School of Electrical Engineering & Telecommunications

ELEC4632 Lab 5

Design and Real-time implementation of PI control system for set-point control

In this lab, you are going to 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. Similar to the situation you faced in Lab 4, our identified model from Lab 4 may no longer be valid for your specific W-T setup since some valves positions may have been changed. Therefore, for the purpose of this lab, you must repeat system identification, as what you did in Lab 2 and Lab 4, to be able to design and simulate your PID control system and validate it by implementing your controller on the actual W-T system. However, in real world, having a valid model for PID control system design is not essential since the structure of PID controller only depends on the reference signal, y_{ref}, and the measured output, y. In our case, to be able to simulate the control system first before its real-time implementation, we must have a working model of the W-T process. More details on PID controller structure in both continuous-time and its equivalent discrete-time are provided in the Appendix, which you are highly encouraged to read.

Note:

Pre-lab Exercise

a. If the PI transfer function in continuous-time is given as $G_{CT}(s)$, in Eq. (1) on the left, show that its equivalent discrete-time form in Z-domain is obtained as $G_{DT}(z)$, Eq. (1) on the right, using *Zero-Order-Hold* (ZOH) method. Note that in Eq. (1), K_p is the proportional gain, K_i is the integral gain, and h is the sampling time. Write your answer in the provided box.

$$G_{CT}(s)=\frac{U(s)}{E(s)}=K_p+\frac{K_i}{s}, \quad \xrightarrow{ZOH} \quad G_{DT}(z)=\frac{U(z)}{E(z)}=K_p+\frac{K_i h}{z-1}, \qquad (1)$$

Solution:

b. Use inverse Z-transform for $G_{DT}(z)$ in Eq. (1) to find the difference equation relating the input (error signal $e(k) = y_{ref}(k) - y(k)$) to the output (control input $u(k)$) of the discrete-time PI controller as shown in Fig. 1. Write your answer in the provided box.

Solution:

c. Using the block diagram of given in Fig. 1, find the closed-loop transfer function $G_{cl}(z) = Y(z)/Y_{ref}(z)$, and the transfer function from reference signal to control input, i.e., $U(z)/Y_{ref}(z)$. Write your answer in the provided box.

Solution:

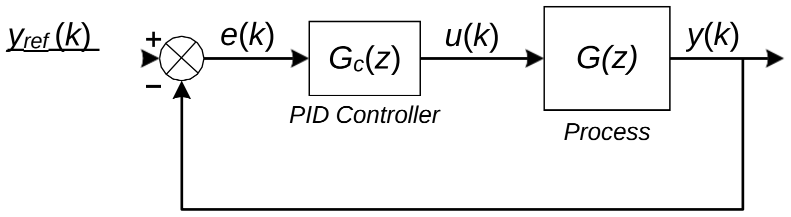

Fig. 1. Block diagram of a feedback control system with PID controller.

Lab Exercise (2 marks)

Make sure that you have your lecture notes on the topic of "Digital Control System Characteristics", and previous lab notes and your personal notes with yourself. You can always access lecture and lab notes via Moodle as well. Now, follow the steps below for this lab exercise.

1. **System Identification (0.5 marks, checked after 50 minutes)**

   a. Repeat system identification on the W-T system as you did before in Lab 2. Show your results to the demonstrator to receive the mark for this part of the lab exercise by plotting the figures that you were asked before in Labs 1 and 2.

2. **PI control design (0.5 marks, checked after 1 hour 30 minutes)**

   a. Once a good model is obtained, use either **MATLAB or Simulink** to design and simulate a PI controller for your identified system (check the remarks below). You can assume initial conditions are zero. Use the same reference signal for the control system to track as fast as possible without exceeding the control input limits, i.e., $y_{ref}(k) = \{0, 0.7, -0.2, 0.5, 0\}$ with each level period to be 140×0.75 = 105sec. For a proper design, you need to find good values of PI controller gains K_p and K_i such that the *overshoot* remains less than 2 percent. Also, make sure your choice of PI controller gains results in control input remaining within its limits.

   b. There are several methods for selecting proper PI gains to achieve the desired control performance. The most trivial method is using the trial-and-error approach. We know that from many resources that increasing K_p reduces the rise time (increasing response speed) and adding K_i removes steady state error as well as slowing down the response while it increases overshoot. You can begin by choosing $K_p = 0.5$ and $K_p = 0.01$, run the simulation and examine the output following the reference signal and control input. To improve the control system performance, change K_p by 0.05 increments/decrements and K_i by 0.005 increments/decrements. Other famous empirical methods to find suitable K_p and K_p is the so-called **Ziegler-Nichols Step-Response** method and **Ziegler-Nichols Ultimate- Sensitivity** method, which is explained in Appendix. MATLAB and Simulink also provide automatic tuning for PID, particularly if you use built-in PID Controller block in Simulink.

   Remarks:

   - Note that we are designing just a PI controller here. The reason for not using the full PID controller with derivative action is that the measurements are noisy and adding derivative of the error signal to the controller would aggravate the noise in the closed-loop system. Refer to Appendix for more discussion on reducing the side effects of derivative action in PID controllers.
   - To design the PI control system in **MATLAB**, you can define the closed-loop transfer function $G_{cl}(z) = Y(z)/Yref(z)$ (obtained in your prelab) and then use "*lsim"* function to simulate the control system. Note that you can only get the output when using *lsim* with G_{cl}.Therefore, you also need to use lsim a second time with the transfer function from reference signal to control input U(z)/Yref(z) and the simulated output obtained to find the control input values separately. You may find the MATLAB functions "*feedback"* and "*pid"* useful. You can also use other methods in MATLAB.
   - To design the PI control system in **Simulink**, You can build the block diagram using transfer functions similar to Fig. 2. Another possible way is to build the system using state space representation (i.e. using G, H and C obtained from system identification) and using the built-in PID Controller block in Simulink.

3. <u>**Real-time implementation of the control system (1 marks, checked after 2 hours)**</u>

a. When you are satisfied with simulation results, download the pre-built Simulink file named *WaterTankSysControlPID.slx* from Moodle. Set your model and controller parameter in the real-time model as shown in Fig. 3. The parameters are *PI Controller* block parameters *Kp* and *Ki*, and input and output offsets, *u_offset* and *y_offset*, respectively. You just need to assign them with their values in MATLAB Workspace as Simulink can read them from there. Moreover, you should change the default values in *Saturation TANK#1* block to V_{max} and V_{min} of your W-T setup.

b. After making sure all the proper settings are in place, run the Simulink model. The running time is set to 5×140×0.75 = 525sec or 8 minutes and 45 seconds, and the program will stop after this time. The data will be recorded in MATLAB Workspace as *PIDLogData* in Structure format, and it is saved in the directory "Documents/MATLAB" as `PIDControlData_0.mat`. Similar to what was explained in Lab 2 for data recording, if you repeat the experiment, the new data will be save under the same name with an increment of one unit, so you would never lose any test data.

c. **Copy the auto-generated data files (found in *Documents/MATLAB*) for both system identification (i.e. all `SysIdenData_x.mat`) and control system implementation (i.e. all `PIDControlData_x.mat`) in addition to your codes and/or Simulink model to your own computer. You will need these for the final report.**

d. Finally, extract the data as shown below and plot them against your simulated results similar to Fig. 4.

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

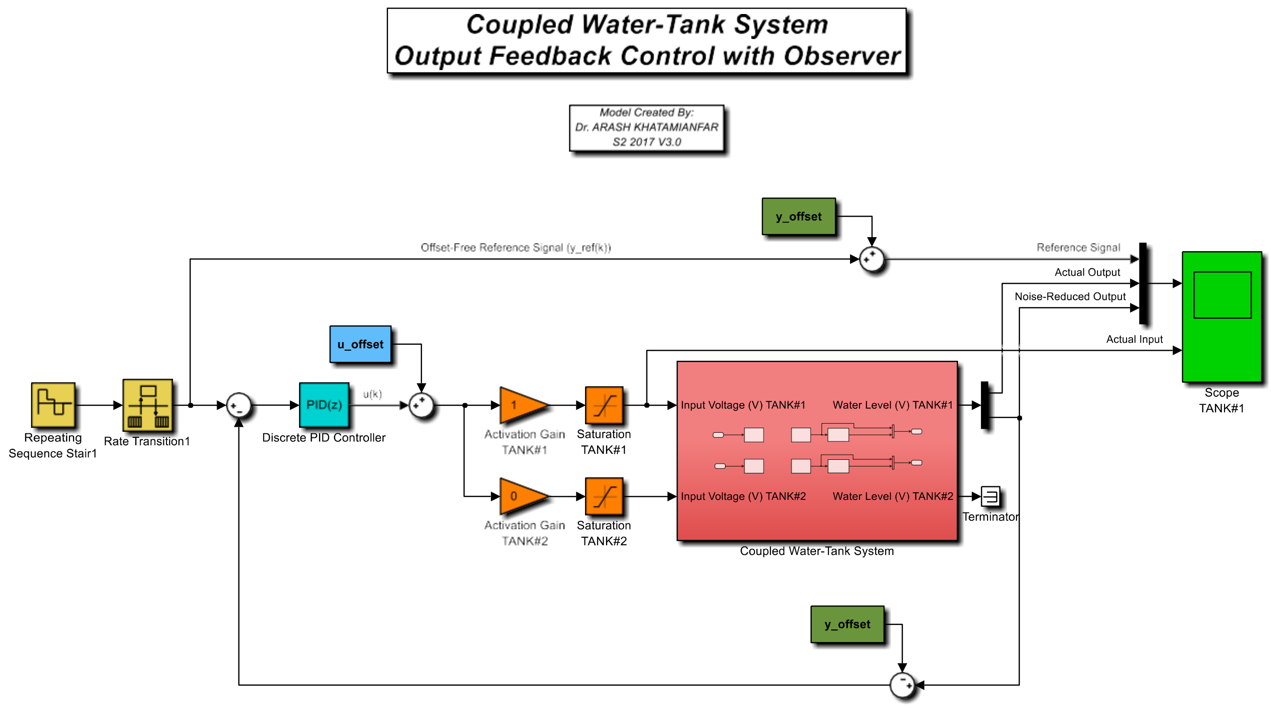

Fig. 3. Pre-built Simulink model for real-time output state feedback control with observer.

This figure illustrates the actual PID control results obtained from one of the W-T systems. It compares the real-time results with the simulated ones. The PI controller gains were chosen as $K_p = 0.68$ and $K_p = 0.03$ for this test with zero initial conditions. As you can see, the practical results are quite similar to the simulation ones for output signal in Fig. 4(a) and control input signal in Fig. 4(b). Both simulated output and control input signals are shifted up by output offset and input offset, respectively. This confirms that the identified model was accurate enough to represent the process andto be used for the controller design, as well as validity of the PI control system design.

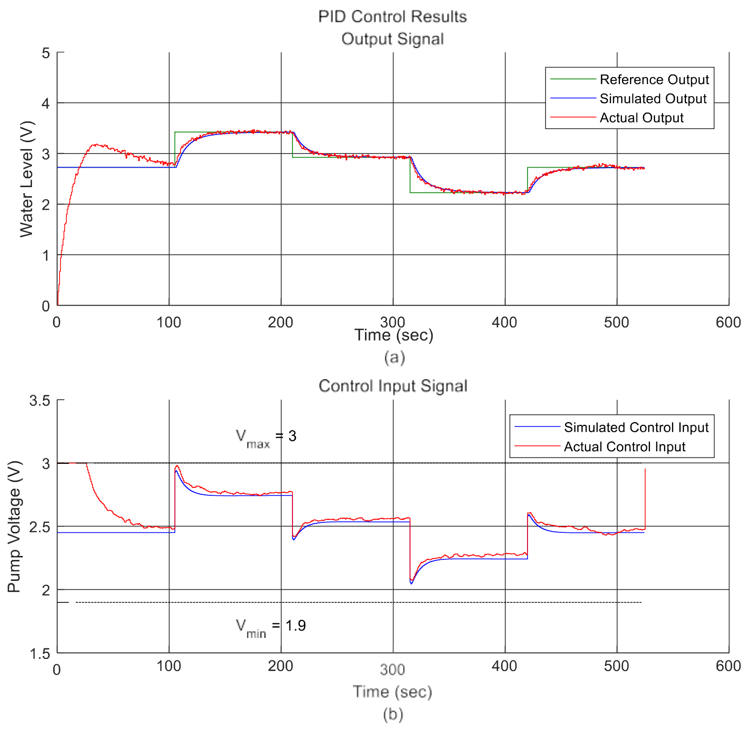

Fig. 4. Comparison between simulated and actual PI control of W-T system in set-point tracking, (a) Output for different water levels, (b) Control input.

a. ***Optional as Bonus***: Can you explain the behaviour of the real-time control operation in the first period (initial transient behavior) shown in Fig. 4? Why is the control input in Fig. 4(b) saturated at the beginning?

Appendix

Many practical control problems are solved by PID-controllers. The well-known version of the PID-controller can be described by the following equation [1],

$$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(\lambda)d\lambda + T_d\frac{de(t)}{dt}\right) \qquad \text{(A1)}$$

where u is the control input, *e* is the difference between the reference signal y_{ref} (the set-point signal) and process output *y* (see Fig. 1), K_p is the *proportional gain* of the controller, T_i is the *integration time* or *reset time*, and T_d is the *derivative time*. This version can also be described in terms of controller gains as below

$$u(t) = K_p e(t) + K_i\int_0^t e(\lambda)d\lambda + K_d\frac{de(t)}{dt} \quad \rightarrow \begin{cases} K_i = \frac{K_p}{T_i} \\ K_d = K_p T_d \end{cases} \qquad \text{(A2)}$$

where K_i and K_d are integration gain and derivative gain, respectively. The PID-controller was originally implemented using analog technology that went through several development stages, that is, pneumatic valves, relays and motors, transistors and Op-Amps, and integrated circuits. In this section, we will discuss the digital PID-controller in some detail.

Pure derivative cannot, and should not, be implemented, because it amplifies measurement noise in the control loop. The gain of the derivative must be limited and therefore, the PID control law is implemented in Laplace domain as follows,

$$\frac{de(t)}{dt} \xrightarrow{\text{Laplace Transform}} sE(s) \approx \frac{s}{1+\frac{s}{N}}E(s) = \frac{Ns}{N+s}E(s), \qquad \text{(A3)}$$

where *N* is the gain of derivative at high frequencies. Thus, the transfer function for the PID controller is given as follows,

$$G_{CT}(s) = \frac{U(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s} + \frac{T_d s}{1+\frac{s}{N}}\right), \qquad \text{(A4)}$$

Discretization of PID Controller

There are multiple methods for discretization of PID controller equation given in Eq. (A4), for instance Zero-Order-Hold (ZOH) method you used in pre-Lab exercise (not an easy method for discretization of the derivative part). The integral part can be discretised by using Forward Euler method, Backward Euler method, or trapezoidal (Tustin or bipolar transformation) method. The derivative part can be discretised by Forward Difference method or Backward Difference method. However, given the special form of the approximate derivate part in Eq. (A3), we can use Forward Euler method for both integrator and differentiator to discretize 1/s term since

$$\frac{s}{1+\frac{s}{N}} = \frac{N}{N\frac{1}{s}+1}. \qquad \text{(A5)}$$

In forward Euler method, the integrator part, I(t), is approximated as below,

$$\begin{cases} I(t)=\dfrac{1}{T_i}\displaystyle\int_0^t e(\lambda)d\lambda \\ I(s)=\dfrac{1}{T_i s}E(s) \end{cases} \xrightarrow{\text{Forward Euler, } t=kh} \begin{cases} I(kh)=I(kh-h)+\dfrac{h}{T_i}e(kh-h) \\ I(z)=\dfrac{h}{T_i(z-1)}E(z) \end{cases}, \tag{A6}$$

and, the derivative part, D(t), (or the term 1/s) in (A5) can also be discretised as follows,

$$D(s)=T_d\frac{N}{N\frac{1}{s}+1}E(s) \xrightarrow{\text{Forward Euler, } t=kh} D(z)=T_d\frac{N}{N\frac{h}{z-1}+1}E(z). \tag{A7}$$

Therefore, The transfer function of the PID controller is given as the following,

$$G_{DT}(z)=\frac{U(z)}{E(z)}=K_p\left(1+\frac{h}{T_i(z-1)}+T_d\frac{N}{N\frac{h}{z-1}+1}\right). \tag{A8}$$

Tunning a PID Controller with the Ziegler-Nichols methods

A PID-controller has parameters K_p, T_i, T_d, and N, that must be chosen. There is another parameter, T_I, known as *tracking-time constant* or *expected-time constant* of the closed-loop system. The primary parameters are K_p, T_i, and T_d. Parameter N can often be given a fixed default value, for example, N =100. Some special methods have, however, been developed to tune the PID parameters experimentally. The behavior of the discrete-time PID-controller is very close tothe analog PID-controller if the sampling interval is short. The traditional tuning rules for continuous-time controllers can thus be used. There are two classical heuristic rules developed by Ziegler and Nichols (1942) that can be used to determine the controller parameters: the *step- response method* and the *ultimate-sensitivity method*.

1. **Step-response method**

   In this method, the unit step response of the open-loop process is determined experimentally. The technique can be applied to processes whose step response is monotone or essentially monotone apart from an initial non-minimum phase characteristic. To use the method the tangent to the step response that has the steepest slope is drawn and the intersections of the tangent with the axes are determined as shown in Fig. A1. The controller parameters are then obtained from Table A1. The Ziegler-Nichols rule was designed to give good response to load disturbances. It does, however, give low damping of the dominant poles.

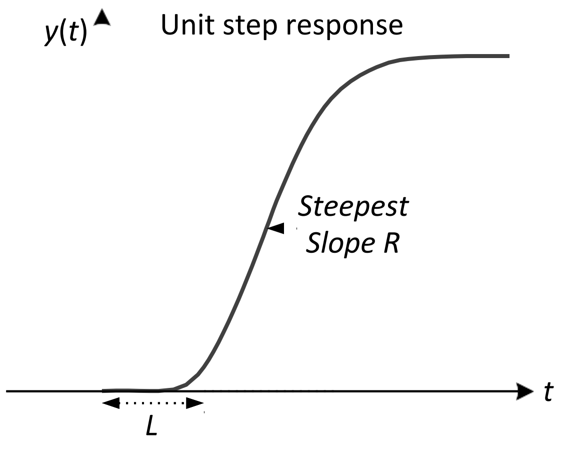

Fig. A1. Unit step response of a system with parameters needed for Ziegler-Nichols step-response method.

Table A1: PID parameters obtained from the Ziegler-Nichols step-response method, $a = RL$.

Controller Type	K_p	T_i	T_d	T_l
P	$1/a$			$4L$
PI	$0.9/a$	$3L$		$5.7L$
PID	$1.2/a$	$2L$	$0.5L$	$3.4L$

2. **Ultimate-sensitivity method**

In this method, the key idea is to determine the point where the Nyquist curve of the open-loop system intersects the negative real axis. This is done by connecting the controller to the process and setting the parameters so that pure proportional control is obtained. The gain of the controller is then increased until the closed-loop system reaches the stability limit (self-oscillation). The gain critical gain, K_u, when this occurs and the period of the oscillation, T_u, are determined. These parameters are called ultimate gain and ultimate period. The controller parameters are then given by Table A2.

Table A2: PID parameters obtained from the Ziegler-Nichols ultimate-sensitivity method.

Controller Type	K_p	T_i	T_d	T_l
P	$0.5K_u$			T_u
PI	$0.45K_u$	$T_u/1.2$		$1.4T_u$
PID	$0.6K_u$	$T_u/2$	$T_u/8$	$0.85T_u$

References

[1] K. J. Astrom and B. Wittenmark, *Adaptive Control, Chapter 8*. 2nd ed., Upper Saddle River, NJ: Prentice Hall, 1995.

V1 by Dr. Hailong Huang, July 2018.

V0 by Dr. Arash KHATAMIANFAR, July 2017.