Experiment C3
PID Temperature Controller
Procedure

Deliverables: Checked lab notebook, Technical Memo

Overview

In this lab, you will use a special feedback method known as Proportional Integral Derivative (PID) control. PID is a common method for controlling temperature or other system parameters. For PID temperature control, the heater power is continuously adjusted using feedback from a temperature sensor. The algorithm sets the power using the formula

$$\dot{q} = k_p(T_S - T) + k_I \int (T_S - T)\,dt + k_D \frac{d}{dt}(T_S - T)\,, \qquad (1)$$

where T is the measured temperature, T_S is the desired temperature (or set point), and k_p, k_I, and k_D are the proportional, integral, and derivative "gains", respectively. The parameters T_S, k_p, k_I, and k_D are set by the user.

In this lab, you will learn how to use "pulse width modulation" to digitally control power. Then, you will learn how to implement a PID algorithm in LabView. Lastly, you will explore how the proportional, integral, and derivative gains change the behavior of the controller, and how to choose optimal values for these parameters.

Part I: Pulse Width Modulation

Background

Pulse width modulation (PWM) is a common technique for controlling analog electronics with a digital square wave. Shown in Fig. 1, a digital square wave is used to open and close the gate of a MOSFET transistor, which rapidly switches the source-drain current ON and OFF. Changing the width of the pulses or "duty cycle" changes the *average* amount of power delivered to a heater or motor.

Experimental Procedure

1. Use the alcohol thermometer to measure the temperature of the air in the lab T_{Air}. Record the value in your lab notebook.
2. Sketch the circuit and pin-out shown in Fig. 1 in your lab notebook.
3. Construct the thermistor voltage divider circuit from C1 **near the top** of the breadboard.
4. Connect the output of the voltage divider *Vs* to the analog input on the analog input of the USB-6341 and use the LabView program from C1 to measure the temperature vs. time. Make sure it is working before you move on to the next step.

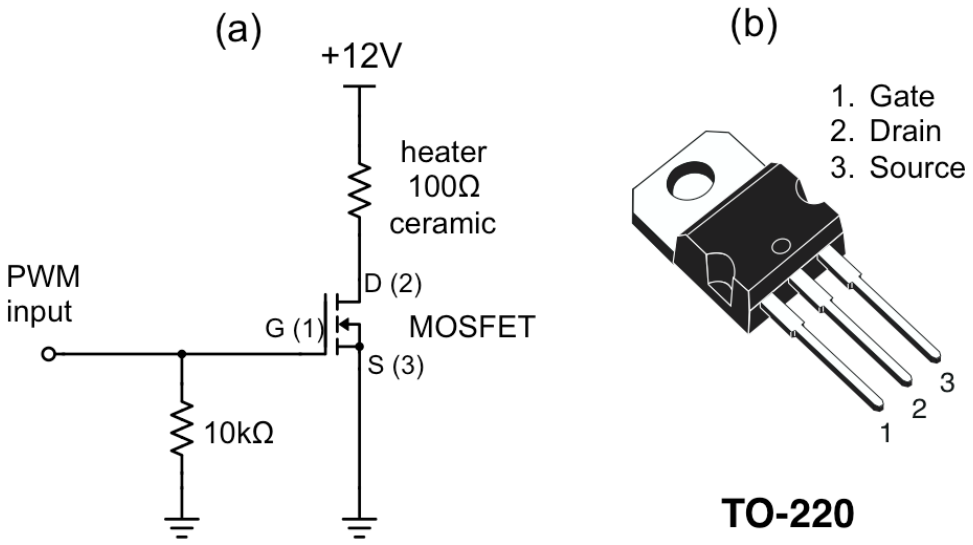

Figure 1 – (a) An N-Channel MOSFET allows a low power digital signal to control a large amount of current passing through a resistive heater. (b) The pin-out for the TO-220 power MOSFET.

5. Sketch the MOSFET PWM circuit shown in Fig. 1a.
6. Sketch the MOSFET pin-out shown in Fig. 1b.
7. Construct the MOSFET PWM circuit shown in Figure 1, such that the 100Ω heater is in contact with the thermistor. Bend the thermistor so it touches the resistor, and inject a small dab of thermal paste between the two.
8. Test the circuit. Use a handheld DMM to measure the voltage across the heater.
    a. Initially, there should be 0V across the heater.
    b. Connect the MOSFET gate ("PWM input" in Fig. 1) to the 5V power supply. The heater should turn on with ~12V across it, and the temperature should start to increase.
    c. Disconnect the 5V from the gate. The heater voltage should go back to 0V, and the temperature should decrease.
9. Using the BNC T, connect the output of the function generator to CH1 of the oscilloscope.
10. Set up the function generator for "Pulse" function with a frequency of 500 Hz, a high value of 5V and low value of 0V.
11. Using the BNC T, connect the output of the function generator to the MOSFET gate.
12. Turn on the output from the function generator. Vary the duty cycle or pulse width on the function generator, and observe how it affects the heater power and temperature.
13. In your notebook, record the equilibrium temperature as a function of the PWM duty cycle. You should measure 6 data points from 0 – 100% duty. At this point, you could set the temperature using data from this table, interpolating between values. This is known as a "feed forward, open loop" control system. However, small changes in the surrounding environment usually make this approach innacurate.

14. Disconnect the function generator and turn it off.

Part II: Generating PWM Signal with LabView

Experimental Procedure

1. Create a new LabView VI, give it an intelligent file name, and save it in your C3 folder.
2. Create a while loop with a stop button.
3. Right-click inside the while loop. Under "Express" > "Input", select the "Simulate Signal" express VI and place it in the while loop.
4. In the Configure window, make a 500 Hz square wave. Under timing, set it to 50000 samples per second. Make sure "Automatic" and "Run as fast as possible" are activated, and press OK.
5. Create a control for the Duty Cycle. Right click the duty cycle input on the "simulate signal" VI, then go to "Create" > "Control".
6. Wire up a constant value of 2.5 volts for both the amplitude *and* the offset. This will make it so the low value is always 0V and the high value is always 5V. (LabView uses peak-to-peak amplitude.)
7. Right-click inside the while loop. Under "Express" > "Output", select the "DAQ Assist" express VI and place it in the while loop.
8. In the pop-up window, select "Generate Signal" > "Analog Output" > "Voltage" > "ao0" > "Finish".
9. A subsequent window will appear with a sine wave. Set the "Generation Mode" to "Continuous Samples", then click OK.

NOTE: We are essentially using a digital-to-analog converter to simulate a digital pulse train on an analog output. This is a very inefficient way to make a PWM signal, and we are only doing it this way for educational purposes. In the "real world", PWM output should be kept all digital, which is what we will do next week with the Arduino microcontroller.

10. Add jumper wires to the corresponding analog output (AO 0) and ground connections on the USB-6341.
11. Connect the output of "Simulate Signal" to the input of the "DAQ Assist".
12. Test the program in the Front Panel. Press run, and try adjusting the duty cycle. Check the output on the oscilloscope to make sure it works. Make sure the low value is always 0V and the high value is always 5V.
13. Connect the PWM output from the USB-6341 as you did in Part I. Adjust the duty cycle in LabView and observe how it affects the heater voltage.
14. Save your VI.

Part III: Proportional Control with LabView

You will now combine your C1 temperature measurement VI with the PWM circuit and VI to create a PID controller. The measured temperature will be used to adjust the PWM output *a la* Eq. (1).

Experimental Procedure

1. Create a copy of your C1 thermistor VI, give it an intelligent file name (i.e. "C3_proportional_feedback_yourName.vi"), and save it in the C3 folder.
2. Right click inside the while loop and select "Programming" > "Structures" > "Flat Sequence". Draw a box around all of the temperature measurement, formulae, temperature array, and Build XY graph stuff, but NOT the timing code and stop button at the bottom.
3. Right click the edge of the sequence and select "Replace" > "Replace with Stacked Sequence".
4. Right click the edge of the sequence and select "Add Sequence Local". This will create a local variable for you to store the measured temperature. Connect the sequence local to the *temperature* scalar wire (thin orange wire).
5. Right click the edge of the sequence and select "Add Frame After".
6. Copy (ctrl+c) everything inside the while loop from Part II that you created to generatre the PWM signal. Paste (ctrl+v) it into the empty "frame" you just created.
7. Delete the control for Duty Cycle. Instead, you will use **proportional feedback** to adjust the PWM % duty cycle.
8. Hit control+E to toggle to the front panel window. Right-click to create controls for the temperature set point T_S proportional gain k_p. Arrange them nicely on the front panel, and give them appropriate names and units.
9. Double click the gray edge of the newly created control. This should take you back to the block diagram. Place the newly created controls in frame 1 of the stacked sequence.
10. Use the numeric functions palette to implement the proportional feedback (i.e. the first term on the right hand side of Eq. (1)). The measured temperature can be read in the sequence local variable.

NOTE: The heater *power* is not directly proportional to the % duty cycle. You will need to use the formula from problem 1 in the pre-lab assignment to correctly scale the PWM signal.

11. Use a "Case Structure" so the program sets the duty cycle to 0% if the actual temperature is greater than the set point. ("Simulate Signal" does not like negative values of duty cycle.)
12. Create an indicator to display the duty cycle on the front panel.
13. Test the program with a set-point $T_S = 315$ K to make sure it works.
14. When you are convinced the program works, record test data for several different value of proportional gain, starting with $k_p = 0.1$ W/K. Keep the set-point constant at $T_S = 315$ K, test it with at least three different values of of k_p, and **save the data**. How does the proportional gain k_p change the behavior of the controller? How does this compare with your performance prediction in the pre-lab assignment?

Part IV: Implementing the PID Controller

Experimental Procedure

1. Create a copy of your code from Part III, give it an intelligent file name (i.e. "C3_PID_yourName.vi"), and save it in your C3 folder.
2. Use shift registers and a Riemann sum formula to compute the integral of the temperature error $T_S - T$ in the frame next to the proportional feedback. Add an indicator to display the value in the front panel.
3. Use a conditional "case structure" to set the integral to zero if $(T_S - T) > 5$ Kelvin. If $(T_S - T) < 5$ Kelvin, then calculate the integral using a Riemann sum. This will prevent the integral from becoming too large in the beginning when the temperature is still low.
4. Use shift registers and a finite difference formula to compute the derivative of the temperature. Add an indicator to display the value in the front panel.
5. Add controls to the front panel for the integral gain k_I and derivative gain k_D.
6. Add the integral and derivative feedback to the proportional feedback to create a full PID controller.
7. Use a fixed set-point $T_S = 315$ K, and test the controller starting with $k_I = k_D = 0$. Then, test it again with a small value of $k_I < 0.1$ W/Ks and $k_D = 0$. Test it again with a larger value of $k_I > 0.2$ W/Ks and $k_D = 0$. You should see that large values of k_I also lead to oscillations. **Save the data**.
8. Test it again with the same large value of $k_I > 0.2$ W/Ks, but with $k_D > 10$ Ws/K. You should see that the derivative feedback dampens the oscillations. **Save the data**.

Part V: Tuning the PID Controller

As you can probably see, it is not obvious what values should be used for the gains k_p, k_I, and k_D. You will now use a heuristic method developed by **Ziegler and Nichols** to "tune" the controller to the optimal values of k_p, k_I, and k_D. By "optimal", we mean fast response with minimal oscillations.

Experimental Procedure

1. Recall, that for small values of k_p, the controller never reaches the set point, but for large values of k_p, it will overshoot and oscillate. Using a fixed set-point $T_S = 315$ K and $k_I = k_D = 0$, experiment with different values of k_p and find the critical value of k_p that causes the controller to overshoot the set point and oscillate. Denote it as the "critical gain" k_u.
2. Measure the period of the oscillations T_u and write it in your lab notebook. Be mindful of units.
3. For a proportional-integral (PI) controller with $k_D = 0$, Ziegler-Nichols says the optimal gains are $k_p = 0.45k_u$ and $k_I = 0.54k_u/T_u$. Calculate these values and write them down in your lab notebook.
4. Test the PI controller with $k_p = 0.45k_u$, $k_I = 0.54k_u/T_u$, and $k_D = 0$. Save the data.

5. For a full PID controller, Ziegler-Nichols says the optimal gains are $k_p = 0.6k_u$, $k_I = 1.2k_u/T_u$, and $k_D = 3k_uT_u/40$ Calculate these values and write them down in your lab notebook.
6. Test the PID controller with $k_p = 0.6k_u$, $k_I = 1.2k_u/T_u$, and $k_D = 3k_uT_u/40$. Save the data. How does the PID controller compare with the PI controller? What effect does the derivative term have?
7. Leave the MOSFET and thermistor circuits intact on the breadboard and disassemble everything else. You will use it again next week. Carefully store everything in the large tote and place it in the cabinet above your lab bench.

Data Analysis and Deliverables

Using LaTeX or MS Word, make the following items and give them concise, intelligent captions. Make sure the axes are clearly labeled with units. Plots with multiple data sets on them should have a legend. **Additionally, write several paragraphs describing the plots/tables. Any relevant equations should go in these paragraphs.**

IMPORTANT NOTE: Check the units of your gains k_p, k_I, and k_D. For example, the proportional gain k_p in your code likely has units of duty_cycle/Kelvin. However, in the pre-lab assignment it has units of Watts/Kelvin. You will need to use some factor of V^2/R to convert from duty cycle to average power in Watts if you did not all ready do so in your code. (See the C3 pre-lab assignment.)

1. From Part III, a plot of the temperature vs. time for at least 3 different values of k_p (all on the same plot with a legend). **Add a horizontal line denoting the set-point.** Compare this with your predictions from the pre-lab assignment.
2. From Part IV, plot some of the temperature vs. time traces (all on the same graph) for different values of the gains. Add a horizontal line denoting the set-point. Be sure to include a legend denoting the gains. Describe the behavior.
3. From Part V, a plot of the temperature vs. time traces (both on the same graph) for the tuned PI and PID controllers. Add a horizontal line denoting the set-point.
4. From Part V, a table containing the optimal gains for the PI and PID controller that you determined using the Ziegler-Nichols tuning method.

Talking Points – Discuss these in your paragraphs.

- Include the important equations you derived in the Pre-lab Assignment.
- Qualitatively discuss the response time of the proportional controller as a function of the proportional gain k_p.
- Does your proportional controller from Part III oscillate? Is this predicted by the differential equation you derived in the Pre-lab? Explain.
- In Part IV, you selected the gains using a "guess-and-check" method. Qualitatively compare the performance using the guessed gains in Part IV with the performance using gains obtained using the Ziegler-Nichols tuning in Part V.

Appendix A

Equipment

- USB-6341 DAQ
- PC computer with LabView 2017
- Bread board
- 2 BNC cables
- BNC to mini-grabber adapter
- Extech handheld DMM
- Vishay NTCLE100E3103JB0, NTC Thermistor 10k Bead (Digikey part # BC2301-ND)
- 4.7kΩ resistor
- N-Channel MOSFET TO-220AB (Digi-key part #: 497-2765-5-ND)