Flori Yellin
Advanced Physics Laboratory 39
Lab I: Thermal Control
March 3, 2010

Abstract

This experiment studies temperature control. We used the LabVIEW to write our Proportional- Integral- Differential (PID) control algorithm. We also used a Data Acquisition (DAQ) USB device and a PC. We acquired temperature data using a temperature sensor (thermistor), and we used a heat pump, or Thermoelectric Cooler (TEC), to heat and cool the system. A drive circuit sent a pulse width modulation (PWM) to the system to supply the power.

H-Bridge Amplifier

One piece of hardware that we used was a FTX300 H-bridge amplifier. An H-bridge amplifier circuit is shown in Diagram (1).

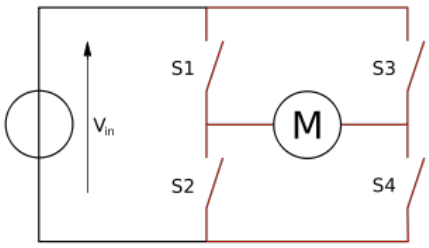


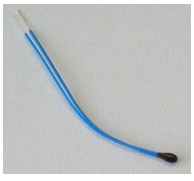
Diagram 1a: H-Bridge Amplifier

As shown in Diagram (1), the H-bridge consists of four switches. The advantage of an H-bridge is that it allows a voltage V_{in} to be applied across a load M in either direction. When switched S1 and S4 are closed (S2 and S3 are left open), there will be a positive voltage (towards the right) applied across M. When S2 and S3 are closed, the polarity is reversed, and a negative voltage (towards the left) is applied.

The H-Bridge that we used was an Accuthermo Technology FTX300 H-Bridge Amplifier. This device is useful for bi-directional thermal control because of its ability to switch voltage polarity and enable both heating and cooling.

Thermistor

A thermistor is a resistor with a temperature-dependent resistance. It can be used as a temperature sensor. The thermistor that we used was a negative temperature coefficient (NTC) thermistor, which means that the thermistor resistance decreases as the temperature increases. For small temperature ranges, the relationship between the change in resistance and change in temperature is approximately linear, but this linear relationship is limited to a narrow temperature range. A more accurate



description of the temperature-resistance relationship of a thermistor is given in the Steinhart-Hart Equation (Equation (1)).

$$(1)\quad 1/T = a + b\ln(R) + c\ln^3(R)$$

In Equation (1), R is the thermistor resistance, in ohms; T is the absolute temperature, in Kelvins; and a, b, and c are the Steinhart-Hart parameters that are specific to each thermistor. For the thermistor that we used, the Steinhart-Hart parameters were $a = 0.001129148$, $b = 0.000234125$, and $c = 8.76741 \times 10^{-8}$.

Another equation that relates temperature to resistance for thermistor is the B-parameter equation (Equation (2)). This equation can be derived from the Steinhart-Hart Equation when $c = 0$ and $B = 1/b$.

$$(2)\quad 1/T = 1/T_0 + 1/B\ln(R/R_0)$$

In Equation (2), T is the absolute temperature and R is the resistance in ohms. T_0 is the temperature when the resistance is R_0. For the EPCOS thermistor that we used, $R_0 = 2000$ Ohm, $T_0 = 298.15$ K, and $B = 3560$ K. Using these constants, we were able to create a LabVIEW program that converted the resistance across the thermistor into temperature measurements around the thermistor.

EMANT300 Data Acquisition System

The EMANT300 is a data acquisition (DAQ) USB. It connects to a computer and is used to communicate between the computer and the world (non-computer devices).

The data acquisition system has both analog inputs and digital outputs, and it is used for digital-to-along and analog-to-digital conversions. It is also used for interfacing inputs and outputs to the computer. For example, using the DAQ device, we were able to input data acquired from the thermistor onto the computer and to output digital signals, such as one to tell the Thermoelectric Cooler when to heat or cool.

The Emant300 DAQ device has six analog inputs, eight digital I/Os, an analog ground, two digital grounds, a counter input, a PWM output, two 5V Supply sources, two reference voltages, a common analog input, and an analog current output. We attached the DAQ to a breadboard, to which we connected the hardware using wires.

Pulse Width Modulation

We supplied power to our heating/cooling system by using pulse-width modulation (PWM). PWM is used to supply partial power to an electronic device. PWM modulates a rectangular pulse wave by controlling the percent of time that power is supplied. The duty cycle is the percent of the total pulse-period that the power is on. That is, the duty cycle measures the ratio of how long the pulse is “up” (supplying full power) to how long the pulse is “down” (supplying no power). The duty cycle is

measured as a percent (between zero and one hundred), with 100% corresponding to full power always on, and 0% corresponding to no power ever on.

Diagram (2) shows a square pulse. It appears from the diagram that the duty cycle is around 33%, since the pulse is "up" for about one third of its period. By using PWM, were able to control the temperature by modulating the amount of time that heating or cooling power was supplied to our TEC.

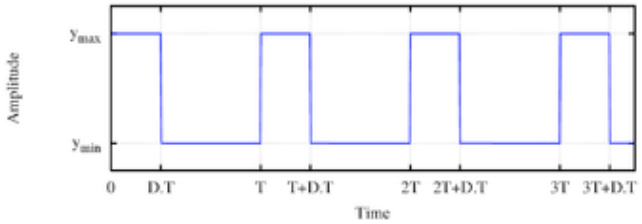


Diagram 2: Pulse Width Modulation Square Pulse

Enable/ Disable

We used the enable/disable wire on the H-bridge by writing a LabVIEW code that allowed us to turn the PWM on or off directly. That is, rather than setting the PWM to zero, we could "override" the PWM by disabling the circuit. The enable/disable capability as an added safety mechanism: we always disabled the circuit after running our code to ensure that the PWM would not continue to supply power to our circuit when our LabVIEW program was not running.

Safety Bi-Metal Switch

A thermal cut-out thermostat was attached to our Thermoelectric Cooler setup. This device was another safety mechanism to ensure that we did not accidentally overheat the system. The thermostat was set to automatically disable our circuit if the temperature exceeded seventy-Celsius degrees.

The safety bi-metal switch is a type of bimetallic thermostat. Bimetallic thermostats consist of two pieces of metal that are connected and act as a bridge in an electric circuit. When the temperature increases sufficiently, the metal expands, causing the bridged circuit to break. When the temperature cools sufficiently, the bridge re-forms, and the closed circuit turns on the heat once again. The safety switch that was installed in our circuit was set to disable the circuit before it could overheat.

Thermoelectric Cooler

To heat and cool our system, we used a thermoelectric device, a device that converts between temperature differences and electric voltage differences. When a temperature difference is applied across a thermoelectric device, it creates an electric voltage (Seebeck effect), and when there is a voltage difference across such a device, it will create a temperature difference (Peltier effect). The Thermoelectric Cooler (TEC) that we used is also known as a Peltier because it "pumps" heat using the Peltier effect.

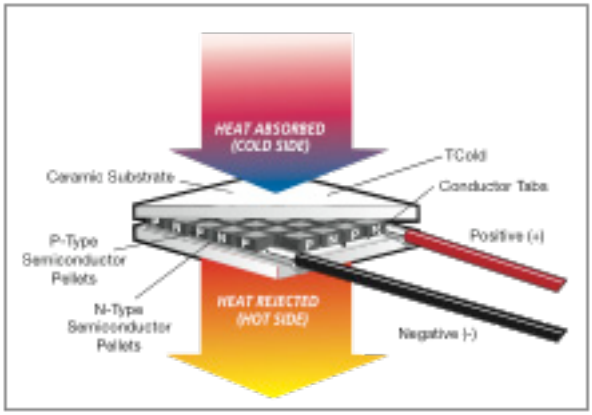


Diagram 3: Thermoelectric Cooler

Diagram (3) shows the heating and cooling process of the Thermoelectric Cooler. The thermoelectric cooler consists of two different semiconductor metals. They are doped so that they carry an excess of either electrons or holes (charge carriers). If the ends of a conductor are at different temperatures, the hot carriers will diffuse towards the cold and the cold carriers will diffuse towards the hot end.

Because the diffusing charge carriers are scattered by imperfections, the diffusion rates of the hot and cold carriers are not equal. This causes a potential difference and an electric voltage. The voltage forms an electric field, which causes the charges to flow in the opposite direction. Eventually, the number of carriers in one direction is equal to the number of carriers in the other direction, and equilibrium is reached.

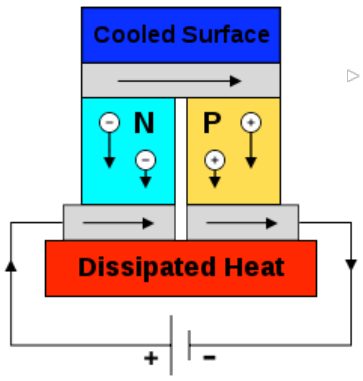


Diagram 4: Peltier Effect

Diagram (4) shows a thermoelectric device made of two connected pieces of semiconductors connected to a power source. The P-types move towards the right with the current, but the N-types oppose the current, moving towards the left. This effectively removes the heat from one side of the device.

The significance of a TEC is that when the DC voltage is applied to the device, the electrons and holes "pump" heat from one surface to the other. The side that absorbs heat becomes cold, and the side that rejects heat becomes hot.

Changing the polarity of the DC voltage source reverses the hot and cold sides. Meaning simply switching the positions of the black and red wires in Diagram (3), will switch the direction of heat flow, making the top surface hot and the bottom one cold. Because it is so easy to switch the heat flow direction in a thermoelectric cooler, TECs are convenient for thermal control systems, such as ours.

Water Cooling System

Our setup was attached to a Koolance Water Cooling System. The Water Cooling System used fans, a pump and water, and was attached to our setup by tubes. It uses liquid cooling to remove heat from the system.

LabVIEW Software

We used LabVIEW to program our PID algorithm. LabVIEW is the National Instrument's graphical programming language. LabVIEW programs consist of block diagrams and front panels. The block diagram is the G code, and the front panel contains controls and indicators for the user to use and view. We build two LabVIEW programs: a subVI to measure the temperature of the system, and a VI to control the temperature.

Connecting the Hardware

When building our thermal controller, we used wires to attach the thermistor, EMANT300, H-bridge, and TEC. We attached one end of the thermistor to the IDAC socket and to an analog input socket (on the EMANT300), and we attached the other end of the thermistor to ground, AINCOM (analog input common), and REFIN- (the negative reference voltage). This allowed the analog current to flow through the resistor, and for us to measure the voltage drop across the thermistor. We also connected the PWM, enable, and heat/cool wires from the H-bridge to the EMANT300. The TEC and Water Cooling System were also attached to the setup, and the USB was plugged into the computer. The general hardware setup is shown in Diagram (4a), and Diagram (4b) shows a connection setup to the sockets of a DAQ system similar to our connection-setup.

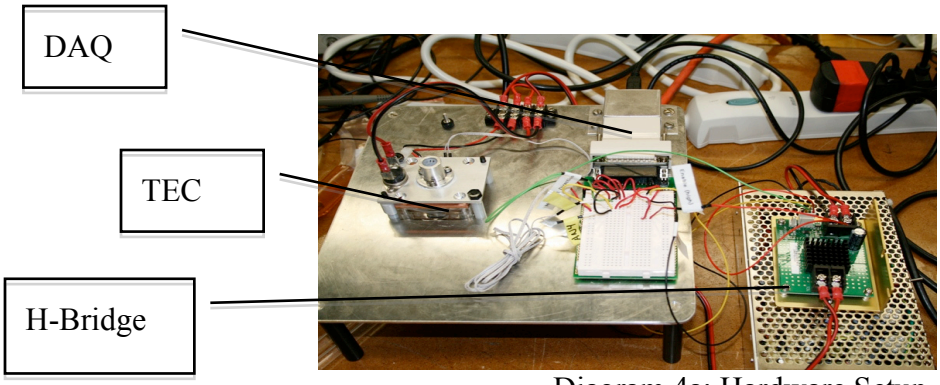


Diagram 4a: Hardware Setup

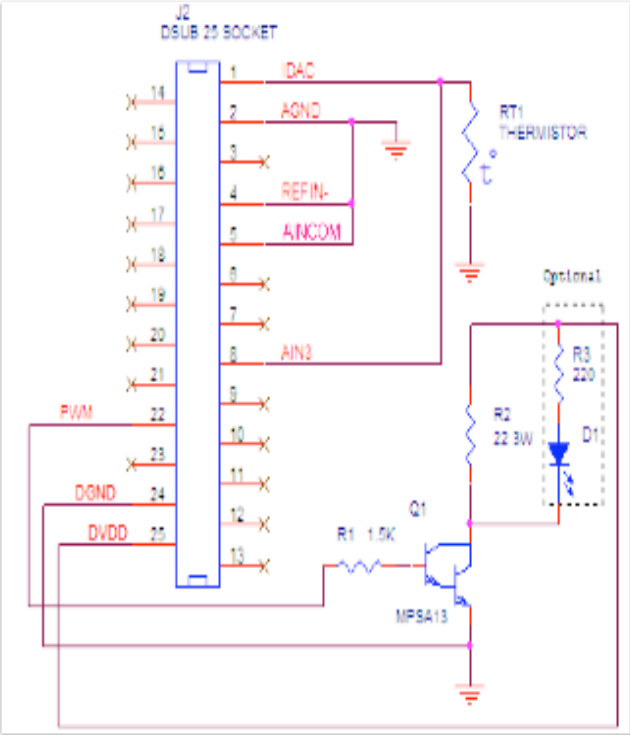


Diagram 4b: Connection Setup

To make sure that the hardware was set up correctly, we wrote a LabVIEW code tot control the PWM, and we used an oscilloscope to measure the power output. At this point, the front panel of our program consisted of an on/off toggle switch and a PWM knob that we sued to control the duty cycle of the power output. We knew that our code functioned correctly and that our setup was correct when we were able to use the LabVIEW code to modulate the pulse on the scope.

Temperature Measurement VI

As the temperature in the room changes, the thermistor's resistance changes as well. The data acquisition system passes data regarding the changing voltage across the thermistor to the computer. We wrote a LabVIEW "Thermometer" program to convert this voltage drop into a temperature measurement, using the Steinhart-Hart temperature-resistance equation. We wrote the program with the guidance of a similar LabVIEW sample VI, Example Thermistor.

The conversion formula that we used is Equation (2), solved for T and with the appropriate constants plugged into the equation. We also converted the absolute temperature to Celsius and Fahrenheit temperatures, which were displayed on our front panel, along with a graph of the temperature over time. The block diagram and front panel for our Thermometer subVI are shown in Diagrams (5). Although it is not shown in the diagram, the block code was enclosed in a while loop, so that we could take data continuously.

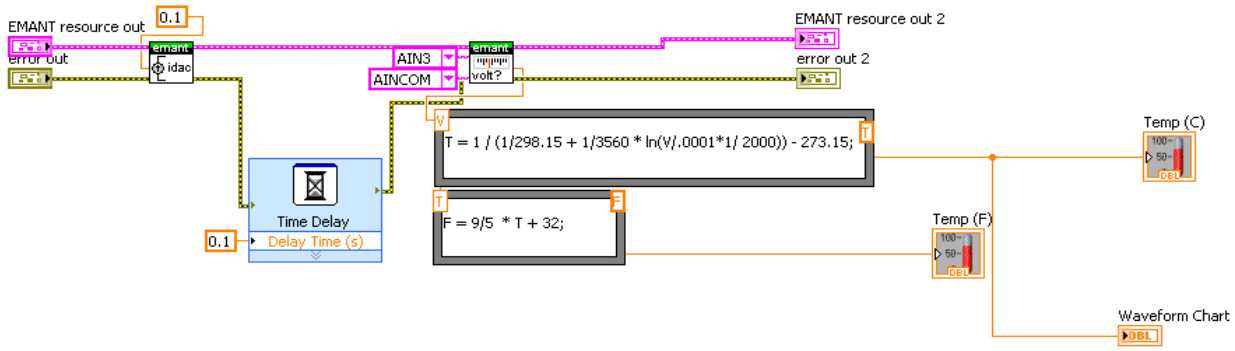


Diagram 5a: Block Code for Thermometer SubVI

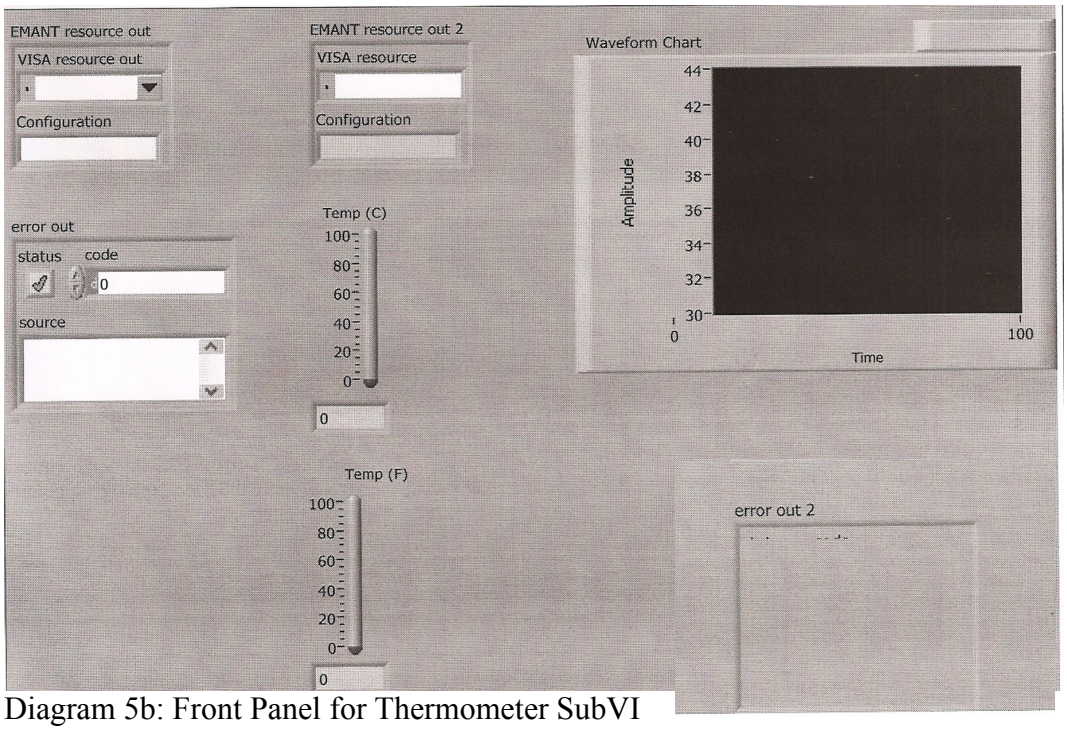


Diagram 5b: Front Panel for Thermometer SubVI

We tested our Thermometer SubVI by blowing on or squeezing the thermistor and making sure the temperature reading responded as expected.

PID Algorithm and Code

A common type of control algorithm is the Proportional-Integral-Differential controller (PID). The purpose of a PID program is to minimize the error between a setpoint and the actual measured value of whatever is being controlled (the "process variable"). The PID algorithm that we used is a type of feedback mechanism, meaning the transformation of the process variable is based on the process variable's previous output. Diagram (6) shows the PID controller process.

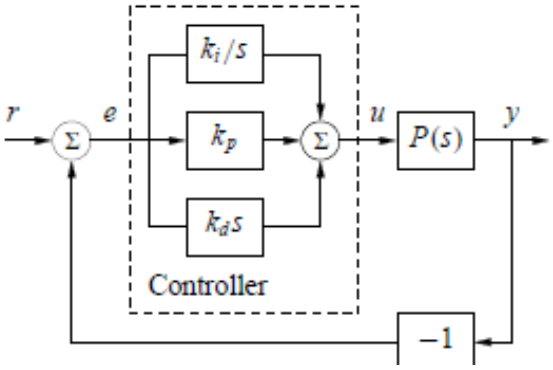


Diagram 6: PID Controller Process

In our particular system, we were controlling the temperature and trying to minimize the difference between the user-input set temperature and the actual temperature measured by the Thermometer SubVI. Ideally, the temperature should reach exactly the set temperature as quickly as possible, without overshooting or oscillating about the set temperature.

We began by adding the Thermometer SubVI to our code and adding a heat/cool toggle switch. We created a program that allowed us to manually control the temperature. Our front panel now contained a heat/cool switch, and the PWM controlled how much heating or cooling power was being output. The goal of our PID algorithm was to add an automatic setting to our program to regulate the amount of power being output.

Equation (3) is the PID formula that we added to our code (we used the right side of the equation, although the only difference is in the numeric constants). e is the error, or the difference between the set temperature and the measured temperature, and t is time. The equation includes three constants: k_p, the proportional term; T_i, the integral time; and T_d, the derivative time. u is the sum of the proportional, integral, and differential terms, and is the percent power output.

$$u = k_p e + k_i \int_0^t e(\tau)\,d\tau + k_d \frac{de}{dt} = k_p\left(e + \frac{1}{T_i}\int_0^t e(\tau)\,d\tau + T_d \frac{de}{dt}\right). \quad (3)$$

Our complete block code and front panel, after incorporating the PID algorithm, are shown in Diagrams (7). A few notes regarding the block diagram and front panel follow:

- The final value of u must be forced between zero and one hundred, because it corresponds to the percent duty cycle
- The code includes an anti-windup that automatically resets the integral term to zero when it becomes greater than one hundred or less than negative one hundred. This avoids overshooting the set temperature due to integrator windup, when the integrated term builds up so much that it remains saturated even after the error has changed.

- The three constants in Equation (3) are controls, rather than constants to allow for easy tuning.
- The left side of the block diagram contains the controls. The automatic controls allow the user to set the desired temperature and the tuning constants. The manual controls allow the user to control the duty cycle of the PWM and to set the system is heating or cooling.
- The right side of the diagram contains data, including: Celsius and Fahrenheit thermometers, a chart of the set temperature and the measured temperature with up to 1,000 seconds of history, a plot of the difference between the set temperature and actual temperature, a dial indicating the duty cycle of the PWM, and lights to indicate when the system is on/off and heating/cooling.

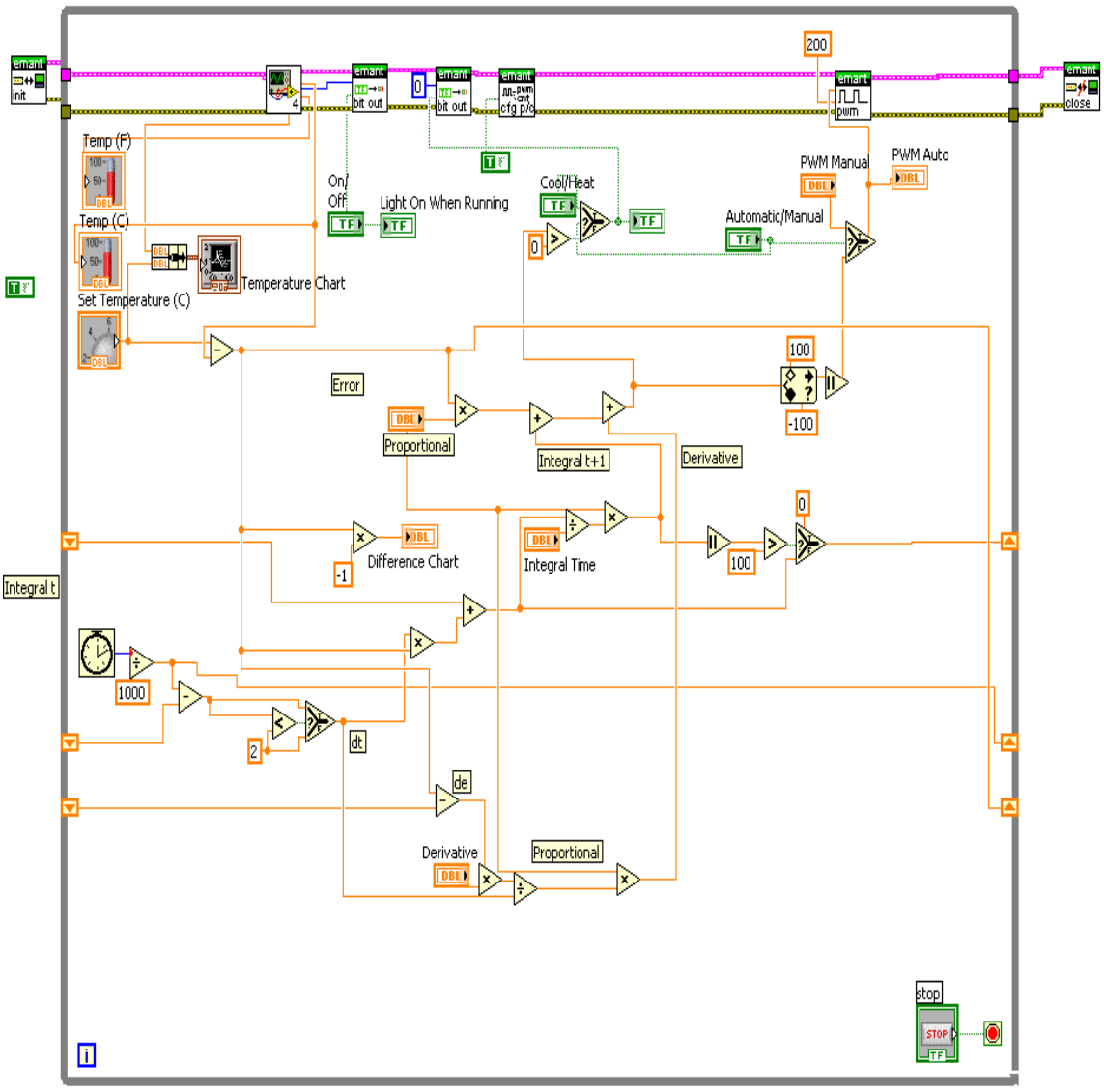


Diagram 7a: Block Diagram

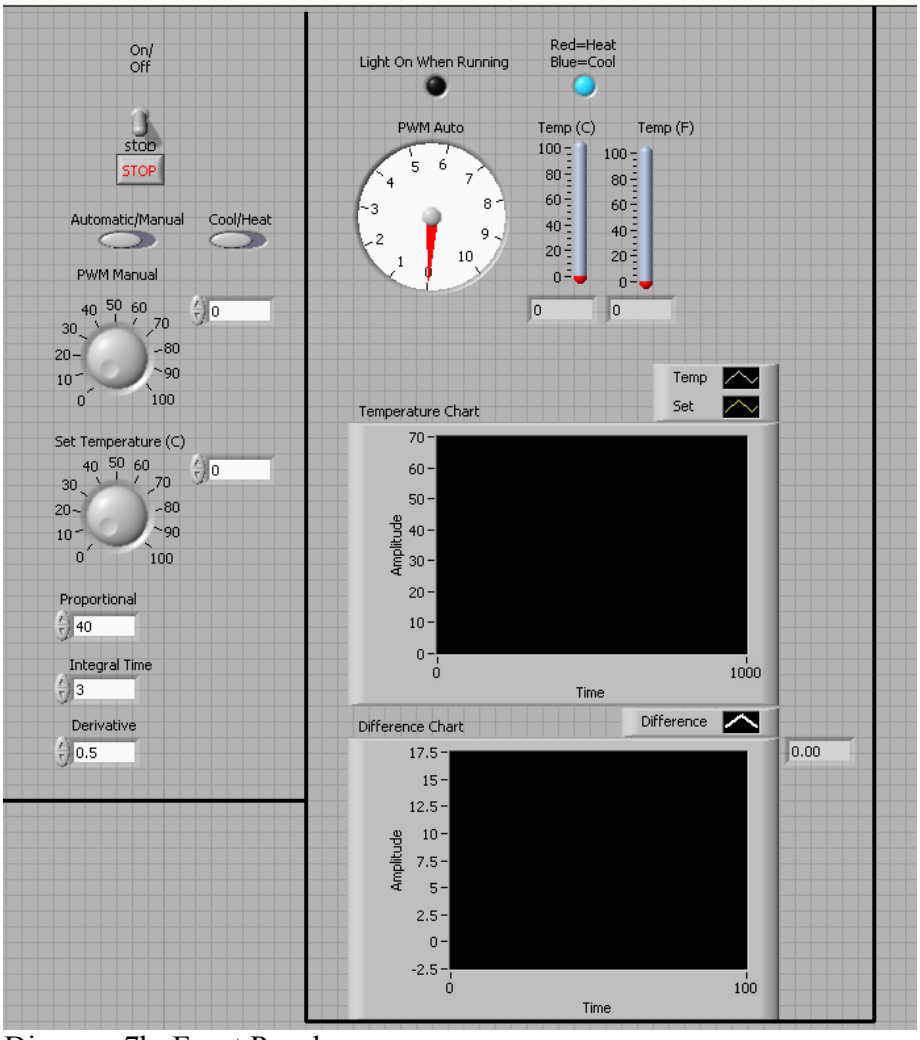


Diagram 7b: Front Panel

Tuning

After completing our code, we needed to tune our control loop. We first looked at how the system worked with only a proportional term. We set the derivative and integral times to zero by setting T_i to a very large number (because as T_i approaches infinity, the integral term approaches zero) and T_d to zero. For very small values of k_p, the temperature flat-lined before reaching the set temperature. There was heat loss into the environment, and the temperature reached a steady state before reaching the desired temperature. By increasing the proportional gain to 100, the temperature reached the set temperature but continued to oscillate about the set temperature. We measured the period of oscillation to be 20s. Although the proportional term alone was not enough to control the temperature, its advantage is in its fast response rate: A larger proportional gain very quickly brought the measured temperature near (but not exactly equal to) the set temperature.

Adding the integral term brought the temperature closer to the set temperature, but slowed down the response rate. When the integral time was small (making the integral term large), the temperature slowly climbed or fell towards the set temperature

without an overshoot. For larger integral times, the response was faster, but the system was more oscillatory. Ideally, when the PID is properly tuned, the proportional term is initially be large, but decreases as the measured temperature nears the set temperature. As the proportional term decreases, the integral term increases, until it eventually dominates. When the integral term is added, the system did not heat every time the temperature was lower than the desired temperature and cool when the opposite was true. This is because the accumulated integrated error did not immediately change signs every time the measured temperature passed the set point. Additionally, as mentioned above, the anti-windup made sure the integral term never became too large, by automatically clearing the accumulated integral when its magnitude was large.

The derivative term did not have a very big impact on the system. It did, however, help minimize the overshoot and help make the temperature target in faster on the desired temperature. Diagram (8) shows common characteristics of proportional, proportional-integral, and proportional-integral-differential controllers. The data, however, was not taken from our setup.

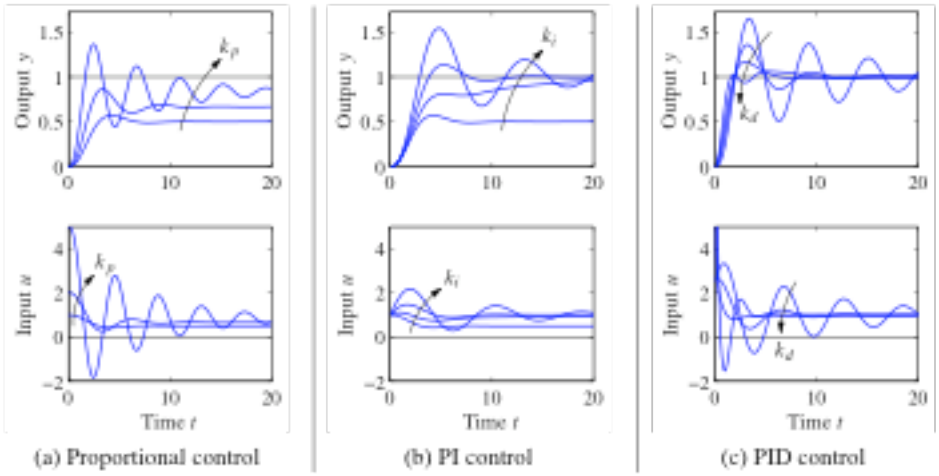


Diagram 8: Response of P, PI, and PID Controllers

To tune the PID parameters, we used the Ziegler-Nichols tuning rules. The frequency response method calculates the constants as functions of the critical gain k_c and the critical period T_c. We measured the critical gain and period by setting $T_d = 0$ and $T_i = \infty$. We then measured the period of oscillation and the gain when k_i was set large enough that the temperature oscillated. We used the Zeigler-Nichols tuning rules given in Diagram (10a) to determine rough values for the three constants. We then playing around with the parameters slightly to find the exact values where the controller response was both fast and accurate, with a minimum overshoot. We saw that the constant values that worked best were: $k_p = 40$, $T_i = 3$s, and $T_d = 0.5$s.

Type	k_p	T_i	T_d
P	$0.5k_c$		
PI	$0.4k_c$	$0.8T_c$	
PID	$0.6k_c$	$0.5T_c$	$0.125T_c$

Diagram 10a: Ziegler-Nichols Tuning Rules

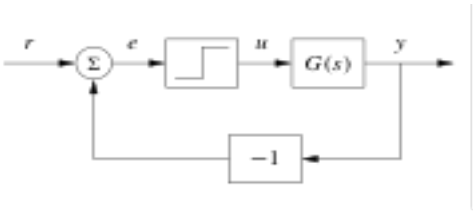


Diagram 10b: Relay Feedback

Relay feedback can be used to automatically tune the controller by automatically obtaining the values of k_c and T_c. This is done by connecting the process in a feedback loop with a nonlinear element with a relay function (Diagram (10b)). The purpose of relay feedback is to maintain an oscillation of minimum amplitude about the set temperature. However, we did not use relay feedback in our thermal controller.

System Response

When we tuned our controller, we found that it behaved as expected. The measured temperature came within a few hundredths of a degree to the set temperature. There was, however, a small overshoot (that was larger when the initial error was larger). Diagrams (10) include four charts. Two charts each show 1,000 seconds of data of the set and measured temperatures versus time. The other two charts graph the difference between the set and measured temperatures. The initial difference in each graph is either ±20 degrees Celsius.

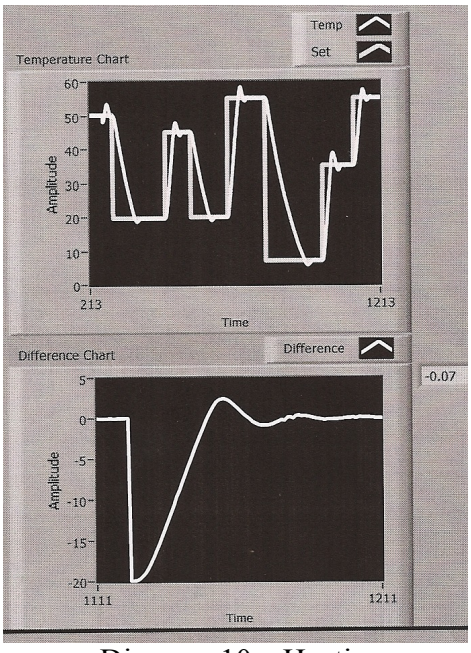


Diagram 10a: Heating

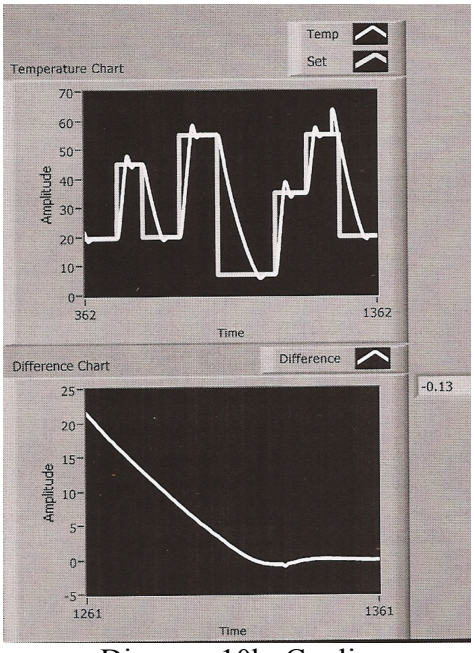


Diagram 10b: Cooling

It is obvious from Diagrams (10) that the controller overshot more when heating than when cooling. For heating, the overshoot was around two to three degrees, while for cooling, it was only about one degree. The explination for this asymetry lies in the Equation (4).

(4) $$Q_h = S_m T_h I + 0.5 I^2 R_m - K_m(T_h - T_c).$$

In Equation (4), Q_h is the heat, S_m is the Seebeck coefficient, R_m is the electrical resistance, K_m is the thermal conductance of the TEC, I is the electric current (in amps) in the TEC, T_h is the absolute temperature on the hot side, and T_c is the absolute temperature on the cold side of the TEC. Equation (4) shows that the heat can be broken down into three different terms. The first term is the amount of heat moved by the Peltier effect, the second term is the joule heat due to the electrical current that is input, and the last term accounts for the conduction due to the temperature difference across the TEC.

The first and third term can be either positive or negative, but the second, I^2 term term will always be positive. Equation (4) shows that regardless of whether the system was heating or cooling, there was always heat coming onto the plate due to the current. As long as the current was flowing in *either* direction there was at least one positive term added to the total amount of heat. Thus, the total heat in the TEC plate always was greater than it would have been if the heat was strictly regulated by Equation (3) (Of course, Equation (3) does not dircetly regulate heat). Because we did not account for the joule heat in our controller, the overshoot when the temperature was climbing was always greater than the overshoot when the temperature was falling.

Although we did not do so, it is possible to account for the added heat by adding a feed forward term to the process. A feedforward term would anticipate in advance how much joule heat would be added to the plate. In order to include a feedforward term, we would need to revise our model, process, and code.

The revised process with a feed forward term is shown in Diagram (11a). Diagram (11b) shows the process: u enters the process and its absolute value is taken (and it is forced between -100 and 100). |u| enters the DAQ, and current is drawn into the H-bridge, using PWM to control the amount of current. The high and low voltages from the H-bridge enter the TEC, where heat is geterated. The thermistor's resistance changes because of the temperature change, and the data goes to the DAQ. Then the data aquired enters the PID program and is coverted into a temperature. This temperature is what exits the process P(s).

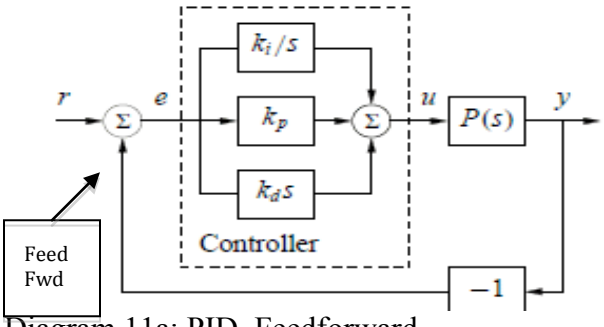


Diagram 11a: PID, Feedforward

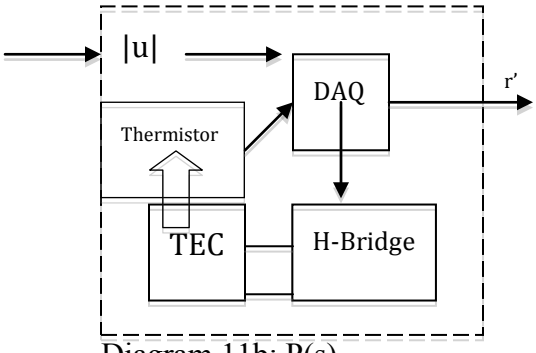


Diagram 11b: P(s)

Conclusion

This experiment was successful, in that we were able to build a thermal controller (with a user-friendly front panel) that uses the PID algorithm. Our controller safely, accurately, and quickly responds to changes in the set temperature by automatically modulating the PWM until the measured temperature matches the set temperature (it can also maintain the set temperature indefinitely). While our controller is precise to within a few hundredths of a degree, we could improve it by adding a both a feed forward process to reduce the overshoot (especially when heating) and a relay feedback system to automatically tune the proportional, integral, and differential constants. Still, without these additions our thermal controller is accurate enough to use in the next part of the experiment. Perhaps most importantly, this experiment taught us the basics of control algorithms and how to work with common thermal devices.

Works Cited

http://people.brandeis.edu/~fraden/phys39/Thermal%20Control/assignment.html
(including links from the website)

http://en.wikipedia.org/wiki/H-bridge

http://www.accuthermo.com/products_5.asp?pid=20099603084411311 01&pname=FTX300+H-Bridge+Amplifier+for+TEC+Temperature+Controller+Systems

http://en.wikipedia.org/wiki/Thermistor

http://www.emant.com/251004.page

http://en.wikipedia.org/wiki/Pulse-width_modulation

http://www.explainthatstuff.com/thermostats.html

http://en.wikipedia.org/wiki/Thermoelectric_effect

http://en.wikipedia.org/wiki/PID_controller