Experiment C3
PID Temperature Controller
Procedure

Deliverables: Checked lab notebook, Technical Memo

Overview

In this lab, you will create a digital PID temperature controller using an Arduino UNO microcontroller. The Arduino and other microcontrollers have several advantages: they are small, portable, and inexpensive. This lab will teach you how to program an Arduino microcontroller. You will also get a deeper look at PID feedback control.

Part I: Pulse Width Modulation (PWM)

Background

The Arduino microcontroller can output several PWM signals using the analogWrite() command. The command takes an output pin number and an 8-bit integer between 0 – 255 that determines the duty cycle of the PWM (0 yields 0% and 255 yields 100% duty cycle).

Experimental Procedure

1. Connect the Arduino with the USB cable and open the Arduino software on the lab computer.
2. Go to "File" > "Examples" > "01.Basics" > "Fade" to pull up the "sketch". (Arduino calls its code files "sketches".) Look through the code and read the comments. Internalize the basic structure of the code: variable declaration, then hardware setup, followed by the main loop.
3. The fade program uses PWM output to power an LED. Go to the link in the comments (https://www.arduino.cc/en/Tutorial/Fade) and follow the instructions in the tutorial. Build the circuit on the small portable breadboard. Connect the battery pack to the + and – bus lines on the breadboard. Connect the analog ground "GND" and "5V" on the "power side" of the Arduino to the + and – bus lines on the breadboard.
4. Sketch a schematic of the circuit in your lab notebook.
5. Use the BNC to mini-grabber cable to connect the Arduino's PWM output to CH1 of the oscilloscope and turn on the scope.
6. In the Arduino software, go to "Tools" > "Port" and select the COM port that says "(Arduino/Genuino Uno)" next to it.
7. Click the check mark at the top of the Arduino program to check the code for errors.
8. Press the arrow button to compile the program and send it to the Arduino. After it compiles you should see the LED start blinking. If the program throws an error, ask the lab instructor for help.

9. Examine the PWM signal on the oscilloscope. Does it make sense? Record its amplitude and frequency in your lab notebook.
10. Change the millisecond value in the in the delay() command in the main loop, and recompile the program. How does this affect the system?
11. Disconnect USB cable from the Arduino microcontroller and the mini-grabber from the circuit. You should see that it is still running, and you can pick it up and move it around the room. It is compact and portable!
12. Press the "Reset" button on the Arduino. It should stop for a moment, then start right back up again. This is an important point about microcontrollers: **they always run in an infinite loop**, and the only way to stop it is to physically disconnect the power source.
13. The only way to stop the microcontroller is to physically disconnect the power source. Modify the circuit so it has a "kill switch". (Use the toggle switch in your kit.)
14. Pick up your compact and portable system and walk around the lab with it. Present it to the lab instructor.
15. Disconnect the batteries to stop the Arduino. Remove the LED and resistor from the breadboard and return them to their proper place. (The remainder of the lab will be use the normal breadboard with its convenient DC power supplies.)

Part II: Temperature Sensor

Background

The Arduino microcontroller can read analog voltage signals using the analogRead() command. This command takes an analog voltage between 0 – 5V and converts it to a 10 bit digital number between 0 – 1023 (0 maps to 0V and 1023 maps to 5V). In this portion of the lab, you will use the Arduino to measure the voltage from thermistor circuit and calculate the temperature using the voltage divider and Steinhart equations, as you did in C1.

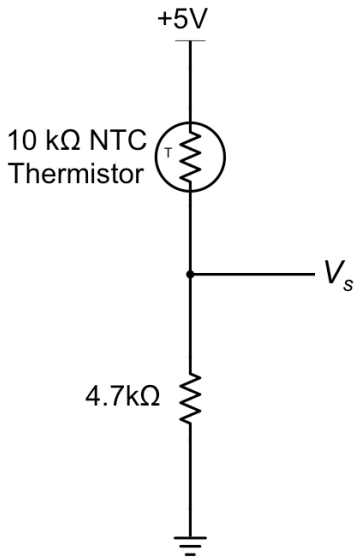

Figure 1 – A thermistor is wired up in a voltage divider circuit. Note that the voltage divider is now driven by 5V instead of 12V.

Procedure

1. Construct the thermistor voltage divider circuit near the top of the big breadboard (not the portable one). The Arduino does not like voltages above 5V, so you will use the **5V DC power supply on the Arduino for V_{in}** on the voltage divider, as shown in Fig. 1.
2. Connect the "GND" pin on the Arduino to the large breadboard ground.
3. The MOSFET heater circuit should still be driven by +12V.
4. Make a copy of the C1 LabView VI, give it an intelligent file name, and save it in the C3 folder. You will use this LabView program to observe the temperature in real time.
5. Connect the USB-6341 to the output of the thermistor circuit. Update the code to reflect that V_{in} on the voltage divider was reduced from 12V to 5V.
6. Test the thermistor circuit and make sure LabView displays the correct temperature.
7. Update the LabView code to convert the time from milliseconds to seconds. (The plot should display the time in seconds, and the data should be saved with time in seconds.)
8. Update the LabView code so that V_{in} in the thermistor voltage divider equation is decreased from 12 to 5V.
9. Connect the Arduino with the USB cable to the lab computer and open the Arduino software.
10. Connect the Arduino analog input pin A0 to the output of the thermistor circuit.

Pro-tip: Be thoughtful when you choose colored wires. It is extremely difficult to debug a circuit when every wire is red.

11. Download the C3 code template from the lab webpage. Read the comments and fill in the missing values for the variables denoted with "***". Save the sketch to your C3 folder with an intelligent file name.
12. Click the check mark at the top of the Arduino program to check the code for errors.
13. Press the arrow button to compile the program and send it to the Arduino.
14. Go to "Tools" > "Serial Monitor" (or press "Ctrl + Shift + M") to view the output from the "Serial.print()" commands at the bottom of the screen. You should see the measured temperature printed. Does it agree with your LabView program?
15. Disconnect the USB cable and "Vin" power 5V connection to turn off the Arduino.

Part III: MOSFET Circuit

You will now use the PWM output to control the heater power. The PWM signal from the Arduino will be amplified by the MOSFET circuit shown in Fig. 2.

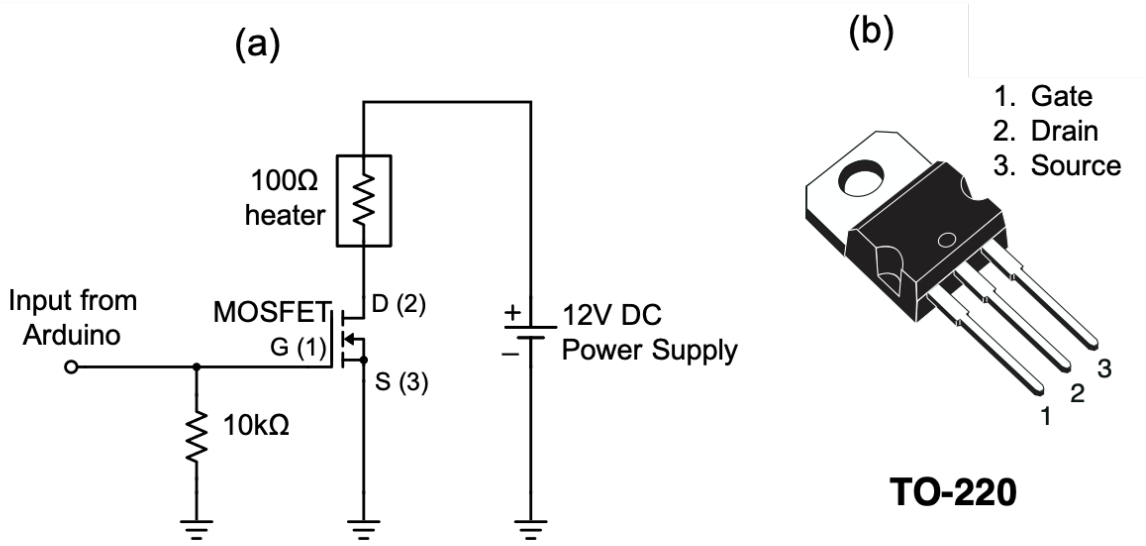

Figure 2 – An N-channel MOSFET is used to modulate the heater power. A PWM signal from the Arduino is used to drive the gate of the MOSFET and modulate the flow of current through the heater.

1. Construct the MOSFET PWM heater circuit shown in Fig. 2, 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.
2. Create a copy of the "Fade" sketch from Part I, give it a new file name, and save it in the C3 folder. Modify the delay in the main loop to be 300 ms.
3. Connect the Arduino PWM signal to the gate of the MOSFET.
4. Add the **kill switch** to the circuit, so you can manually cut power to the heater when it is not in use. We recommend you use it to cut the 12V power supply from the 100Ω heater.
5. Use the oscilloscope to monitor the PWM signal, and the LabView VI to record the temperature vs. time.
6. Compile the code and send it to the Arduino. You should see the temperature periodically rise and fall. After about 3 cycles, stop the LabView program and save the data.

Part IV: System Identification and Characterization

As you did in C2, you will measure T_{Air}, T_{max}, τ, and q and use them to calculate the system parameters mc_p and Ah.

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. Create a copy of your thermistor sketch from Part II, give it a new file name "C3_TEST_yourName", and save it to your C3 folder.
3. Modify the sketch to have PWM output. That is, declare pin 9 to be an output in the "setup()".
4. In the main loop, use the "analogWrite()" command to generate a PWM signal using a duty cycle of 255.

IMPORTANT NOTE: AnalogWrite() takes integers between 0 and 255. Stranger things will happen if you give it values outside this range.

5. Modify the serial.print() commands to display the values of the temperature and PWM strength.
6. Start the LabView program to record the temperature vs. time. Compile and run the Arduino code.
7. Measure the voltage across the heater and record the value in your lab notebook. You will use this to calculate the heater power q_{max}.
8. With LabView recording the data, let the temperature rise, and wait for it to get to a steady state maximum T_{max}.
9. Record the steady state maximum temperature T_{max} in your lab notebook.
10. With the LabView program still running, disconnect the 12V power to the heater. Watch the temperature drop.
11. Use the temperature vs. time data to determine the time constant τ, and calculate S, mc_p, and Ah, as you did in C2.

Part V: Proportional Feedback Controller

The strength of the PWM signal (or heater power) will be calculated using proportional feedback from the thermistor.

1. Create a copy of your thermistor sketch from Part IV, give it a new file name "C3_proportional_feedback_yourName", and save it to the C3 folder.
2. Declare new float variables for the temperature set-point T_S and proportional gain k_p.
3. Make an initial estimate of the proportional gain. It should be approximately the max heater power divided by the max temperature difference $k_p = q_{max}/|T - T_S|$. Write the values down in your lab notebook.

4. In the main loop, use the "analogWrite()" command to generate a PWM signal calculated using proportional feedback. Note that k_p will have units of W/K, and you will have to convert the calculated in Watts to an 8 bit duty cycle between 0 and 255 using the conversion factor $255/q_{max}$.

IMPORTANT NOTE: AnalogWrite() takes integers between 0 and 255. You should include an if-else statement that overrides your $k_p(T_S - T)$ calculation and sets the PWM to a max of 255 and min of 0, depending on the value of T. (https://www.arduino.cc/reference/en/language/structure/control-structure/else/)

5. Modify the serial.print() commands to display the values of the temperature and PWM strength.
6. Use a fixed set-point $T_S = 315$ K, and test the program. Use LabView to record the temperature vs. time and the Arduino serial monitor to observe the system parameters. In between tests, you should disconnect the PWM signal from the MOSFET gate to turn off the heater and allow it to cool down.
7. When you are confident that the proportional feedback works, demonstrate it to the lab instructor.
8. Test the system for at least 3 different values of k_p, and use LabView to record the temperature vs. time traces. Turn off the heater and allow the sample to cool in between tests. Record and save the data.
    a. Start with the initial estimate for k_p you just made.
    b. Try a value that is much lower than the initial estimate.
    c. Try a value that is much greater than the initial estimate.
    d. Is the behavior consistent with what you predicted in the pre-lab assignment?
9. Turn OFF the heater and allow it to cool down.

Part VI: PID Controller

Background

The strength of the PWM signal (or heater power) will be calculated using PID feedback from the thermistor.

1. Create a copy of your sketch from Part V, give it a new file name "C3_PID_yourName.ino", and save it to your C3 folder.
2. Declare new float variables for the integral and derivative and their respective gains. The integral should be initialized to zero.
3. In the main loop, use a Riemann sum to compute the integral of the error. (Be careful with the units of Δt.) Put the calculation in a conditional "if{}" statement. If $(T_S - T) > 5$ Kelvin, the integral should be set to zero. Else if $(T_S - T) < 5$ Kelvin, then calculate the integral using a Riemann sum.
4. Use a finite difference to compute the derivative.
5. Use the "analogWrite()" command to generate a PWM signal calculated using PID

feedback.

6. Modify the serial.print() commands to display the values of the temperature, integral, derivative, and PWM strength.
7. Thoroughly, test the program. Start with only proportional feedback (i.e. $k_I = k_D = 0$). Use LabView to view the temperature vs. time. In between tests, use the kill switch to turn off the heater and allow it to cool down a little bit.
8. When you are confident that it works with only proportional feedback, test it with the integral and derivative feedback. That is, test it with non-zero values for the integral and derivative gains.
9. When you are confident that the full PID controller work, demonstrate it to the lab instructor.
10. Choose a small value of k_p, such that the system will not oscillate. Find a value of the *integral* gain where system goes into significant oscillations (i.e. $k_I = 0.001$ W/Ks, then 0.005 W/Ks, etc.). Save the temperature vs. time data measured in LabView.
11. Using the values of k_p and k_I from the previous step, increase the derivative gain (i.e. $k_d = 0.1$Ws/K, then 1Ws/K, 10Ws/K, …). This should dampen the oscillations. Record at least two data sets showing how increasing the derivative gain k_d dampens the oscillations.

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. **Add a horizontal dashed line denoting the set-point whenever it is applicable.**

1. A table containing the relevant values you measured in Part IV: temperatures T_{Air} and T_{max}, heater power q, and the time constant τ. Then, use the equations you derived in the C2 pre-lab assignment to determine values mc_p and Ah, and include them in the table, as well. Be careful with units
2. From Part V, a plot of the temperature vs. time traces for at least 3 different values of k_p (all on the same plot with a legend). Use the values you determined for T_{Air}, S, mc_p, and Ah to *quantitatively* compare your results to your predictions from the pre-lab assignment.
3. From Part VI, a plot of the temperature vs. time traces for at least 2 different values of k_d (all on the same plot with a legend). Compare this with your predictions from this week's pre-lab assignment.

Talking Points – Discuss these in your paragraphs.

- Include a few of 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.
- How does the derivative gain k_D affect the amplitude of the oscillations? Use the equations you derived in the pre-lab to explain the behavior.

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)