Experiment C2
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.
4. Wire up the vertical bus lines on the small breadboard:
    a. Connect the battery pack to the bus lines on the left side of the breadboard. Then, connect the analog ground "GND" and "5V" on the "power side" of the Arduino to same bus lines on the breadboard.
    b. Connect the bus lines on the right side of the breadboard to the 12V DC power supply (looks like a laptop charger).
    c. Use a black jumper wire to connect both the left and right **ground** bus lines establish a common ground.
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. 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.
8. Examine the PWM signal on the oscilloscope. Does it make sense? Record its amplitude and frequency in your lab notebook.
9. 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.
10. Disconnect the batteries to stop the Arduino. Remove the LED and resistor from the breadboard and return them to their proper place.

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.

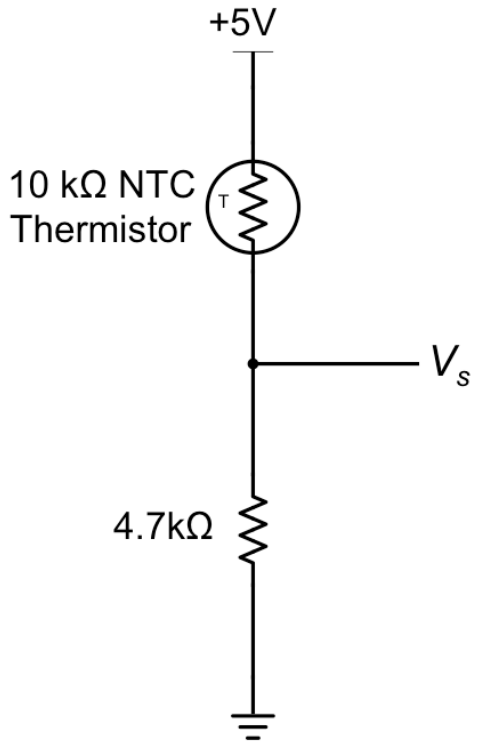

Figure 1 – A thermistor is wired up in a voltage divider circuit. Note that the voltage divider is now driven by 5V instead of 12V.

Procedure

1. Construct the thermistor voltage divider circuit near the top of the small breadboard, as you did in previous lab. The Arduino does not like voltages above 5V, so you will use the **5V bus line for V_{in}** on the voltage divider (NOT 12V), as shown in Fig. 1.
2. Connect the "GND" pin on the Arduino to the large breadboard ground.
3. Make a copy of the C1 LabView VI, give it an intelligent file name, and save it in the C2 folder. You will use this LabView program to observe the temperature in real time.
4. Connect the USB-6341 to the output of the thermistor circuit, as you did in the previous lab.
5. Update the LabView code so that V_{in} in the thermistor voltage divider equation expression node is decreased from 12 to 5V.
6. Test the thermistor circuit and make sure LabView displays the correct temperature. Briefly connect the red heater wires to the 12V bus line, then disconnect after about 10 sec. You should see the temperature go up a bit.
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.)
   a. Right click the appropriate wire in the block diagram, select "insert", and choose the division operator.
   b. Right click the bottom input terminal of the division operator, select "constant", and enter the appropriate conversion factor.
8. Connect the Arduino with the USB cable to the lab computer and open the Arduino software.
9. 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.
10. Download the C2 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 C2 folder with an intelligent file name.
11. Click the check mark at the top of the Arduino program to check the code for errors.
12. Press the arrow button to compile the program and send it to the Arduino.
13. 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?
14. 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.

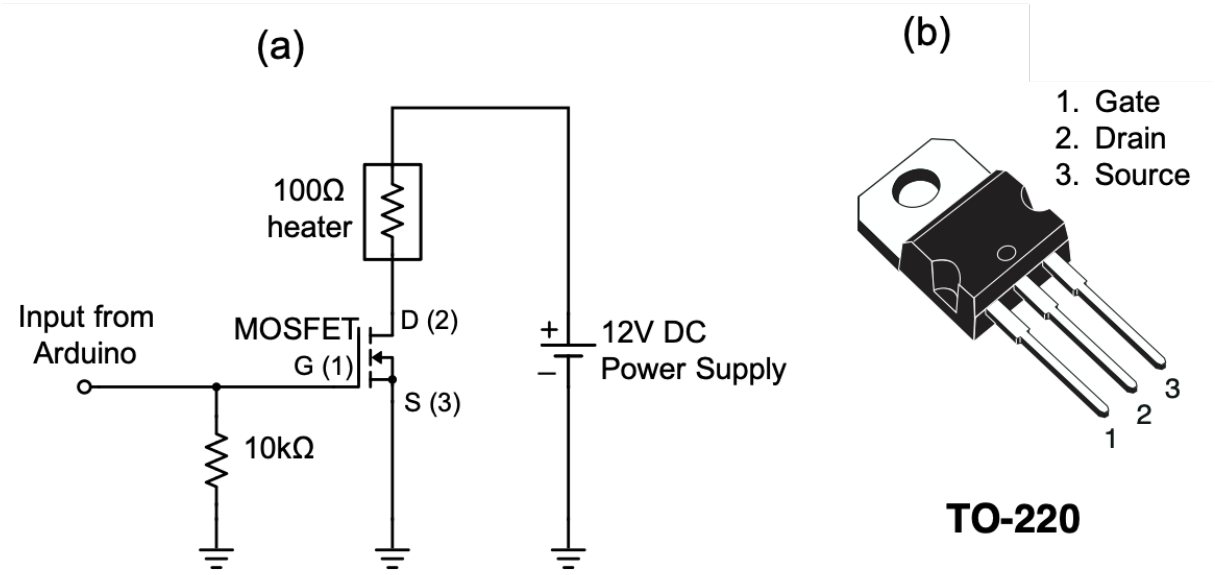

Figure 2 – An N-channel MOSFET is used to modulate the heater power. A PWM signal from the Arduino is used to drive the gate of the MOSFET and modulate the flow of current through the heater.

1. Construct the MOSFET PWM heater circuit shown in Fig. 2.
2. Create a copy of the "Fade" sketch from Part I, give it a new file name, and save it in the C2 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. Also, connect the gate of the MOSFET to ground via a 10k "pull down" resistor. (The pull down resistor will dissipate any static charge build-up on the gate, preventing it from randomly switching ON.)
5. 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 place it in series between the +12V bus line and heater.
6. Use the oscilloscope to monitor the PWM signal, and the LabView VI to record the temperature vs. time.
7. 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,

Part IV: 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 "C2_proportional_feedback_yourName", and save it to the C2 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 a tenth of the max heater power divided by the max temperature difference $k_p = q_{max}/10|T - T_{max}|$. Write the value 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. Wire up the fan to the 12V bus lines, and place the sample in the duct, as you did in the previous lab.
7. 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.
8. When you are confident that the proportional feedback works, demonstrate it to the lab instructor.
9. 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 increasingly larger values of k_p until you get significant overshoot and oscillations.
    c. Estimate the period of the oscillations T_u and write it in your lab notebook. Be mindful of units.
    d. Save your three data sets with intelligent file names. Write down the file names with their corresponding gains in your lab notebook.
10. Turn OFF the heater and allow it to cool down.

Part V: 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 "C2_PID_yourName.ino", and save it to your C2 folder.
2. Declare new float variables for the integral "I" and derivative "dTdt" and their respective gains. The integral should be initialized to zero.
3. Use a finite difference $\Delta T/\Delta t$ to compute the derivative.
   a. Use the millis() function to get time stamp in milliseconds at the beginning of the loop. Convert it to seconds by multiplying by 0.001, *t* = millis()*0.001;
   b. Declare new float variables for the time stamp and temperature from the previous iteration, "tprev" and "tempPrev".
   c. Compute the finite differences: $\Delta T = temp - tempPrev$, $\Delta t = t - tprev$, and $\Delta T/\Delta t$.
   d. At the end of the loop, set "tprev" and "tempPrev" equal to the current time and temperature.
4. In the main loop, use a Riemann sum to compute the integral of the error:
   $Integral = Integral + (temp - tempSet) \cdot \Delta t$.
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.

a. In between tests, use the kill switch to turn off the heater and allow it to cool down a little bit.

b. Focus your engineering awareness on the values of the integral and derivative printed in the serial monitor to check that they seem reasonable.

c. Press the reset button on the Arduino at the beginning of each test to clear the integral term.

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. If you are confident that the full PID controller is functioning properly, then it is time to use the **Ziegler-Nichols Method** 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.
10. 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.

11. Test the PI controller with $k_p = 0.45k_u$, $k_I = 0.54k_u/T_u$, and $k_D = 0$. Save the data.
12. 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.
13. 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?

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. From Part IV, a plot of the temperature vs. time traces for at least 3 different values of k_p.
2. From Part V, make a plot of the temperature vs. time traces for the Zeigler-Nichols tuned PI controller and PID controller.

Talking Points – Discuss these in your paragraphs.

- Include a few of the important equations you derived in the Pre-lab Assignment. In particular, use the time constant you measured in the previous lab to *quantitatively* compare your results to your prediction from the pre-lab assignment. How does the proportional gain k_p affect the time constant?
- How does the derivative gain k_d affect the amplitude of the oscillations?

Appendix A

Equipment

- USB-6341 DAQ
- PC computer with LabView
- Bread board
- 2 BNC cables
- BNC to mini-grabber adapter
- Extech handheld DMM
- Blower fan mounted to small plywood sheet – Amazon Part # B08P1S5DBN
- Breadboard mounted to small plywood sheet
- 2” × 2.5” × 1/16” aluminum sample w/ polyimide film heater and thermistor
    - Polyimide film heater – Amazon Part # B09X16XCVS
    - 10k NTC Thermistor, adhesive mount, Amphenol – Digikey Part # 235-1457-ND
- 4.7kΩ resistor
- N-Channel MOSFET TO-220AB (Digi-key part #: 497-2765-5-ND)