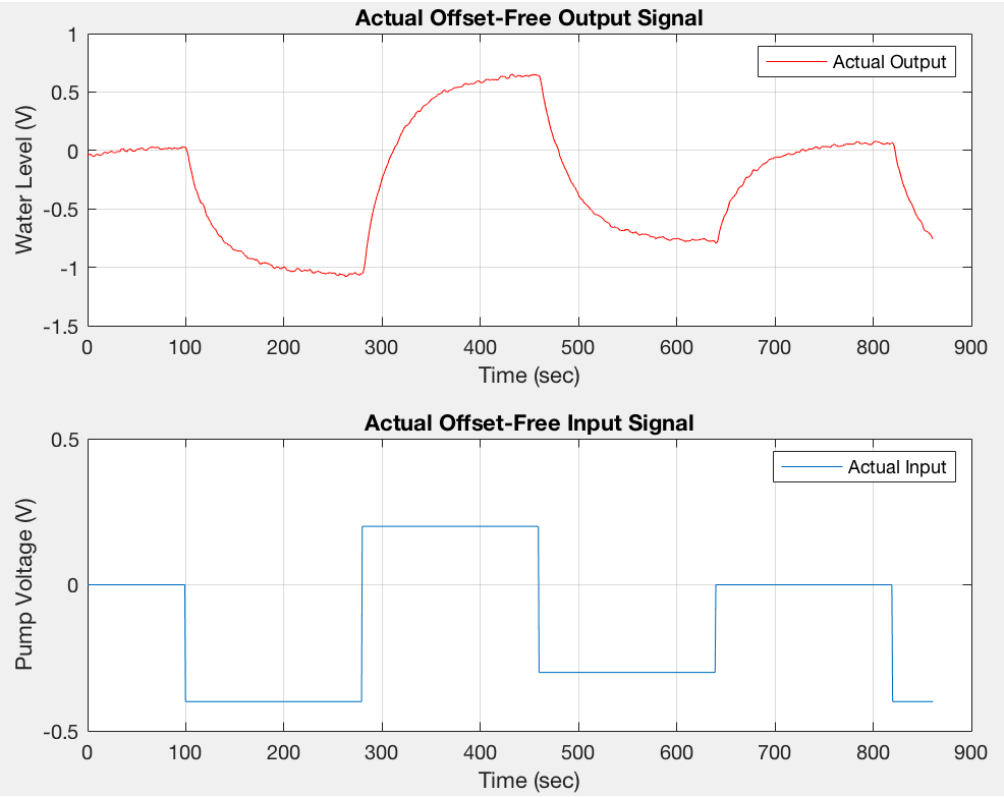



**Comment:** We have collected two periods of data from Simulink, we only need to keep the second period to perform system identification. Consequently, we throw away data before 800.25 sec (where the first period ends).

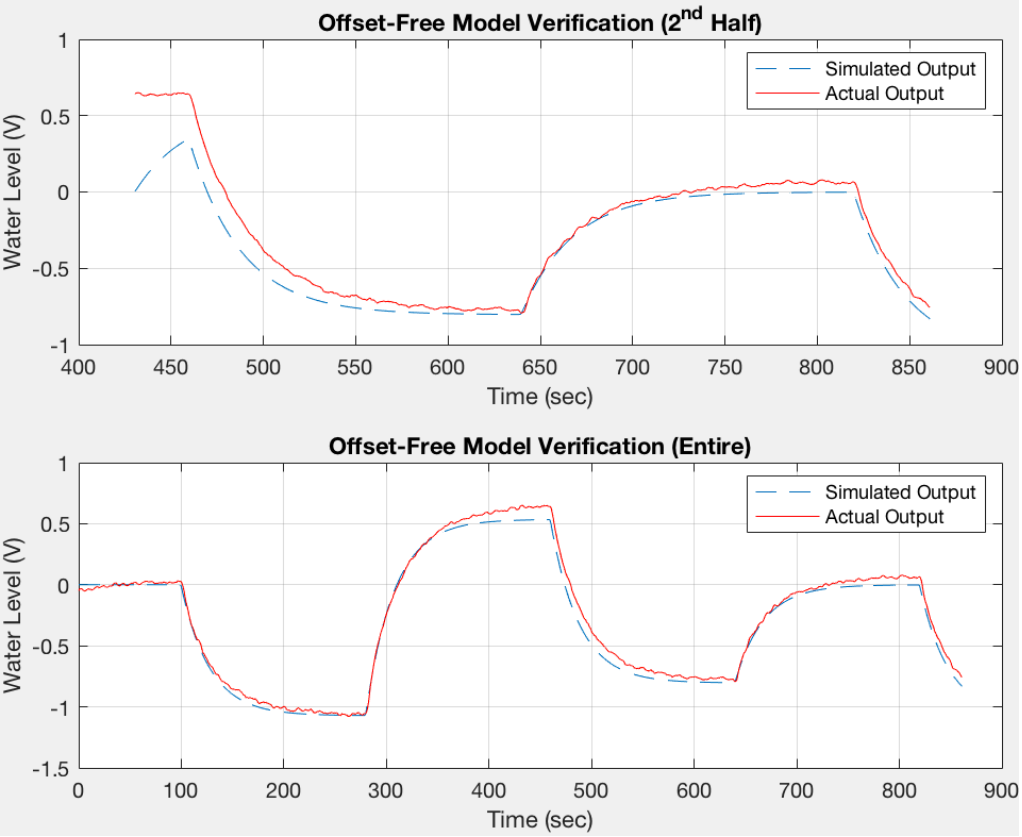
Plotting of non-offset free input, output:



Plotting of offset-free output, input:




System identification simulation vs actual result:



# Chapter 3: Design of state feedback control with observer for both regulation and set-point control

**Lab description:** In lab 3, we explore the method of designing state feedback control (both deadbeat control and non-deadbeat control) with observer for both regulation and set-point control. Disturbance rejection performance and robustness analysis of the designed control system will be discussed in the post-lab session.

## Lab 3 methodology

### Deadbeat control and non-deadbeat control design

Same as lab 1, the state space form of the water-tank system is:

$$\begin{cases} x(k+1) = Gx(k) + Hu(k) \\ y(k) = Cx(k) + Du(k) \end{cases}$$

$$G = \begin{bmatrix} 0 & 1 \\ -a_2 & -a_1 \end{bmatrix}, \text{H} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \text{C} = [\, b_2 \;\; b_1 \,], \quad \text{D} = 0$$

By using system identification technique (refer to lab 1), we find out:

$$G = \begin{bmatrix} 0 & 1 \\ -0.2287 & 1.209 \end{bmatrix}, \qquad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \qquad C = [0.0318 \;\; 0.0206], \qquad D = 0$$

In this lab, we prefer using canonical observable form of the state space model when designing deadbeat control and non-deadbeat control:

$$G_{observe} = G^T = \begin{bmatrix} 0 & -0.2287 \\ 1 & 1.209 \end{bmatrix}, \; H_{observe} = C^T = \begin{bmatrix} 0.0318 \\ 0.0206 \end{bmatrix}$$

$$C_{observe} = H^T = [0 \quad 1], \qquad D_{observe} = 0$$

State feedback of a second order deadbeat control is:

$$L_{db} = [0\ 1]\left[ {G_{observe}}^{-2} H_{observe} \quad {G_{observe}}^{-1} H_{observe} \right]^{-1}$$

$$= [22.7326 \;\; 23.6520]$$

State feedback of a second order non-deadbeat control is:

$$L_{ndb} = [0\ 1]{W_c}^{-1}P(G)$$

Where $W_c = [\ H_{observe}\ \ G_{observe}H_{observe}\ ] = \begin{bmatrix}0.0318 & -0.0047\\ 0.0206 & 0.0567\end{bmatrix}$, $P(G) = G^2 + p1G + p2I$. p1 and p2 depends on a second order polynomial whose roots are eigenvalues 0.2348 and 0.9. $P = (z - 0.2348)(z - 0.9) = z^2 - 1.1348z + 0.2113$. So p1 = -1.1348, p2 = 0.2113. Base on that,

$$P(G) = \begin{bmatrix}-0.0174 & -0.0170\\ 0.0742 & 0.0723\end{bmatrix},$$

Hence,

$$L_{ndb} = [0\ 1]\begin{bmatrix}0.0318 & -0.0047\\ 0.0206 & 0.0567\end{bmatrix}^{-1}\begin{bmatrix}-0.0174 & -0.0170\\ 0.0742 & 0.0723\end{bmatrix}$$

$$= [1.4314 \quad 1.3946]$$

For both deadbeat control and non-deadbeat control, u(k) = -L x(k). We can find out feedback controls' corresponding outputs with the following Matlab command:

```
sys_ndb = ss((G_obsrv - H_obsrv*L_ndb),[],C_obsrv,D_obsrv,Ts);
sys_db = ss((G_obsrv - H_obsrv*L_db),[],C_obsrv,D_obsrv,Ts);
[y_ndb,t_ndb,x_ndb]= initial(sys_ndb,[x1 x2],Tf);
[y_db,t_db,x_db] = initial(sys_db,[x1 x2],Tf);
```

We find out deadbeat control is not suitable for the design because its deadbeat control input (offset-free) exceeds [-0.5 0.5]. (Results are shown in the next session.)

**Set-point control design**

When set point is introduced in the design,

$$u(k) = -L_{ndb}x(k) + {G_{cl}}^{-1}(1)y_{ref}(k)$$

$G_{cl}$ is the inverse of the DC gain of the closed-loop transfer function. $y_{ref}$ is the reference or desired output, here we let $y_{ref}$ = {0 0.7 -0.2 0.5 0}, with period of each level to be 140×0.75 = 105 sec. The way to calculate $L_{ndb}$ is the same as what we have discussed before, the only difference is that now eigenvalues become [0.9 0.9]. In addition, we should double check that

$$80\ \text{ml} < (\ y_{ref}(k) + \text{y\_offset}\ ) \times 300/5 < 240\ \text{ml}$$

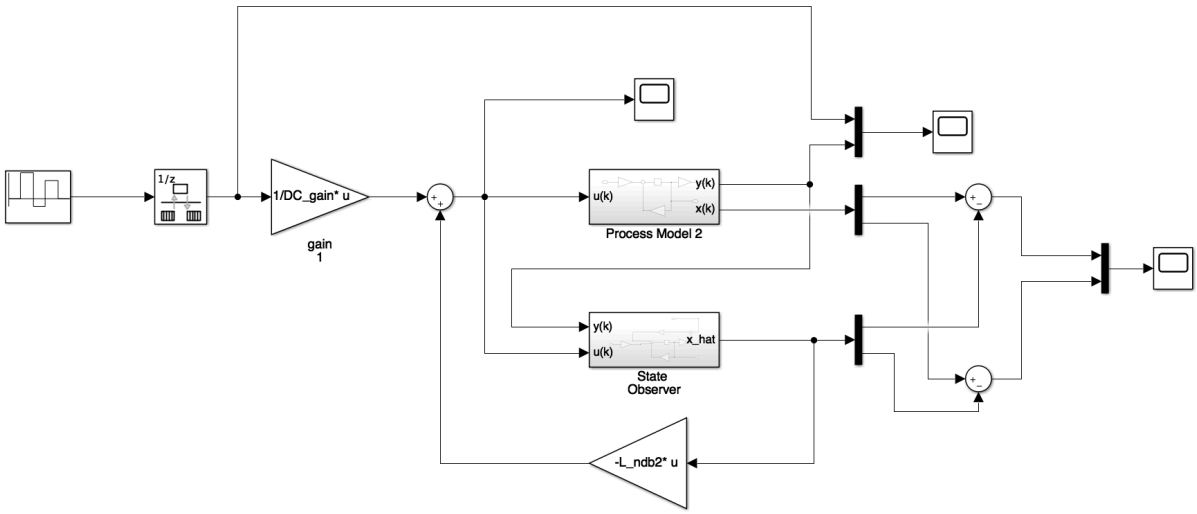
**Set-point control with observer**

When only output can be measured from the system, we need to construct a state observer to find out what the current state is.

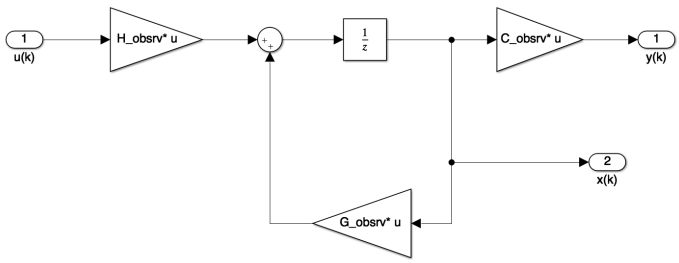
In order to design a deadbeat observer, we should find out gains K:

$$K = PW_o^{-1}[0\ 1]^T = [\ -0.2287 \quad 1.2090]$$

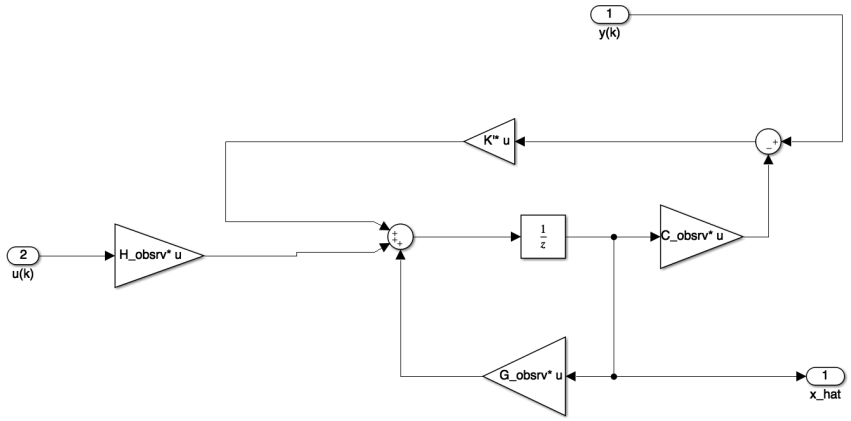
We built this observer in Simulink:




Process Model 2:



State observer:

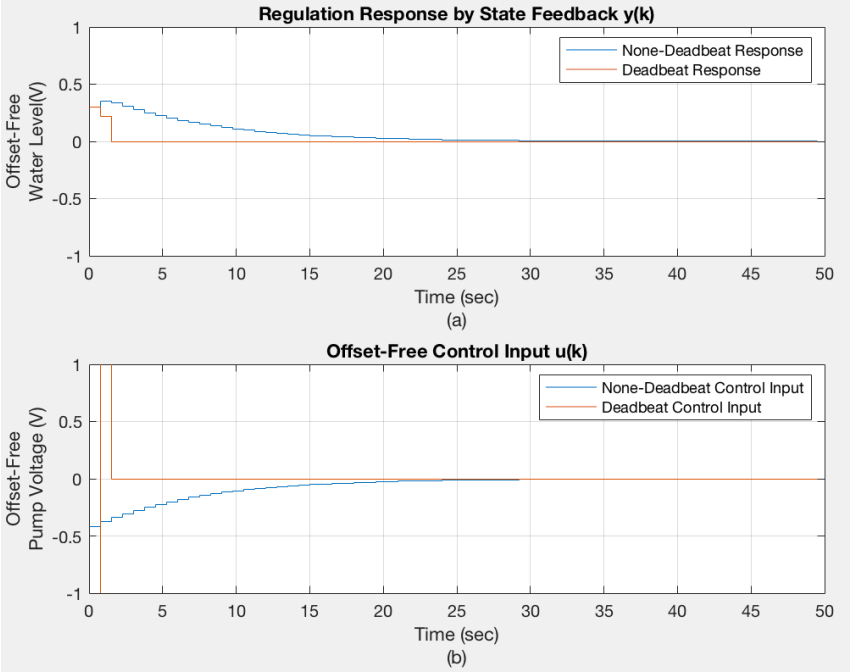



## Lab 3 result

Prelab:

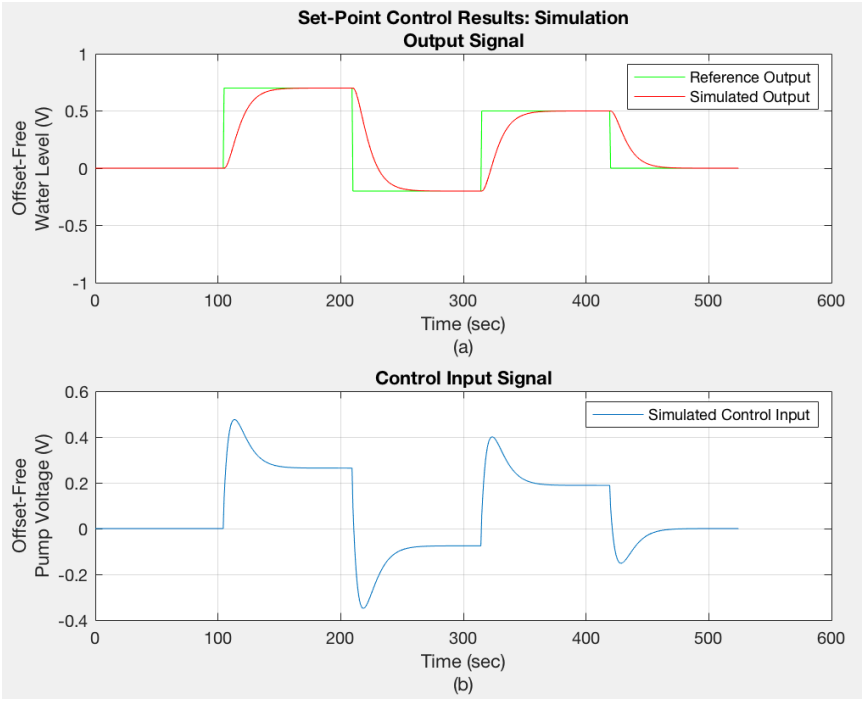
```
sampling time according to calculation is 7.500000e-01
Info about second order state space model is below:

sys =

  A =
           x1       x2
   x1    1.209  -0.4575
   x2      0.5        0

  B =
        u1
   x1  0.25
   x2     0

  C =
           x1       x2
   y1  0.08232   0.2542

  D =
       u1
   y1   0

Sample time: 0.75 seconds
Discrete-time state-space model.

Eigenvalues are 0.974188 and 0.234788
Wc has full rank, reachable.
Wo has full rank, observable, obeservability implies detectability
```

Deadbeat control vs Non-deadbeat control:



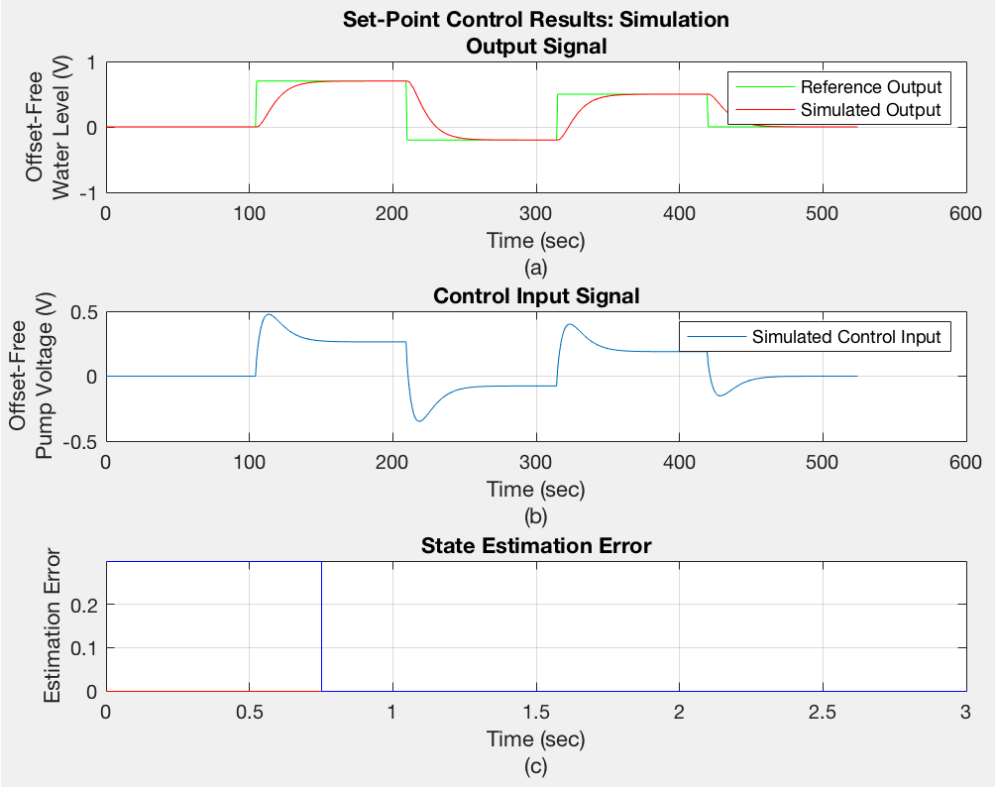

**Comment:** From diagram above we can see deadbeat control has a faster response but its corresponding offset-free input voltage would dramatically exceed the valid range [-0.5 0.5], so we have to choose non-dead control system in the rest of the lab.

Set-point control response:



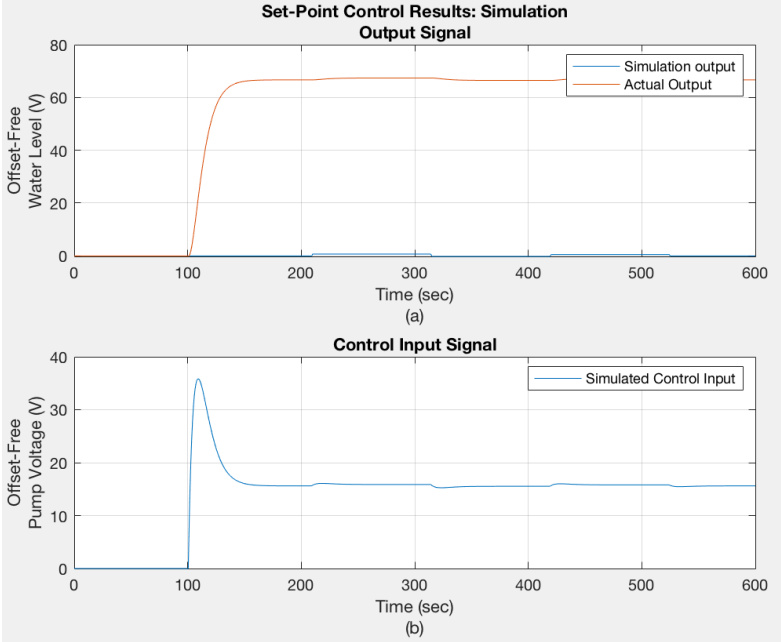

**Comment:** We can observe the simulated output is tracking the reference output and they would be equal at some point. The corresponding control input stays within [-0.5 0.5], so the design is feasible.

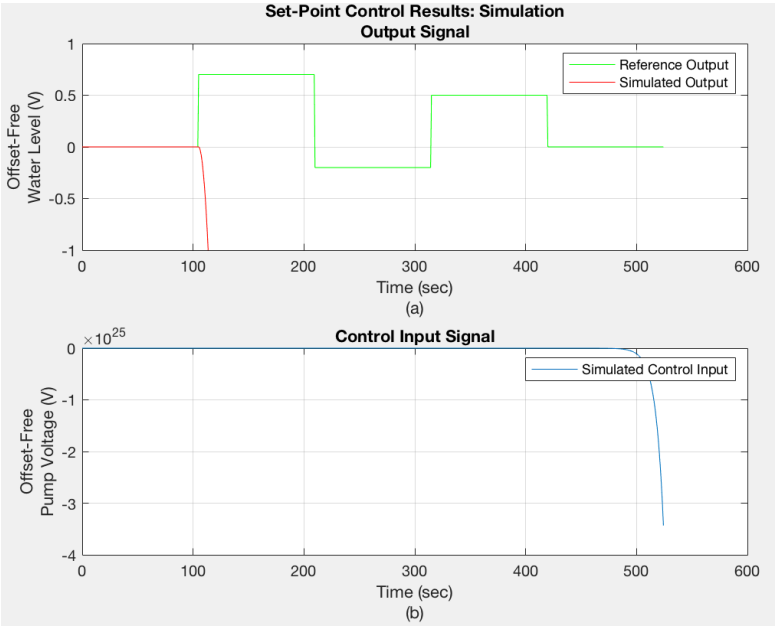
Set-point control response with observer and non-zero initial conditions:




**Comment:** As before, we observe that simulated output is tracking the reference output and they at some point they will be equal. Meanwhile, the offset-free pump voltage stays with the valid range all the time so the design is feasible. We observe that although the observer and the actual system have different initial condition, after one sampling time (0.75 sec) they would be equal. We deduce that the state observer is doing a good job.

Post lab (disturbance rejection and robustness analysis):




**Comment:** It turns out that the system can't deal with large disturbance and it is not robust (when poles' positions are close to unit circle, a little fluctuation may lead to uncontrollable behaviour).

# Chapter 4: Real-time implementation of output feedback control system for set-point control

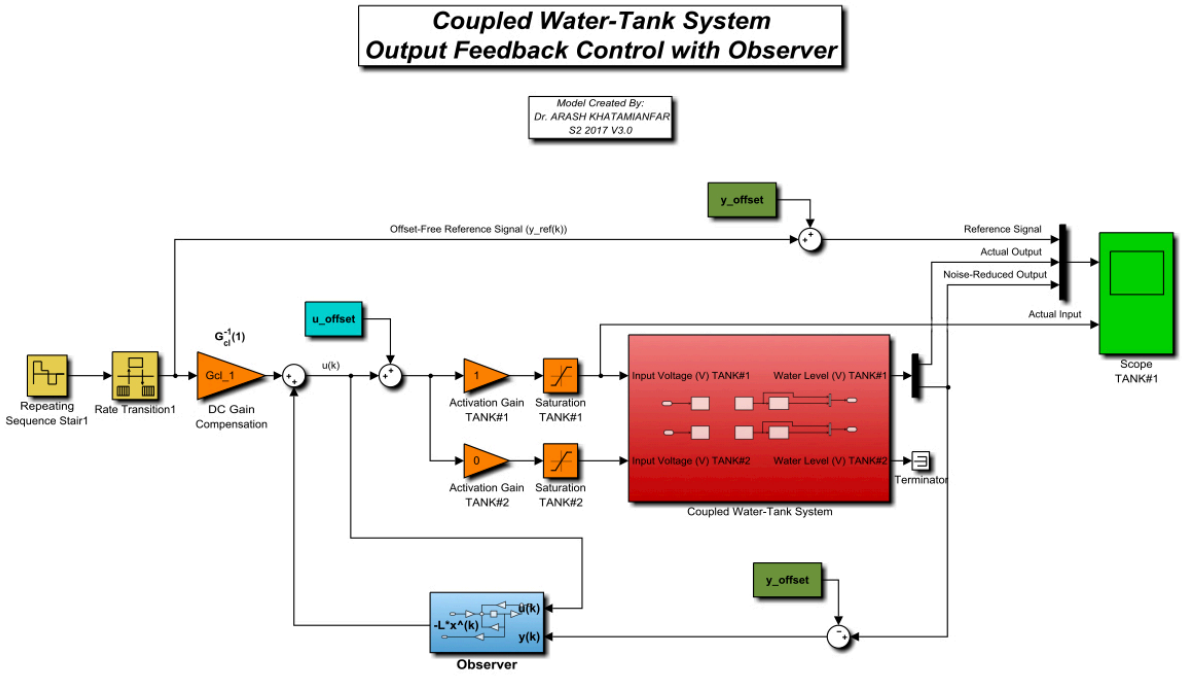
**Lab description:** In lab4, we implement the technique developed in lab 3 on the water-tanks system to control the water level.

## Lab 4 methodology

Methodology in this lab is the same as lab4, except that this time we're implementing our previous work on the water tank system.

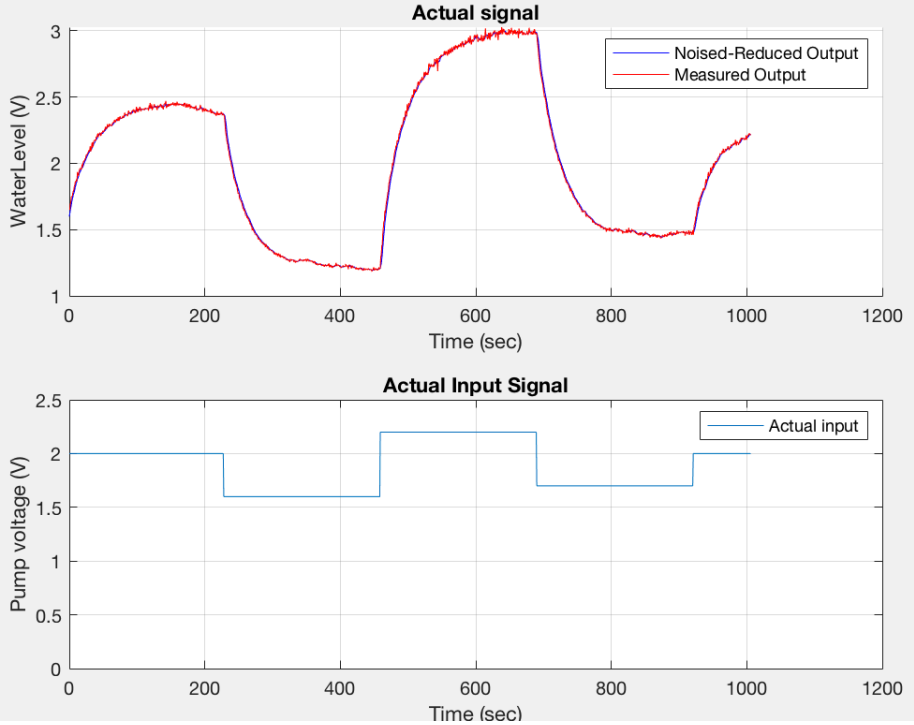
### Lab procedure

1. System identification
2. Output feedback control design (both deadbeat and non-deadbeat)
3. Real-time implementation of the control system: Put all the variables we calculated before into the given Simulink:

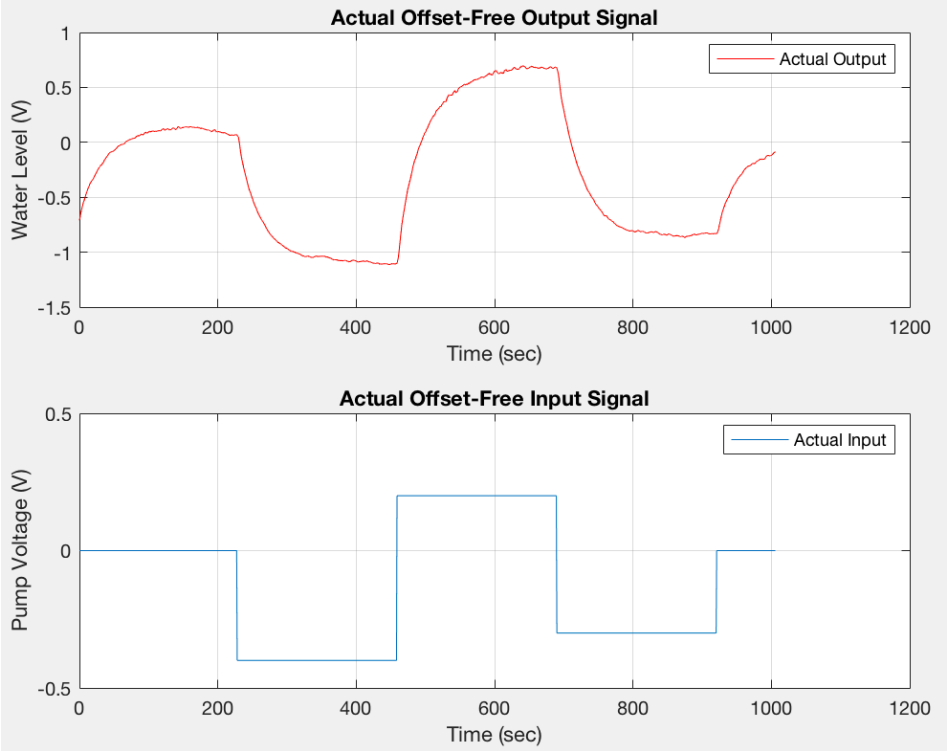


# Lab 4 result

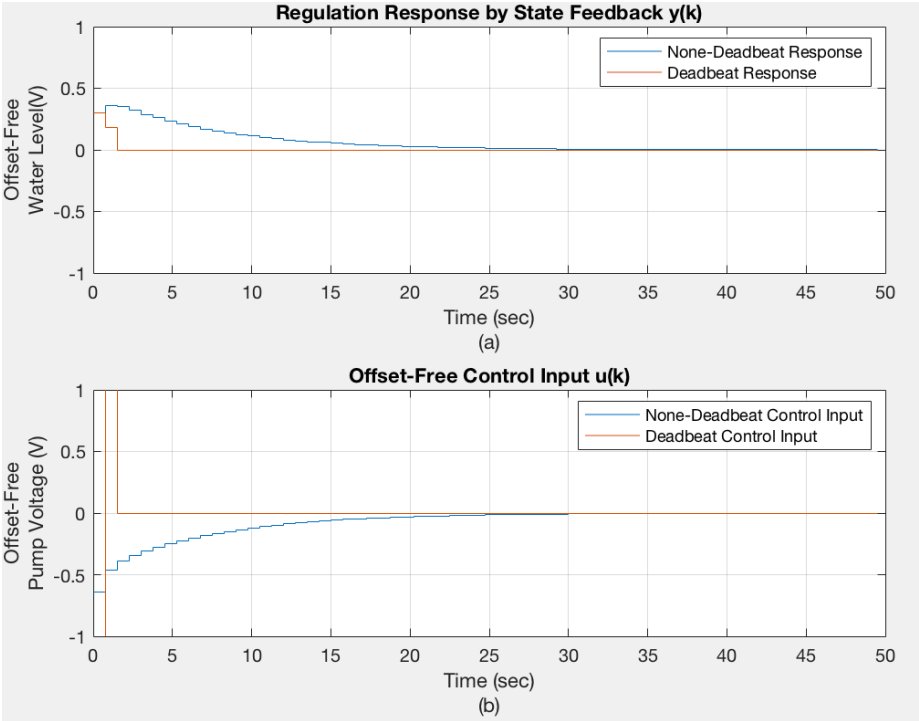
Non-offset free output, input:




Offset-free output, input:

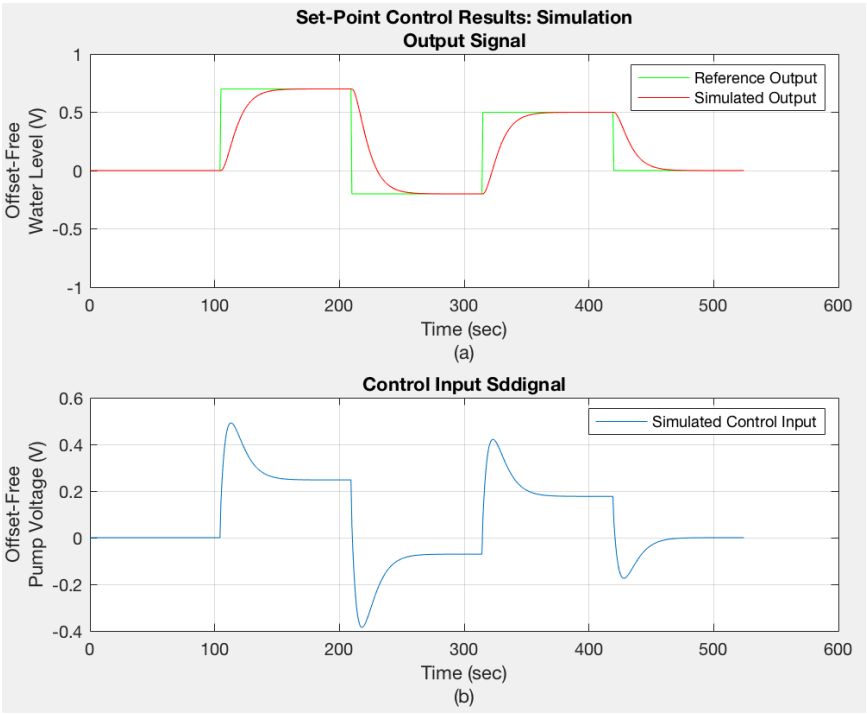


Deadbeat control vs Non-deadbeat control:

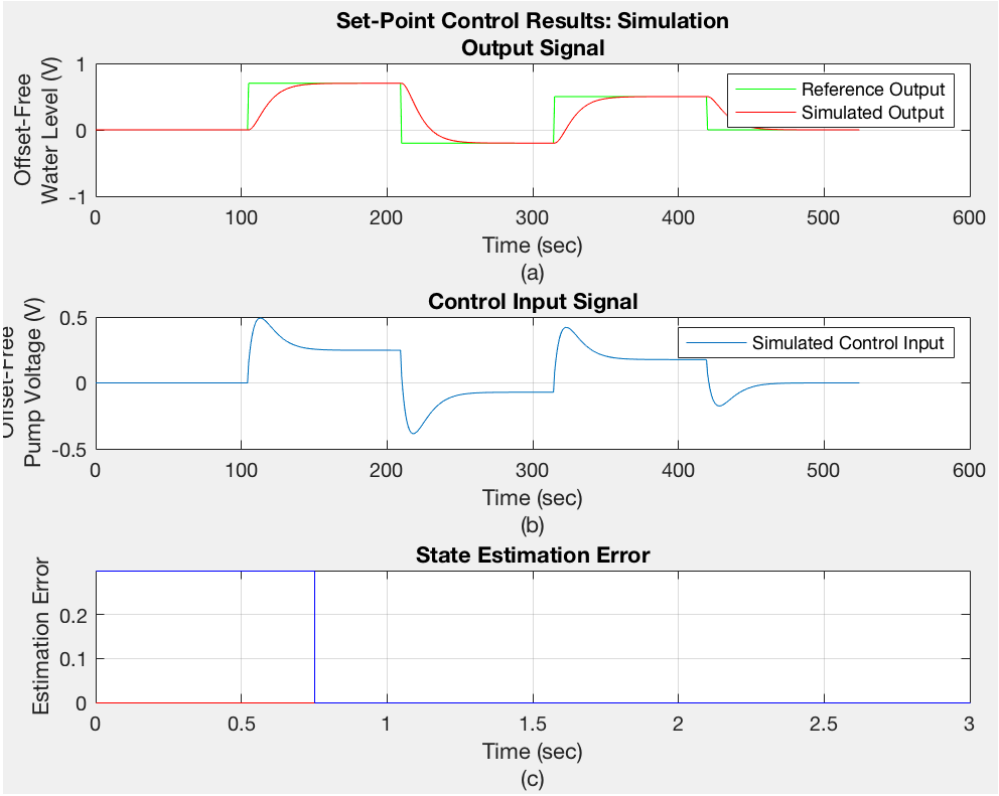



**Comment:** The same as lab 3, although deadbeat control has a faster response, its corresponding input voltage exceeds [-0.5 0.5] so we have to abandon deadbeat control in the rest of our design.

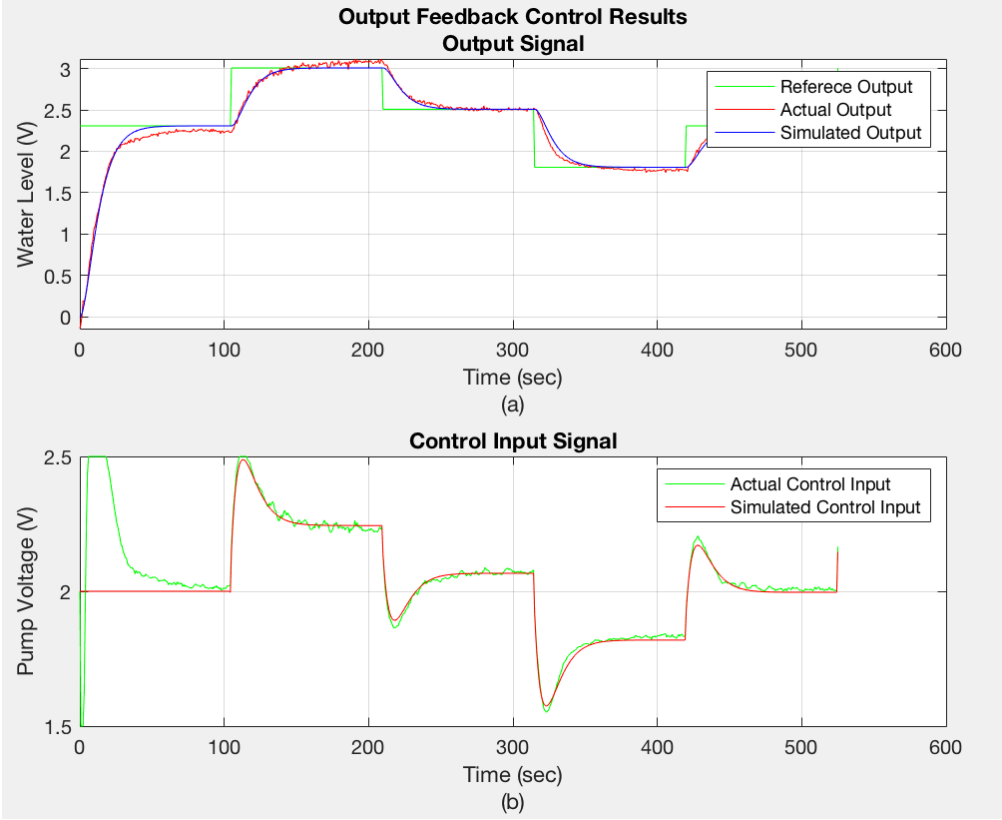
Set-point control response:



Set-point control response with observer and non-zero initial conditions:




Comparison between simulated and actual output feedback control of W-T system in set-point tracking:



**Comment:** From figure above we can see that output feedback control is tracking set-point very well. The reason why actual control input gets saturated to 2.5V (maximum input voltage) at the beginning is that initially the system is not offset-free (actual output starts at 0). After output raises above the offset value the system would operate normally and the corresponding control input signal would stay with [-0.5 0.5].

# Chapter 5: Design and Real-time implementation of PI control system for set-point control

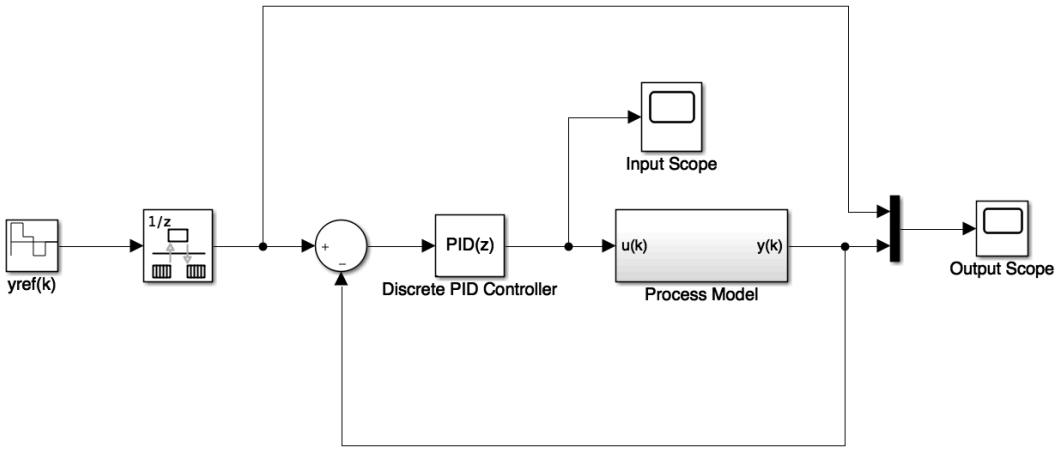
**Lab description:** In this lab, we would design, simulate, and implement a discrete-time proportional-integral (PI) control system on an actual W-T system in real-time. The PI controller is from the proportional-integral-derivative (PID) controller category without its derivative part. System identification is still a pre-requisite of our design, although in real world, having a valid model for PID control system is not essential.

## Lab 5 methodology

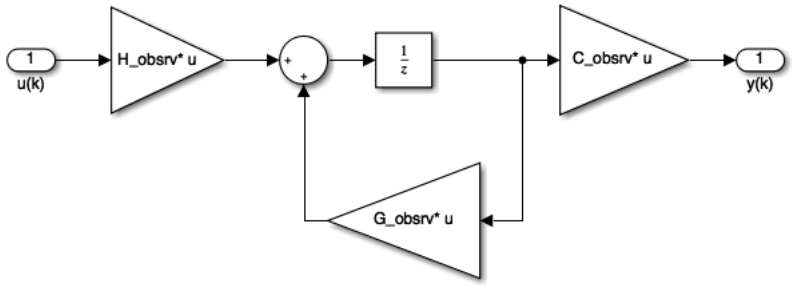
The discrete-time transfer function of PI control is:

$$G_{DT}(Z) = \frac{U(z)}{E(z)} = K_p + \frac{K_i h}{z-1}$$

We introduce PI controller in the block diagram, so Simulink model will become:

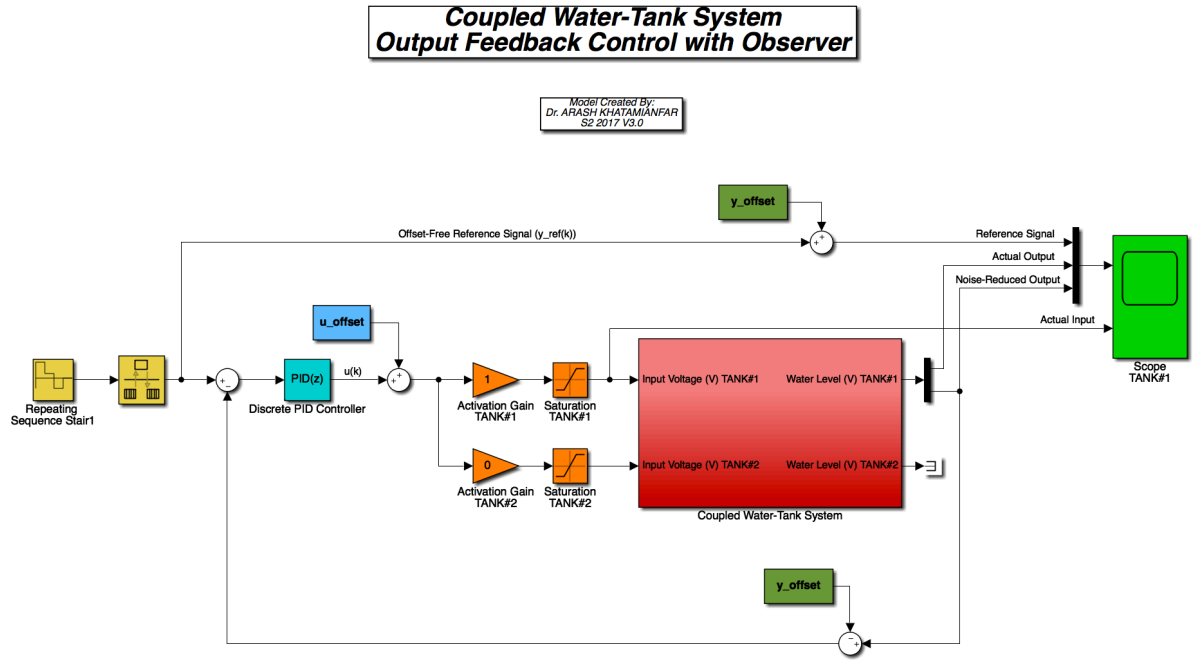



Process Model:



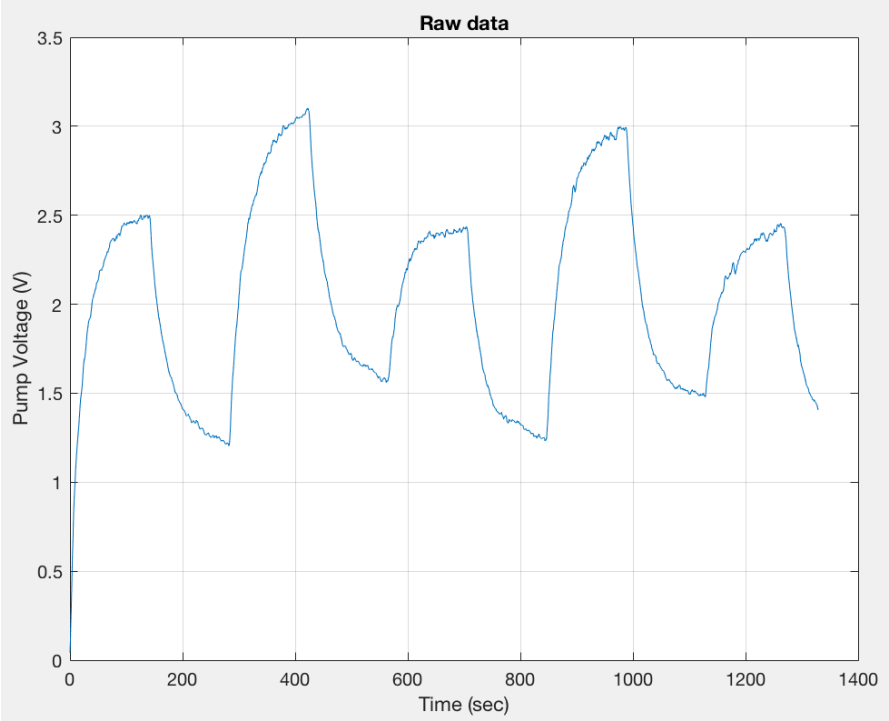
Process model (observer form) is obtained during the system identification process (same process as lab 2).

After system identification is done and PI controller's parameters (i.e., Kp, Ki) are finely tuned, we implement our design on the pre-built Simulink to control water tank's water level in real time:

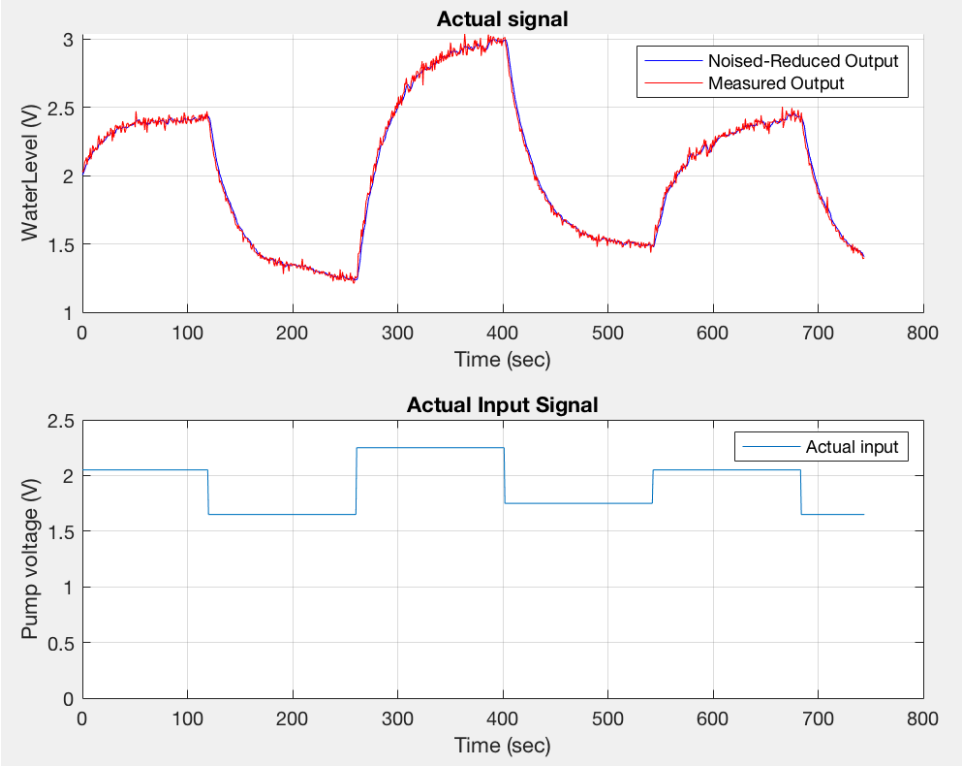


# Lab 5 result

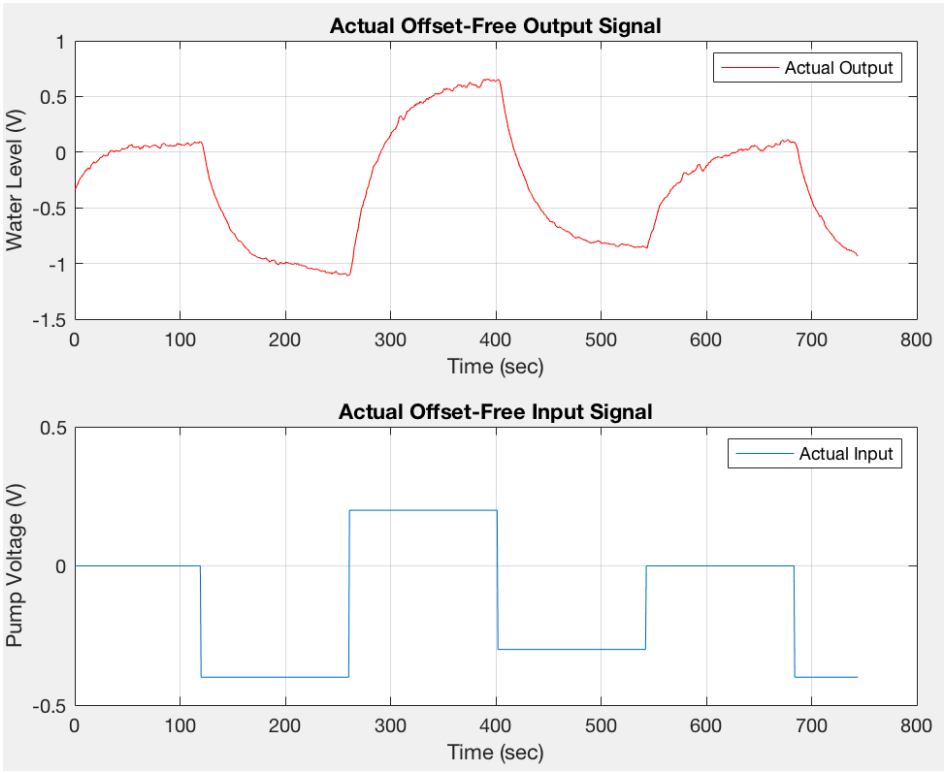
Raw data from Simulink (only the second period is used):



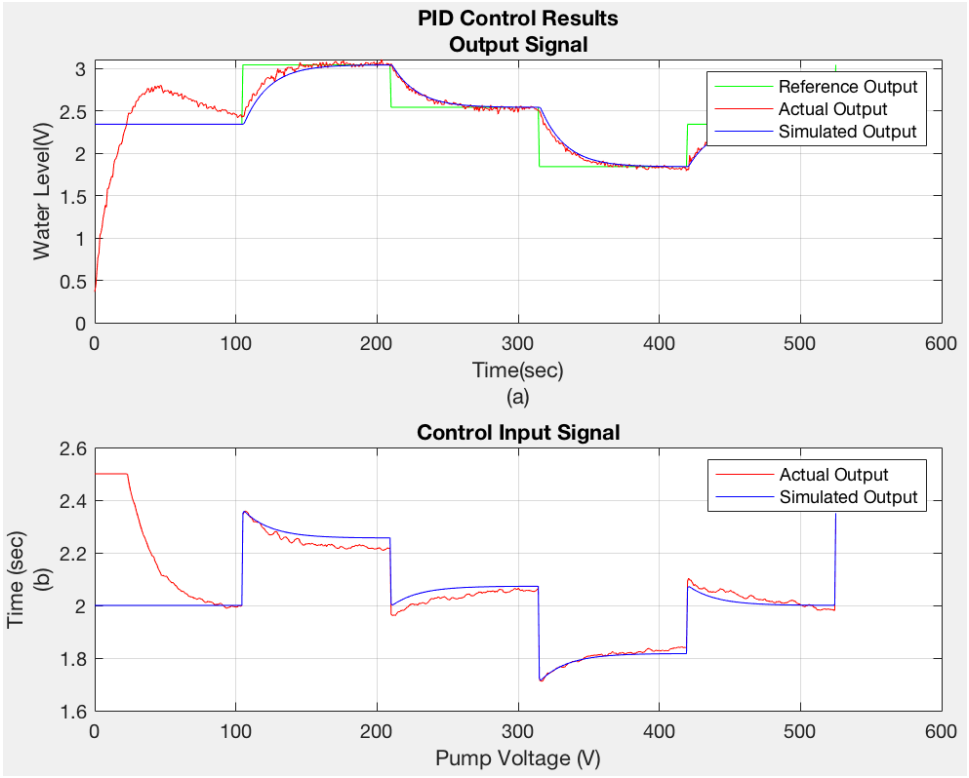

Non-offset free output, input:



Offset-free output, input:




Comparison between simulation and actual PI control of W-T system:




**Comment:** As before, Simulation and actual result differ a lot at the beginning, this is because initially the water tank system is not offset-free (start at 0 V).

# Knowledge learned from the lab

**Lab1:** In lab1 we learned how to perform system identification of a given data set using least square method. More specifically, we learned how to build first-order and second-order regression models of an unknown system. We also learned how to decide which model better fits the given data set by comparing the MSE.

**Lab2:** In lab 2 we learned how to apply techniques from lab 1 to reality. In addition, we learned how to calculate the input and output offset value base on the data we collected from the experiment.

**Lab3:** In lab 3 we learned how to design deadbeat controller and non-deadbeat controller. We also learned set-point control with observer design. We found out our control system lacks the ability to reject disturbance and noise in post-lab.

**Lab4:** In lab4 we learned how to apply techniques from lab3 to reality.

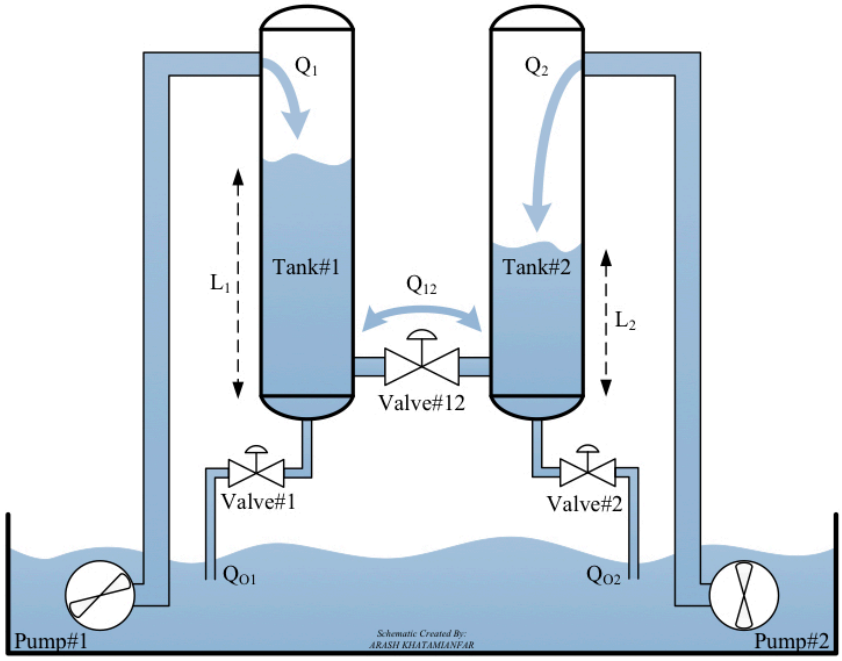
**Lab5:** In lab5 we learned how to implement PID controller for set-point control. System identification is still computed although this is not a necessary process for PID control.

# Conclusion

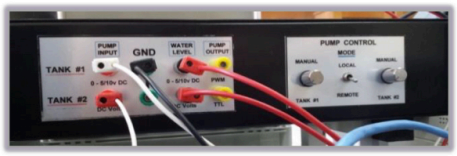
In this task, in order to control the water level of the water tank system, we have designed and implemented some modern control techniques such as system identification, deadbeat and non-deadbeat feedback control, state observer, PI control. We have also test out the model's accuracy by comparing the simulated result with the actual result (model verification). Disturbance rejection and noise robustness of the system have also been discussed. It turns out that both non-deadbeat control and PI control can efficiently control the water-level without exceeding the input voltage range.

# Appendix

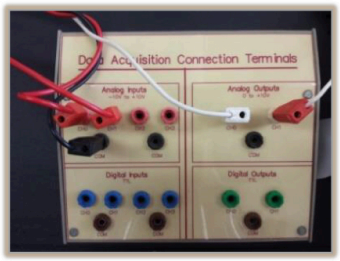
## Water tank system diagram



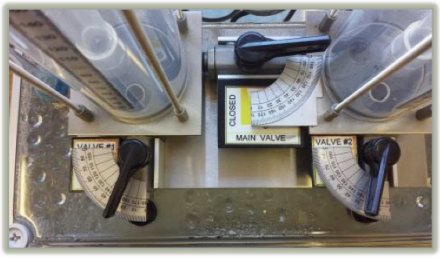

(a)




(b)




(c)




(d)

(a) Experimental W-T setup, (b) Valve position measurement, (c) Driver system and power supply, (d) Data acquisition terminal board.

Lab 1 Matlab code:

```
% %ELEC4632 lab1
% prelab %
clc
close all
clear
% data generation
N = (100-0)/0.1;
t = linspace(0,100,N);
noise = 0.4*rand(1,N) - 0.2; % generate noise from -0.2 to 0.2
y1 = sin(0.02*pi*t)+noise;
y2 = cos(0.02*pi*t)+noise;

y1_new = y1(200:end);
y2_new = y2(200:end);
t_new = t(1:length(y1_new));
data_new = [t_new; y1_new; y2_new;]';
% original data plotting
figure(1)
subplot(2,1,1)
plot(t,y1,'r')
xlim([0 140]);
ylim([-1.5 1.5]);
xlabel({'Time (sec)';'(a)'})
ylabel('Data')
title('Original Data')
grid on
hold on
plot(t,y2,'b')
legend('sin(0.02\pit)','cos(0.02\pit)');
hold off
% cut-off data plotting
subplot(2,1,2)
plot(t_new,y1_new,'g');
xlim([0 140]);
ylim([-1.5 1.5]);
xlabel({'Time (sec)';'(b)'})
ylabel('Data')
title('Cut-off Data')
grid on
hold on
plot(t_new,y2_new,'Color',[0.6 0.7 0.8])
legend('sin(0.02\pit)','cos(0.02\pit)');
hold off
% question 1 %
clear
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;
% find sampling time
Ts = (t(end)-t(1))/(length(t)-1);
```

```
%Ts = t(2)-t(1);
fprintf("sampling time according to calculation is %d\n",Ts);
figure(2)
subplot(2,1,1)
hold on
plot(t,y_act,'b');
xlim([0 700]);
ylim([1 4]);
xlabel('Time (sec)');
ylabel('WaterLevel (V)');
title('Actual signal');
grid on
plot(t,y_actm,'r');
legend('Noised-Reduced Output','Measured Output');
hold off
subplot(2,1,2)
plot(t,u_act);
xlim([0 700]);
ylim([1 3]);
xlabel('Time (sec)');
ylabel('Pump voltage (V)');
title('Actual Input Signal');
legend('Actual input');
grid on

% remove input offset
u_offset = u_act(1);
u = u_act - u_offset;
figure(3)
subplot(2,1,2)
plot(t,u)
xlim([0 700]);
ylim([-0.5 0.5]);
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
title('Actual Offset-Free Input Signal');
legend('Actual Input');
grid on
% remove output offset
count = 0;
i = 1;
while(u_act(i+1) == u_act(i))
    i=i+1;
    count = count + 1;
end
%y_offset = sum(y_act(1:count))/count;
y_offset = mean(y_act(1:count));
y = y_act - y_offset;
subplot(2,1,1);
plot(t,y,'r')
xlim([0 700]);
ylim([-2 1]);
grid on
xlabel('Time (sec)');
```

```
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (Entire)');
% move on to first order %
% same as before, start from k=10. k should be greater than 2
 otherwise
% index error will occur
k = 10;
phi1 = zeros(length(k:N),2);
%build the phi matrix for first order model
for i = k:N
    phi1(i-k+1,1) = y(i-1);
    phi1(i-k+1,2) = u(i-1);
end
theta_hat1 = inv(phi1'*phi1)*phi1'*y(k:N);
a1 = -theta_hat1(1);
b1 = theta_hat1(2);
H1 = tf(b1,[1 a1],Ts);
fprintf("Info about the first order state space model is below:\n");
sys1 = ss(H1)
y_simulate_firstOrder = filter(b1,[1 a1],u);
figure()
plot(t,y,'r');
hold on
plot(t,y_simulate_firstOrder,'g.');
plot(t,y_simulate_entire_2nd_order,'--b');
xlabel('Time (sec)');
ylabel('Water Level (V)');
xlim([0 700]);
ylim([-1.5 1]);
grid on
title('Comparison of Different Offset-Free Order Models');
legend('Actual Output','1^{st} Order Model Response','2^{nd} Order
 Model Resonse');
% first_order_mse = immse(y_simulate_firstOrder,y);
% second_order_mse = immse(y_simulate_entire_2nd_order,y);
first_order_mse = myMSE(y_simulate_firstOrder,y);
second_order_mse = myMSE(y_simulate_entire_2nd_order,y);
str = {'MSE1 = ',first_order_mse,'MSE2 = ',second_order_mse};
text(5,0.5,str)
```

To make our coding neater, we put all the work in lab1 into a function called second_order_regression, so we can call it whenever we want to perform system identification.

```
function [a1,a2,b1,b2] = second_order_regression(k,y,u)
% Second order model using least square method
% return parameters a1, a2, b1, b2

if k < 3
    % k should be greater than 2 otherwise index error
    % will occur
    fprintf('Index error, k must be greater than 2\n');
    pause;
end
N = round(length(y));
%build the phi matrix for second order model
phi = zeros(length(k:N),4);
for i = k:N
    phi(i-k+1,1) = y(i-1);
    phi(i-k+1,2) = y(i-2);
    phi(i-k+1,3) = u(i-1);
    phi(i-k+1,4) = u(i-2);
end

theta_hat = inv(phi'*phi)*phi'*y(k:N);
a1 = -theta_hat(1);
a2 = -theta_hat(2);
b1 = theta_hat(3);
b2 = theta_hat(4);
end
```

Lab 2 Matlab code:

```
clear
close all
clc
load SysIdenData_8.mat
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;
figure()
plot(y_act)
% truncate the first period, keep data afer 800.25 seconds
index = find(t==800.25);
y_act = y_act(index:end);
y_actm = y_actm(index:end);
u_act = u_act(index:end);
t = t(1:length(u_act));
Ts = (t(end)-t(1))/(length(t)-1); % find sampling time
%Ts = t(2)-t(1);
fprintf('sampling time according to calculation is %d\n',Ts);
figure()
subplot(2,1,1)
hold on
plot(t,y_act,'b');

xlabel('Time (sec)');
ylabel('WaterLevel (V)');
title('Actual signal');
grid on
plot(t,y_actm,'r');
legend('Noised-Reduced Output','Measured Output');
hold off
subplot(2,1,2)
plot(t,u_act);
xlabel('Time (sec)');
ylabel('Pump voltage (V)');
title('Actual Input Signal');
legend('Actual input');
ylim([0 2.5]);
grid on

% remove input offset
u_offset = u_act(1);
u = u_act - u_offset;
figure()
subplot(2,1,2)
plot(t,u)
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
title('Actual Offset-Free Input Signal');
ylim([-0.5 0.5])
```

```
legend('Actual Input');
grid on
% remove output offset
count = 0;
i = 1;
while(u_act(i+1) == u_act(i))
    i=i+1;
    count = count + 1;
end
y_offset = mean(y_act(1:count));
y = y_act - y_offset;
subplot(2,1,1);
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
title('Actual Offset-Free Output Signal');
legend('Actual Output');

N = round(length(y)/2);
% start from k = 10, k should be greater than 2
k = 10;
phi = zeros(length(k:N),4);

for i = k:N
    phi(i-k+1,1) = y(i-1);
    phi(i-k+1,2) = y(i-2);
    phi(i-k+1,3) = u(i-1);
    phi(i-k+1,4) = u(i-2);
end

theta_hat = inv(phi'*phi)*phi'*y(k:N);
a1 = -theta_hat(1);
a2 = -theta_hat(2);
b1 = theta_hat(3);
b2 = theta_hat(4);

%[a1,a2,b1,b2] = second_order_regression(k,y,u);
H = tf([b1 b2],[1 a1 a2],Ts);
fprintf('Info about second order state space model is below:\n');
sys = ss(H)

figure()
%simulate second half
subplot(2,1,1)
b = [b1 b2];
a = [1 a1 a2];
y_simulate_2nd_Half = filter(b,a,u(N:end));
plot(t(N:end),y_simulate_2nd_Half,'--');
hold on
plot(t(N:end),y(N:end),'r');
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
```

```
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (2^{nd} Half)');
hold off
% simulate entire
subplot(2,1,2)
y_simulate_entire_2nd_order = filter(b,a,u);
plot(t,y_simulate_entire_2nd_order,'--');
hold on
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (Entire)');

immse(y,y_simulate_entire_2nd_order)

close all
```

Lab 3 Matlab code:

```
% ELEC4632 lab 3
%prelab %
clear
close all
clc

load SysIdenData_8.mat
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;
% truncate the first period, keep data afer 800.25 seconds
index = find(t==800.25);
y_act = y_act(index:end);
y_actm = y_actm(index:end);
u_act = u_act(index:end);
t = t(1:length(u_act));
Ts = (t(end)-t(1))/(length(t)-1); % find sampling time
%Ts = t(2)-t(1);
fprintf('sampling time according to calculation is %d\n',Ts);
figure()
subplot(2,1,1)
hold on
plot(t,y_act,'b');

xlabel('Time (sec)');
ylabel('WaterLevel (V)');
title('Actual signal');
grid on
plot(t,y_actm,'r');
legend('Noised-Reduced Output','Measured Output');
hold off
subplot(2,1,2)
plot(t,u_act);
xlabel('Time (sec)');
ylabel('Pump voltage (V)');
title('Actual Input Signal');
legend('Actual input');
ylim([0 2.5]);
grid on
% remove input offset
u_offset = u_act(1);
u = u_act - u_offset;
figure()
subplot(2,1,2)
plot(t,u)
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
title('Actual Offset-Free Input Signal');
ylim([-0.5 0.5])
legend('Actual Input');
grid on
```

```
% remove output offset
count = 0;
i = 1;
while(u_act(i+1) == u_act(i))
    i=i+1;
    count = count + 1;
end
y_offset = mean(y_act(1:count));
y = y_act - y_offset;
subplot(2,1,1);
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
title('Actual Offset-Free Output Signal');
legend('Actual Output');

% start from k = 10, k should be greater than 2
k = 3;
[a1,a2,b1,b2] = second_order_regression(k,y,u);
H = tf([b1 b2],[1 a1 a2],Ts);
fprintf('Info about second order state space model is below:\n');
sys = ss(H)

%{
figure()
%simulate second half
subplot(2,1,1)
b = [b1 b2];
a = [1 a1 a2];
N = round(length(y)/2);
y_simulate_2nd_Half = filter(b,a,u(N:end));
plot(t(N:end),y_simulate_2nd_Half,'--');
hold on
plot(t(N:end),y(N:end),'r');
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (2^{nd} Half)');
hold off
%simulate entire
subplot(2,1,2)
y_simulate_entire_2nd_order = filter(b,a,u);
plot(t,y_simulate_entire_2nd_order,'--');
hold on
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (Entire)');
%}
A = sys.A;
```

```
eigenVal = eig(A);
%check stability from eigen values
if(isempty(find(eigenVal>0))==0)
    fprintf("Eigenvalues are %f and %f\n",eigenVal);
end
[z,gain] = zero(sys);

G = [0 1; -a2 -a1;];
H = [0; 1;];
C = [b2 b1];
D = 0;

Wc = [H G*H];
Wo = [C; C*G];

if (rank(Wc) == 2)
    fprintf("Wc has full rank, reachable.\n")
else
    fprintf("Wc has no full rank, not reachable.\n")
end

if (rank(Wo) == 2)
    fprintf("Wo has full rank, observable, obeservability implies
 detectability\n\n")
else
    fprintf("Wo has no full rank, not observable,\n\n")
end
% close all

% q1 %
% Initialize
x1 = 0;
x2 = 0.3;
% From now on use canonical observer form
G_obsrv = G';
H_obsrv = C';
C_obsrv = H';
D_obsrv = 0 ;
Wc_obsrv = [H_obsrv G_obsrv*H_obsrv];
Wo_obsrv = [C_obsrv; C_obsrv*G_obsrv];

% Dead beat control
L_db = [0 1]*inv([(G_obsrv^-2)*H_obsrv (G_obsrv^-1)*H_obsrv]);
%L_db = [0 1]*inv(Wc_obsrv)*(G^2);

% non dead beat control
ndb_eigVal1 = 0.2348;
ndb_eigenVal2 = 0.9;
p_coeffi = poly([ndb_eigVal1 ndb_eigenVal2]);

% Desired characteristic equation
P = p_coeffi(1)*G_obsrv^2 + p_coeffi(2)*G_obsrv + p_coeffi(3)*eye(2);

% Apply Ackermans's Formula
```

```
% L_ndb = [0 1]*inv(Wc_obsrv)*P
L_ndb = place(G_obsrv,H_obsrv,[ndb_eigVal1 ndb_eigenVal2]);

% Simulate y(k)
Tf = 50;
T = [0:Ts:50];

sys_ndb = ss((G_obsrv - H_obsrv*L_ndb),[],C_obsrv,D_obsrv,Ts);
sys_db = ss((G_obsrv - H_obsrv*L_db),[],C_obsrv,D_obsrv,Ts);
[y_ndb,t_ndb,x_ndb]= initial(sys_ndb,[x1 x2],Tf);
[y_db,t_db,x_db] = initial(sys_db,[x1 x2],Tf);

figure()
subplot(2,1,1)
stairs(T,y_ndb);
hold on
stairs(T,y_db);
ylim([-1 1]);
xlim([0 50]);
title("Regulation Response by State Feedback y(k)");
xlabel({"Time (sec)";"(a)"});
ylabel({"Offset-Free";"Water Level(V)"});
legend("None-Deadbeat Response","Deadbeat Response");
grid on
hold off

% simulate u(k)
u_ndb = -L_ndb*x_ndb';
u_db = -L_db*x_db';

subplot(2,1,2)
stairs(T,u_ndb)
hold on
stairs(T,u_db)
drawnow;
ylim([-1 1])
xlim([0 50])
title("Offset-Free Control Input u(k)")
xlabel({"Time (sec)";"(b)"});
ylabel({"Offset-Free";"Pump Voltage (V)"});
grid on
legend("None-Deadbeat Control Input","Deadbeat Control Input");
hold off

% q2 %
% Reference output
y_ref = [zeros(1,140) 0.7*ones(1,140) -0.2*ones(1,140) 0.5*ones(1,140)
 zeros(1,140)];

figure();
subplot(2,1,1)
plot([0:0.75:0.75*(length(y_ref)-1)],y_ref,'g');
grid on
ylim([-1 1]);
```

```
xlabel({"Time (sec)";"(a)"});
ylabel({"Offset-Free";"Water Level (V)"});
title({"Set-Point Control Results: Simulation";"Output Signal"});
hold on

% Desired characteristic equation
ndb_eigVal1 = 0.9;
ndb_eigenVal2 = 0.9;
p_coeffi = poly([ndb_eigVal1 ndb_eigenVal2]);
P = p_coeffi(1)*G_obsrv^2 + p_coeffi(2)*G_obsrv + p_coeffi(3)*eye(2);

% Apply Ackermans's Formula
L_ndb2 = [0 1]*inv(Wc_obsrv)*P;
%L_ndb2 = place(G_obsrv,H_obsrv,[ndb_eigVal1 ndb_eigenVal2]);

DC_gain = dcgain(ss(G_obsrv-H_obsrv*L_ndb2, H_obsrv,C_obsrv,
 D_obsrv,Ts));

sys_ndb2 = ss(G_obsrv - H_obsrv*L_ndb2,H_obsrv,C_obsrv,D_obsrv,Ts);
u_ndb2 = y_ref/DC_gain;
T = [0:Ts:Ts*(length(u_ndb2)-1)];
x1 = 0;
x2 = 0;
[y_spt,t_spt,x_spt] = lsim(sys_ndb2,u_ndb2,T,[x1 x2]);
plot(t_spt,y_spt,'r')
drawnow;
legend("Reference Output","Simulated Output")
subplot(2,1,2)

%Simulated Control Input
u_spt = -L_ndb2*x_spt' + y_ref/DC_gain;
plot(t_spt,u_spt)
grid on
xlabel({"Time (sec)";"(b)"});
ylabel({"Offset-Free";"Pump Voltage (V)"});
title("Control Input Signal")
legend("Simulated Control Input")

% q3 %
db_eigVal1 = 0.9;
db_eigenVal2 = 0.9;
p_coeffi = poly([db_eigVal1 db_eigenVal2]);
P =G^2; %deadbeat
%K = P*inv(Wo_obsrv)*[0 1]';
K = acker(G_obsrv',C_obsrv',[0; 0]);

figure();

subplot(3,1,1)
plot([0:0.75:0.75*(length(y_ref)-1)],y_ref,'g');
grid on
ylim([-1 1]);
xlabel({"Time (sec)";"(a)"});
ylabel({"Offset-Free";"Water Level (V)"});
```

```
title({"Set-Point Control Results: Simulation";"Output Signal"});
hold on
plot(t_spt,y_spt,'r')
drawnow;
legend("Reference Output","Simulated Output")

subplot(3,1,2)
plot(t_spt,u_spt)
drawnow;
grid on
xlabel({"Time (sec)";"(b)"});
ylabel({"Offset-Free";"Pump Voltage (V)"});
title("Control Input Signal")
legend("Simulated Control Input")

subplot(3,1,3)
% use simulink to load data first
fprintf("Fetching result from simulink, might take a moment...\n");
sim('lab3.slx')
fprintf("Simulink data has been fetched.\n");
stairs(linspace(0,3,length(error.signals.values(:,2))),error.signals.values(:,1),'
hold on
stairs(linspace(0,3,length(error.signals.values(:,2))),error.signals.values(:,2),'
title('State Estimation Error');
ylabel('Estimation Error');
xlabel({'Time (sec)','(c)'});
grid on

% Post-lab1 %
figure()
% Add disturbance to system
subplot(2,1,1)
plot(disturbance_output.time,disturbance_output.signals.values(:,1));
hold on
plot(disturbance_output.time,disturbance_output.signals.values(:,2));
xlabel({"Time (sec)";"(a)"});
ylabel({"Offset-Free";"Water Level (V)"});
title({"Set-Point Control Results: Simulation";"Output Signal"});
legend("Simulation output","Actual Output")
grid on

subplot(2,1,2)
plot(disturbance_output.time,disturbance_input.signals.values);
xlabel({"Time (sec)";"(b)"});
ylabel({"Offset-Free";"Pump Voltage (V)"});
title("Control Input Signal")
legend("Simulated Control Input")
grid on

% Post-lab2 %

figure();
subplot(2,1,1)
plot([0:0.75:0.75*(length(y_ref)-1)],y_ref,'g');
```

```
grid on
ylim([-1 1]);
xlabel({"Time (sec)";"(a)"});
ylabel({"Offset-Free";"Water Level (V)"});
title({"Set-Point Control Results: Simulation";"Output Signal"});
hold on

% Desired characteristic equation
ev1 = 0.9;
ev2 = 0.9;
p_coeffi = poly([ev1 ev2]);
P = p_coeffi(1)*G_obsrv^2 + p_coeffi(2)*G_obsrv + p_coeffi(3)*eye(2);

% Apply Ackermans's Formula
L_ndb2 = [0 1]*inv(Wc_obsrv)*P;

% Add uncertainty
a1_new = a1*(1+((rand(1)-0.5)/10));
a2_new = a2*(1+((rand(1)-0.5)/10));
b1_new = b1*(1+((rand(1)-0.5)/10));
b2_new = b2*(1+((rand(1)-0.5)/10));
% State-pace in control canonical form
G_new = [0 1; -a2_new -a1_new;]';
H_new = [b2_new b1_new]';
C_new = [0; 1;]';
D_new = 0;

% construct state-space system
sys_ndb_post2 = ss(G_new - H_new*L_ndb2,H_new,C_new,D_new,Ts);
DC_gain = dcgain(sys_ndb_post2);

u_ndb2 = y_ref/DC_gain;
T = [0:Ts:Ts*(length(u_ndb2)-1)];
x1 = 0;
x2 = 0;
[y_spt,t_spt,x_spt] = lsim(sys_ndb_post2,u_ndb2,T,[x1 x2]);
plot(t_spt,y_spt,'r')
drawnow;
legend("Reference Output","Simulated Output")
hold off
subplot(2,1,2)

%Simulated Control Input
u_spt = -L_ndb2*x_spt' + y_ref/DC_gain;
plot(t_spt,u_spt)
grid on
xlabel({"Time (sec)";"(b)"});
ylabel({"Offset-Free";"Pump Voltage (V)"});
title("Control Input Signal")
legend("Simulated Control Input")
```

Lab 4 Matlab code:

```
% ELEC4632 lab 4 %
% prelab %
clear
close all
clc

load SysIdenData_4.mat
load SFControlData_0.mat

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

%truncate the first period, keep data afer 927 seconds
index = max(find(t<=927));
y_act = y_act(index:end);
y_actm = y_actm(index:end);
u_act = u_act(index:end);
t = t(1:length(u_act));
%Ts = (t(end)-t(1))/(length(t)-1); % find sampling time
Ts = t(2)-t(1);
fprintf('sampling time according to calculation is %d\n',Ts);
figure()
subplot(2,1,1)
hold on
plot(t,y_act,'b');

xlabel('Time (sec)');
ylabel('WaterLevel (V)');
title('Actual signal');
grid on
plot(t,y_actm,'r');
legend('Noised-Reduced Output','Measured Output');
hold off
subplot(2,1,2)
plot(t,u_act);
xlabel('Time (sec)');
ylabel('Pump voltage (V)');
title('Actual Input Signal');
legend('Actual input');
ylim([0 2.5]);
grid on
% remove input offset
u_offset = u_act(1);
u = u_act - u_offset;
figure()
subplot(2,1,2)
plot(t,u)
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
title('Actual Offset-Free Input Signal');
```

```
ylim([-0.5 0.5])
legend('Actual Input');
grid on
% remove output offset
count = 0;
i = 1;
while(u_act(i+1) == u_act(i))
    i=i+1;
    count = count + 1;
end
y_offset = mean(y_act(1:count));
y = y_act - y_offset;
subplot(2,1,1);
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
title('Actual Offset-Free Output Signal');
legend('Actual Output');

% start from k = 10, k should be greater than 2
k = 3;
[a1,a2,b1,b2] = second_order_regression(k,y,u);
H = tf([b1 b2],[1 a1 a2],Ts);
fprintf('Info about second order state space model is below:\n');
sys = ss(H)

%{
figure()
%simulate second half
subplot(2,1,1)
b = [b1 b2];
a = [1 a1 a2];
N = round(length(y)/2);
y_simulate_2nd_Half = filter(b,a,u(N:end));
plot(t(N:end),y_simulate_2nd_Half,'--');
hold on
plot(t(N:end),y(N:end),'r');
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (2^{nd} Half)');
hold off
%simulate entire
subplot(2,1,2)
y_simulate_entire_2nd_order = filter(b,a,u);
plot(t,y_simulate_entire_2nd_order,'--');
hold on
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
```

```
title('Offset-Free Model Verification (Entire)');
%}
A = sys.A;
eigenVal = eig(A);
%check stability from eigen values
if(isempty(find(eigenVal>0))==0)
    fprintf('Eigenvalues are %f and %f\n',eigenVal);
end
[z,gain] = zero(sys);

G = [0 1; -a2 -a1;];
H = [0; 1;];
C = [b2 b1];
D = 0;

Wc = [H G*H];
Wo = [C; C*G];

if (rank(Wc) == 2)
    fprintf('Wc has full rank, reachable.\n')
else
    fprintf('Wc has no full rank, not reachable.\n')
end

if (rank(Wo) == 2)
    fprintf('Wo has full rank, observable, obeservability implies
 detectability\n\n')
else
    fprintf('Wo has no full rank, not observable,\n\n')
end

% q1 %
% Initialize
x1 = 0;
x2 = 0.3;
% From now on use canonical observer form
G_obsrv = G';
H_obsrv = C';
C_obsrv = H';
D_obsrv = 0 ;
Wc_obsrv = [H_obsrv G_obsrv*H_obsrv];
Wo_obsrv = [C_obsrv; C_obsrv*G_obsrv];

% Dead beat control
L_db = [0 1]*inv([(G_obsrv^-2)*H_obsrv (G_obsrv^-1)*H_obsrv]);
%L_db = [0 1]*inv(Wc_obsrv)*(G^2);

% non dead beat control
ndb_eigVal1 = 0.2348;
ndb_eigenVal2 = 0.9;
p_coeffi = poly([ndb_eigVal1 ndb_eigenVal2]);

% Desired characteristic equation
P = p_coeffi(1)*G_obsrv^2 + p_coeffi(2)*G_obsrv + p_coeffi(3)*eye(2);
```

```
% Apply Ackermans's Formula
% L_ndb = [0 1]*inv(Wc_obsrv)*P
L_ndb = place(G_obsrv,H_obsrv,[ndb_eigVal1 ndb_eigenVal2]);

% Simulate y(k)
Tf = 50;
T = [0:Ts:50];
sys_ndb = ss((G_obsrv - H_obsrv*L_ndb),[],C_obsrv,D_obsrv,Ts);
sys_db = ss((G_obsrv - H_obsrv*L_db),[],C_obsrv,D_obsrv,Ts);
[y_ndb,t_ndb,x_ndb]= initial(sys_ndb,[x1 x2],Tf);
[y_db,t_db,x_db] = initial(sys_db,[x1 x2],Tf);

figure()
subplot(2,1,1)
stairs(T,y_ndb);
hold on
stairs(T,y_db);
ylim([-1 1]);
xlim([0 50]);
title('Regulation Response by State Feedback y(k)');
xlabel({'Time (sec)';'(a)'});
ylabel({'Offset-Free';'Water Level(V)'});
legend('None-Deadbeat Response','Deadbeat Response');
grid on
hold off

% simulate u(k)
u_ndb = -L_ndb*x_ndb';
u_db = -L_db*x_db';

subplot(2,1,2)
stairs(T,u_ndb)
hold on
stairs(T,u_db)
drawnow;
ylim([-1 1])
xlim([0 50])
title('Offset-Free Control Input u(k)')
xlabel({'Time (sec)';'(b)'});
ylabel({'Offset-Free';'Pump Voltage (V)'});
grid on
legend('None-Deadbeat Control Input','Deadbeat Control Input');
hold off
% q2 %
% Reference output
%y_ref = [zeros(1,140) 0.7*ones(1,140) -0.2*ones(1,140)
 0.5*ones(1,140) zeros(1,140)];
y_ref = SFLogData.signals(1).values(:,1)';
figure();
subplot(2,1,1)
plot([0:0.75:0.75*(length(y_ref)-1)],y_ref,'g');
grid on
ylim([-1 1]);
```

```
xlabel({'Time (sec)';'(a)'});
ylabel({'Offset-Free';'Water Level (V)'});
title({'Set-Point Control Results: Simulation';'Output Signal'});
hold on

% Desired characteristic equation
ndb_eigVal1 = 0.9;
ndb_eigenVal2 = 0.9;
p_coeffi = poly([ndb_eigVal1 ndb_eigenVal2]);
P = p_coeffi(1)*G_obsrv^2 + p_coeffi(2)*G_obsrv + p_coeffi(3)*eye(2)

% Apply Ackermans's Formula
L_ndb2 = [0 1]*inv(Wc_obsrv)*P;
%L_ndb2 = place(G_obsrv,H_obsrv,[ndb_eigVal1 ndb_eigenVal2]);

DC_gain = dcgain(ss(G_obsrv-H_obsrv*L_ndb2, H_obsrv,C_obsrv,
 D_obsrv,Ts));

sys_ndb2 = ss(G_obsrv - H_obsrv*L_ndb2,H_obsrv,C_obsrv,D_obsrv,Ts);
u_ndb2 = y_ref/DC_gain;
T = [0:Ts:Ts*(length(u_ndb2)-1)];
x1 = 0;
x2 = 0;
[y_spt,t_spt,x_spt] = lsim(sys_ndb2,u_ndb2,T,[x1 x2]);
plot(t_spt,y_spt,'r')
drawnow;
legend('Reference Output','Simulated Output')
subplot(2,1,2)

%Simulated Control Input
u_spt = -L_ndb2*x_spt' + y_ref/DC_gain;
plot(t_spt,u_spt)
grid on
xlabel({'Time (sec)';'(b)'});
ylabel({'Offset-Free';'Pump Voltage (V)'});
title('Control Input Sddignal')
legend('Simulated Control Input')

% q3 %
db_eigVal1 = 0.9;
db_eigenVal2 = 0.9;
p_coeffi = poly([db_eigVal1 db_eigenVal2]);
P =G^2; %deadbeat
%K = P*inv(Wo_obsrv)*[0 1]';
K = acker(G_obsrv',C_obsrv',[0; 0]);

figure();

subplot(3,1,1)
plot([0:0.75:0.75*(length(y_ref)-1)],y_ref,'g');
grid on
ylim([-1 1]);
xlabel({'Time (sec)';'(a)'});
ylabel({'Offset-Free';'Water Level (V)'});
```

```
title({'Set-Point Control Results: Simulation';'Output Signal'});
hold on
plot(t_spt,y_spt,'r')
drawnow;
legend('Reference Output','Simulated Output')

subplot(3,1,2)
plot(t_spt,u_spt)
drawnow;
grid on
xlabel({'Time (sec)';'(b)'});
ylabel({'Offset-Free';'Pump Voltage (V)'});
title('Control Input Signal')
legend('Simulated Control Input')

subplot(3,1,3)
% use simulink to load data first
fprintf('Fetching result from simulink, might take a moment...\n');
sim('lab4.slx')
fprintf('Simulink data has been fetched.\n');
stairs(linspace(0,3,length(error.signals.values(:,2))),error.signals.values(:,1),'
hold on
stairs(linspace(0,3,length(error.signals.values(:,2))),error.signals.values(:,2),'
title('State Estimation Error');
ylabel('Estimation Error');
xlabel({'Time (sec)','(c)'});
grid on
% plot out the result %
load SFControlData_0.mat

treal = SFLogData.time;
yref = SFLogData.signals(1).values(:,1);
yreal = SFLogData.signals(1).values(:,2);
ureal = SFLogData.signals(2).values;

figure()
subplot(2,1,1)
plot(treal,yref,'g');
hold on
plot(treal,yreal,'r');
plot(t_spt,y_spt,'b');

grid on
xlabel({'Time (sec)';'(a)'});
ylabel({'Water Level (V)'});
title({'Output Feedback Control Results';'Output Signal'});
legend('Referece Output','Actual Output','Simulated Output')
subplot(2,1,2)
plot(treal,ureal,'g');
hold on
plot(treal,u_spt+2,'r')
xlabel({'Time (sec)';'(b)'});
ylabel({'Pump Voltage (V)'});
title('Control Input Signal')
```

Lab 5 Matlab code:

```
clear
close all
clc
load SysIdenData_1.mat
t = LogData.time;
y_act = LogData.signals(1).values(:,2);
y_actm = LogData.signals(1).values(:,1);
u_act = LogData.signals(2).values;
figure()
plot(t,y_act)
title('Raw data')
xlabel('Time (sec)')
ylabel('Pump Voltage (V)')
grid on
% truncate the first period, keep data afer 800.25 seconds
index = find((t==585));
y_act = y_act(index:end);
y_actm = y_actm(index:end);
u_act = u_act(index:end);
t = t(1:length(u_act));
Ts = (t(end)-t(1))/(length(t)-1); % find sampling time
%Ts = t(2)-t(1);
fprintf('sampling time according to calculation is %d\n',Ts);
figure()
subplot(2,1,1)
hold on
plot(t,y_act,'b');

xlabel('Time (sec)');
ylabel('WaterLevel (V)');
title('Actual signal');
grid on
plot(t,y_actm,'r');
legend('Noised-Reduced Output','Measured Output');
hold off
subplot(2,1,2)
plot(t,u_act);
xlabel('Time (sec)');
ylabel('Pump voltage (V)');
title('Actual Input Signal');
legend('Actual input');
ylim([0 2.5]);
grid on

% remove input offset
u_offset = u_act(1);
u = u_act - u_offset;

figure()
subplot(2,1,2)
```

```
plot(t,u)
xlabel('Time (sec)');
ylabel('Pump Voltage (V)');
title('Actual Offset-Free Input Signal');
ylim([-0.5 0.5])
legend('Actual Input');
grid on
% remove output offset
count = 0;
i = 1;
while(u_act(i+1) == u_act(i))
    i=i+1;
    count = count + 1;
end
y_offset = mean(y_act(1:count));
y = y_act - y_offset;
subplot(2,1,1);
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
title('Actual Offset-Free Output Signal');
legend('Actual Output');

N = round(length(y)/2);
% start from k = 10, k should be greater than 2
k = 10;
phi = zeros(length(k:N),4);

for i = k:N
    phi(i-k+1,1) = y(i-1);
    phi(i-k+1,2) = y(i-2);
    phi(i-k+1,3) = u(i-1);
    phi(i-k+1,4) = u(i-2);
end

theta_hat = inv(phi'*phi)*phi'*y(k:N);
a1 = -theta_hat(1);
a2 = -theta_hat(2);
b1 = theta_hat(3);
b2 = theta_hat(4);

%[a1,a2,b1,b2] = second_order_regression(k,y,u);
H = tf([b1 b2],[1 a1 a2],Ts);
fprintf('Info about second order state space model is below:\n');
sys = ss(H)

figure()
%simulate second half
subplot(2,1,1)
b = [b1 b2];
a = [1 a1 a2];
y_simulate_2nd_Half = filter(b,a,u(N:end));
plot(t(N:end),y_simulate_2nd_Half,'--');
```

```
hold on
plot(t(N:end),y(N:end),'r');
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (2^{nd} Half)');
hold off
% simulate entire
subplot(2,1,2)
y_simulate_entire_2nd_order = filter(b,a,u);
plot(t,y_simulate_entire_2nd_order,'--');
hold on
plot(t,y,'r')
grid on
xlabel('Time (sec)');
ylabel('Water Level (V)');
legend('Simulated Output','Actual Output');
title('Offset-Free Model Verification (Entire)');
MSE_val = immse(y,y_simulate_entire_2nd_order)

% state space model
G = [0 1; -a2 -a1;];
H = [0; 1;];
C = [b2 b1];
D = 0;
% observer form
G_obsrv = G';
H_obsrv = C';
C_obsrv = H';
D_obsrv = 0 ;
sys = ss(G_obsrv,H_obsrv,C_obsrv,D_obsrv,Ts)
D_obsrv = 0 ;


load SFControlData_0.mat
sim('lab5_model.slx')
treal = SFLogData.time;
yref = SFLogData.signals(1).values(:,1);
yreal = SFLogData.signals(1).values(:,2);
input_simulated = sim_in.signals.values;
output_simulated = sim_out.signals.values(:,2);
t_simulated = sim_in.time;
ureal = SFLogData.signals(2).values;

figure()
subplot(2,1,1)
plot(treal,yref,'g')
hold on
plot(treal,yreal,'r')
plot(t_simulated,output_simulated + y_offset,'b')
grid on
ylabel('Water Level(V)')
xlabel({'Time(sec)';'(a)'})
```

```
title({'PID Control Results';'Output Signal'})
legend('Reference Output','Actual Output','Simulated Output')

subplot(2,1,2)
plot(treal,ureal,'r')
hold on
plot(t_simulated,input_simulated + 2,'b')
xlabel('Pump Voltage (V)')
ylabel({'Time (sec)';'(b)'})
title('Control Input Signal')
legend('Actual Output','Simulated Output')
grid on
```

## Objective checklist

| | Objective requirements | |
|---|---|---|
| lab 1 | Design first-order and second-order regression models via least square method | Plot figure on Matlab |
| lab 2 | Design first-order, second-order regression models via least square method | Calculate input offset and output offset |
| lab 3 | Design deadbeat and non-deadbeat control | Set-point control with observer |
| lab 4 | Design deadbeat and non-deadbeat control | Set-point control with observer |
| lab 5 | Design second-order regression models via least square method | Design PID controller via simulink |