

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

ECEN315 LABORATORY REPORT TWO

Xiaobin(Jerry) Zhuang

Submitted in partial fulfilment of the requirements for
the degree of Bachelor of Science

Abstract

This document is about the report of the last three labs of the entire lab project. Based on the model and the transfer function we have got from the previous lab, we will first implement the open loop control and then close loop control using the proportional gain element. Finally, adding an integral and a derivative control block to complete our PID control system. We will use Simulink to design our PID controller.

1. Introduction

This ECEN315 lab project based on controlling a motorised pendulum system. The last three labs aim to implement a PID controller, according to the result of the first three labs. There was some problem that needs to be solved in this project:

1. In the previous lab, the pendulum position was controlled by our setting of the power supply's output voltage, which is not a practical way to set the control voltage in an automated control system.
2. After we implemented the open loop control to the system, it could not act against disturbances.
3. There are still some drawbacks after we implement the close loop control with a pure proportional gain controller. For example, we cannot arbitrarily select an operating point for our system.

First, we connect a PWM controller to control the duty cycle of the power supply. Next, calibrate the system and design an open loop controller. Next, determine the closed loop transfer function from the open loop transfer function we had derived in lab 3. Next, use Matlab to predict whether the close loop system would be stable or not. Then, placing a pure proportional gain into the system to construct a closed loop system. Last, adding PI and PD controller to complete our design of PID controller. Finally, tuning the PID controller to achieve the system's automatic control.

2. Background

To set our pendulum at a certain point, we required our motor to output different power to fit the required point. And it would be able to against the disturbance. So we needed a PID controller. As defined in [1], The term PID stands for Proportional, Integral and Derivative. A PID controller is part of a feedback system. A PID system uses Proportional, Integral, and Derivative drive elements to control a process. The proportional part is a gain, almost always an essential part of a closed loop control system. The integral component is an integrator that can integrate voltage as time progresses. Finally, the derivative part is a differentiator, which can measure the rate of change.

3. Methods and Results

1. Open Loop Control

Before starting Simulink, we connect our device's one side with a switch box, and the other side with power supply. Also connect switch box with computer for future Simulink control.

In our Simulink program, we used PWM Arduino to control motor speed. The PWM can convert an input voltage(10V) to an average voltage through its duty cycle (D).

The relationship was given by: $V_{avg} = D * V_{in}$

In our system, we built up a chain of block that had the angle as input and the PWM Arduino as output, drop this output into the motor switch box and select pin 4. This is our D/A output. Along this chain, we added some display to observe the voltage, PWM and bits.

We set input voltage from 0 to 10V in steps of 1V, measured the corresponding pendulum angle. So we can get the relationship between them, given by:

$$V = (\text{angle} + 4) / 4.9090$$

There was a resolution of 255 steps on the Arduino output. So we had the block diagram shown in figure 1:

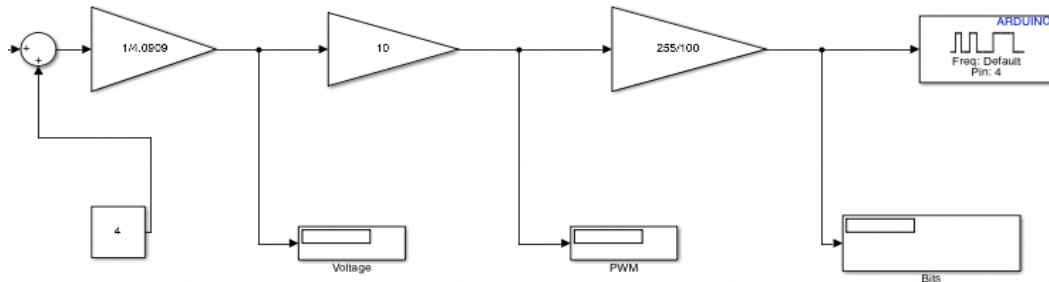


Figure 1

Then we placed an Arduino analogue input block and select pin 15. It took the analogue voltage from the potentiometer, then go through the Simulink. So we calibrated this unit by moving the pendulum from 0 to 40 degrees, observing the output bits, and finding the relationship between them.

There was a problem that happened during this part. When we observed the bits, we realized that the number would constantly decrease even if the pendulum were stable. It was challenging to get the equation under this condition.

Eventually, we found the reason was that our switch box was broken. So we replaced to a new box, and measured again, derive the following equation:

$$\text{Angle} = 0.2879 * \text{bits} - 176.13$$

So we constructed the block diagram shown below:

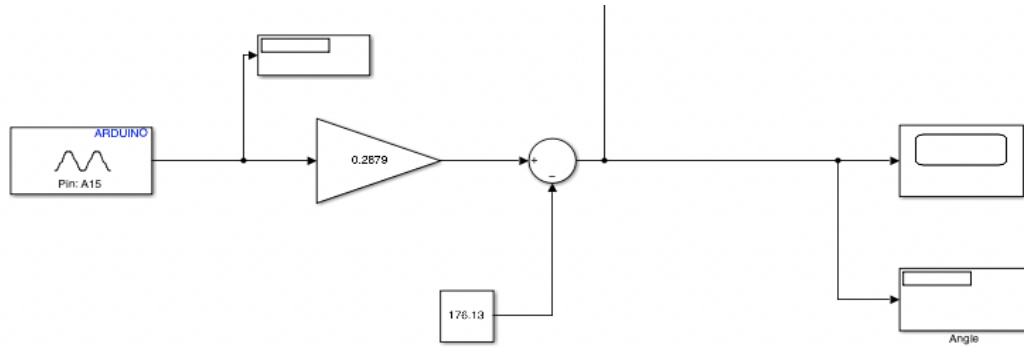


Figure 2

We were able to observe the angle, and compare with the require angle.

In total, we had a constant input, which is an angle, then go through calibrate process, finally, go to the D/A. So when we inputted 20 degrees, the pendulum would rise to approximately 20 degrees. Because it was open loop control, so it was not so accurate. However, if the pendulum was disturbed after reaching equilibrium, it would become unstable. It was also because the open loop control was not able to act against the disturbance.

2. Closed Loop Control – Proportional gain

We first determine the closed loop transfer function of our system from the open loop transfer function we had derived in lab 3.

$$22.913$$

The open loop transfer function : -----

$$0.206 s^4 + 13.6244 s^3 + 241.9 s^2 + 48.27 s + 268.302$$

Apply the following matlab code to the closed loop transfer function,

```

1 - clear all;
2 - sys = tf(22.913,[0.206 13.6244 241.9 48.27 268.302]) %open loop TF
3 - closedSys = feedback(1*sys,1) %gain is the multiplier of sys
4 - pzmap(closedSys)
5 - for i=0.1:0.1:100
6 -     gain = i
7 -     closedSys = feedback(gain*sys,1);
8 -     x = isstable(closedSys);
9 -     if x==0 %its unstable
10 -         break
11 -     end
12 - end
13 - gain % the gain that makes the system unstable (>=25.6)
14 -
15 - gain = gain - 10;
16 - stepplot(closedSys)

```

Command Window

```

25.6000

gain =

    25.6000

fx >>

```

Figure 3

When the gain = 25.6, the system would be unstable, and the value below 25.6 would make system stable.

We added a gain K_p into the block and change the Simulink model into close loop system. We tried to run the system in the closed loop using different values of K_p . We started to try with very small value $K_p = 0.2$. The system was stable, but it was because the minimal voltage of the motor.

From $K_p = 0.5$ to 1 in the step of 0.25, they all showed good stability. However $K_p = 0.75$ was the best value because it took the shortest time to rise from 0 degree to 20 degree.

From $K_p = 1.59$ to 19.59, they all caused the system unstable.

From $K_p = 20.59$ to 23.59, system was very unstable with disturbance. It could be stable without disturbance, because voltage reached the maximum.

From $K_p = 24.59$ to 25.59, system was very unstable. Because the poles move to the left.

So we choose $K_p = 0.75$ as our K_p value.

The actual unstable gain was a little smaller than the Matlab result. Reason might be the error in the measurement.

The advantage of proportional controller is: it helps in reducing the steady-state error, thus makes the system more stable.

While the disadvantage is that due to the presence of proportional controller, we would get some offsets in the system. It also increase the maximum overshoot of the system. These might eventually lead the system to unstable.

3. Closed Loop Control – PID compensation

PI compensator:

There were some drawback of a pure proportional controller, so we added an integral controller to improve the steady state error. [1] The integral controller operates when an error is present. It accumulates this error over time. Therefore, a small error can become a large correction, given enough time. As the error is accumulated, the system will be forced to correct the error.

By placing the integrating element in parallel with our proportional gain as shown below:

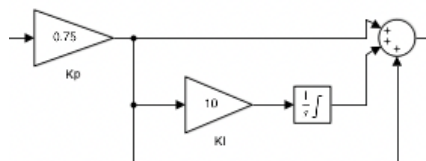


Figure 4

Base on the selected K_p value before (0.75), we would choose K_i value from 0.1 to 15 by testing the rising time from 0 to 25 degree of the pendulum.

However, we could not find the suitable value of K_i at first because we were using the switch, which caused the integrator to wind up. Thus, we changed the integrator setting and tested the next value until the voltage drops to 0.

We chose $K_i = 10$ because it took 4 seconds (the fastest) to go from 0 degrees (voltage 0) to our set angle of 25 degrees. However it overshoot at 25 degrees once, which is PI compensator natural characteristic. The experiment data shown in appendices, table 1.

PD compensator:

We placed a derivative compensator in parallel with both proportional gain and PI compensator to solve the overshoot problem. [1] The output of the differentiator is proportional to the speed of the system. If the system is moving fast, the output is high and vice versa. If the motor is not moving, the differential output will be zero. The

differential term is applied in such a way as to slow down the motor. Therefore, the derivative compensator would eliminate the overshoot.

A different way with selecting KI, we tested the system response to a disturbance when pushing the pendulum from 20 degree to 0 degree. We tested different KD value from 0.01 to 1 until finding the value show the best performance. We finally choose $KD = 0.75$ as it had shortest time of oscillation. Graph shown in figure 5.

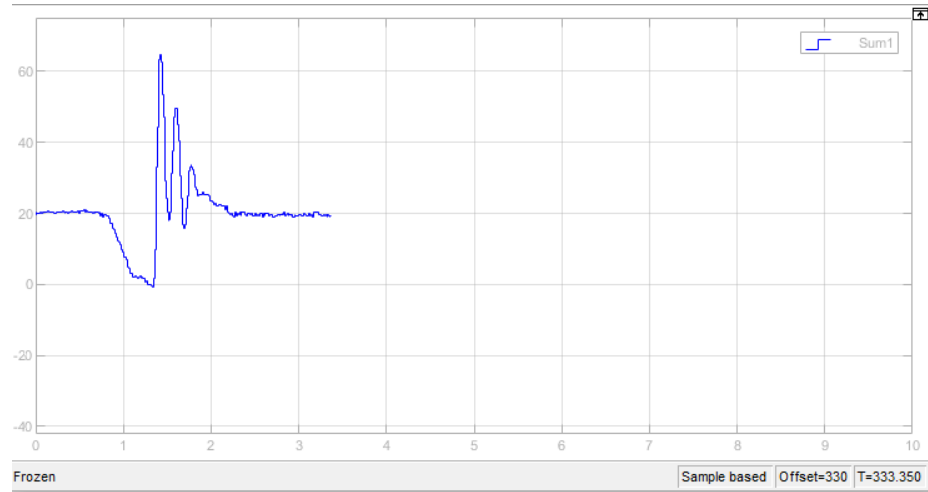


Figure 5

The rest of experiment data shown in appendices, table 2.

4. Tuning a PID controller using Ziegler Nicholl's rules

The Ziegler Nicholl's rules provide us with an empirical method of tuning a PID controller with no knowledge of the system's transfer function. According to the method introduced in [2], the value of K_p at the point of instability is called K_{max} .

Thus, set our $K_{max} = 1.8$. The frequency of oscillation is $f_0 = 0.62\text{Hz}$

For PID controller,

$$K_p = 0.6K_{max} = 1.08, K_I = 2.0 \cdot f_0 = 1.24, K_D = 0.125/f_0 = 0.201$$

Based on these three values, we got the other PID compensator. When we pushed the pendulum back to 0 degrees for testing the system's response to the disturbance, we got the graph shown below:

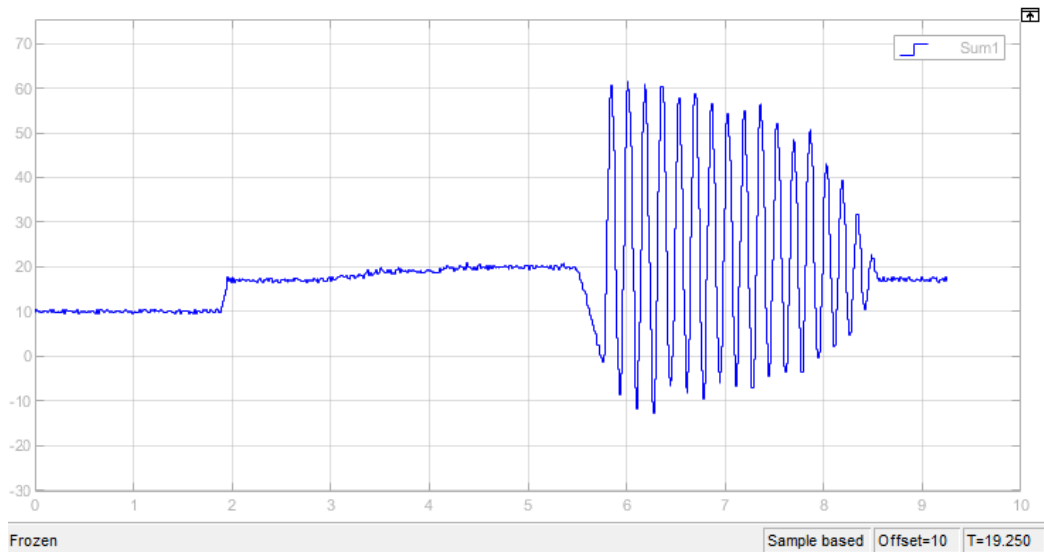


Figure 6

The system showed a worse stability than the PID compensator in the last section after placing a disturbance.

Adding PID compensator into system, we got the transfer function:

$$83.41 s^2 + 83.41 s + 1112$$

$$s^5 + 66.12 s^4 + 1174 s^3 + 150.9 s^2 + 1219 s - 1112$$

which is in a strange form. And according to the pzmap, there was a pole on the left side, which mean the system is unstable. However, this result is different from our system's stability. So we were unable to tell how the poles and zeros move. But one thing we could say was that all of the poles must be on the right side.

4. Discussion

- Our system worked well. When placing a disturbance, the system would automatically control itself, not go unstable, and move back to the original position.
- Due to the time limit, we could not derive the transfer function after adding the PID compensator.
- For the open loop control part, when we discovered our bits number constantly dropping, we should have stopped and asked for help instead of doing the next part.

- It was better to use the same approach to tune the derivative compensator as tuning integral compensator.
- Knowing different system response with different control block, and the roles of proportional controller, PI controller and PD controller in the PID controller.
- The most significant finding in this lab is designing an actual PID controller with a tuning technique. It was very exciting to see our system getting better more stable.
- Simulink is a powerful tool for the control system. It was worth knowing how to use it.

5. Conclusions

In this lab, we implemented the open loop control to the system, then improve it to closed loop control with pure proportional gain. Eventually, design a PID controller for the system using the tuning technique.

References

- [1] Dahlen, A., 2005. The PID Controller — Part 1. [online] Nuts and Volts Magazine. Available at: <https://www.nutsvolts.com/magazine/article/the_pid_controller_part_1> [Accessed 28 May 2021].
- [2] Ellis, G., 2012. Ziegler Nichols Method - an overview | ScienceDirect Topics. [online] Sciencedirect.com. Available at: <<https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method>>

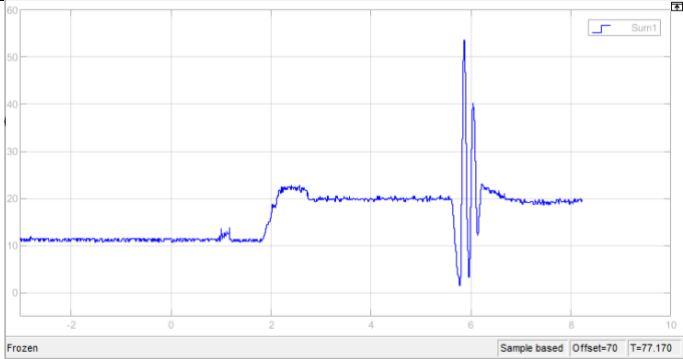
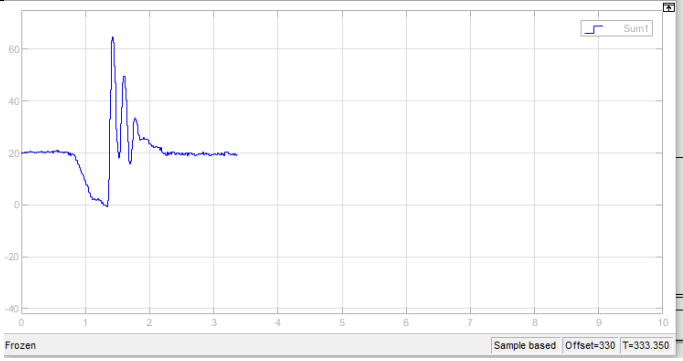
Appendices

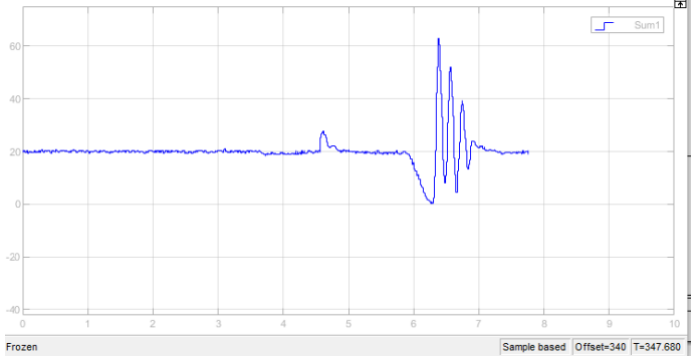
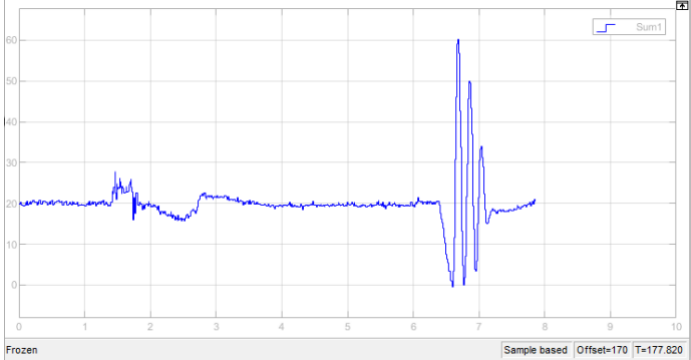
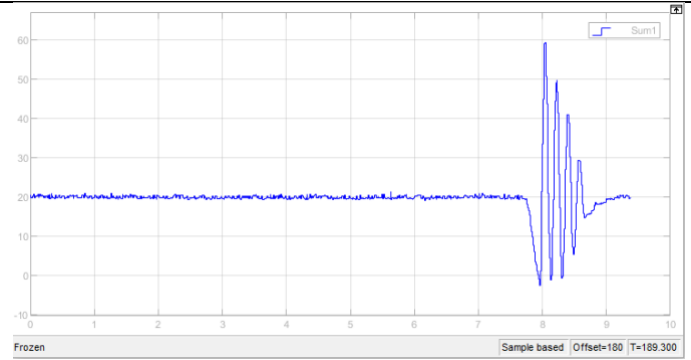
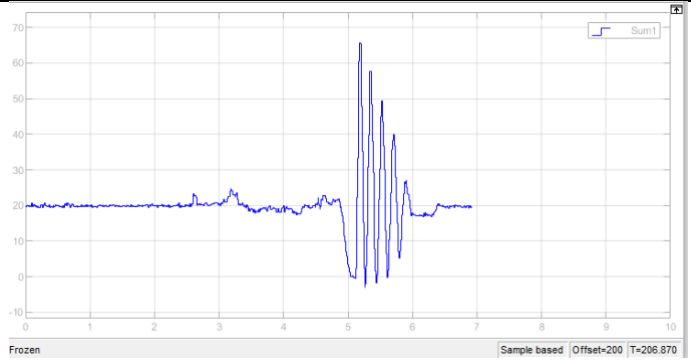
K_p Value	K_2	Response
0.75	0.0	No PI control present
0.75	0.1	Very slow adjustment to no steady state error
0.75	0.2	Very slow adjustment to no steady state error
0.75	0.3	It took 91 seconds to go from 0 degrees (voltage 0)

		to our set angle of 25 degrees
0.75	0.4	It took 68 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	0.5	It took 52 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	0.75	It took 35 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	1	It took 25 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	2	It took 13 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	3	It took 11 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	4	It took 11 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	5	It took 8 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees
0.75	10	It took 4 seconds to go from 0 degrees (voltage 0) to our set angle of 25 degrees. However it overshoot 25 degrees once
0.75	15	This is too high as it overshoot twice before being stable at the desired 25

		degrees.
--	--	----------

Table 1

K_p	K_1	K_D	Response To a disturbance pushing it to 0 deg	Comment
0.75	10	1		
0.75	10	0.75		Best value

0.75	10	0.5		
0.75	10	0.25		
0.75	10	0.1		
0.75	10	0.05		

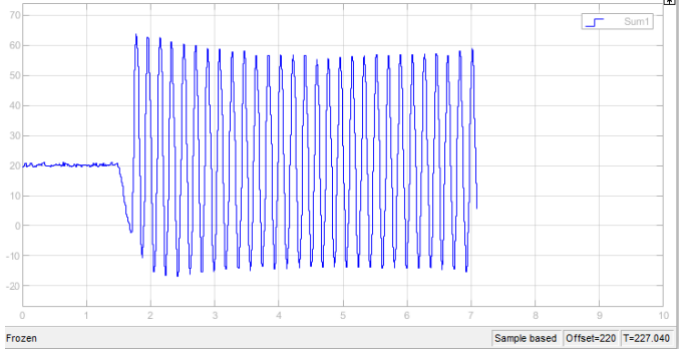
0.75	10	0.01		
------	----	------	--	--

Table 2