Advanced Physics Laboratory
Lab I: Thermal Control

Natalia K. Taub

November 13, 2023

Abstract

To investigate the application of control theory to thermal control problems, we constructed a system which uses a thermoelectric cooler (TEC) and a thermistor to control the temperature of a small metal block attached to a test bench. Using an Arduino Uno micro-controller and a MATLAB application running on a connected laptop, we implemented and then tested and calibrated P and PI control algorithms.

I. Theory

To adjust the temperature of the block toward a given set point and then maintain it there, a control algorithm was required. Because we lack sufficient knowledge of the internal dynamics of the system a feedforward control system could not be utilized. Instead we implemented and then tested two feedback control systems.

a) Proportional Control

The first control algorithm we implemented was proportional control, which may be expressed as

$$y(t) = K_p e(t) \tag{1}$$

where $y(t)$ is the control signal at time t, which, as discussed below, becomes the signed duty cycle of the Arduino PWM output that drives the TEC to heat or cool the block. $e(t)$ is the error given by

$$e(t) = T_{\text{set}} - T(t) \tag{2}$$

where $T(t)$ is the measured temperature at time t and T_{set} is the set temperature. K_p is a positive constant, as positive errors should produce a positive control signal to heat the block. Proportional control has a major flaw, however. This algorithm cannot, for reasons discussed below, hold the system at any set temperature other than room temperature.

b) Integral Control

To remedy the inability of a proportional algorithm to reach the set point, we then implemented an integral control algorithm. Here the control signal is instead given by

$$y(t) = K_p e(t) + K_i \int_0^t e(t')dt' \tag{3}$$

where K_i is an additional positive constant. The integral term will build up, increasing the magnitude of the control signal as the system hovers below or above the set temperature. Hence it will eliminate the drooping effect present with pure proportional control.

c) Tuning

The ideal values for the control parameters K_p and K_i will be determined by the system in characteristics of the control system and other design constraints. Several competing considerations here are particularly important. If the set temperature is not room temperature, there will be some net heat entering the system from or being lost to the endowment even at constant temperature and hence some constant control signal will be required to maintain the final state of the system. Per equation (1), however, the control signal is proportional to the error and so in order to maintain the steady state temperature there must be some error. The system temperature thus a droops away from the set temperature and toward room temperature. Because K_p is the constant of proportionality between $y(t)$ and $e(t)$ the magnitude of the droop will be inversely proportional to K_p for a given set temperature. This steady state error will not occur when integral control is used, however, because the integral of the error will continue to grow for any constant error and the control signal will respond, pushing the system closer to the set temperature.

More generally, larger values of the control parameters will make the control system more responsive to errors and thus lead to the system moving toward faster toward its set point. This does not, however, imply that the ideal values of K_p and K_i are as large as possible, given the constraints of the hardware. As the values of the control parameters increase they eventually reach a critical point where the lag between the activation of the TEC and the temperature of the thermistor increasing – as heat must diffuse through the block – creates first transient and then steady state temperature oscillation. The ideal parameters will be those which balance these dueling considerations, producing a system which is responsive and gets close to the set point but not unstable.

Several methods are available to determine these ideal values, of which we utilized two. The first was trial and error. We tested a variety of values, determined that some were clearly too low – due to slow response to a change in set point – and some too high – due to oscillation – and chose values between the two extremes. We then tried the open loop Ziegler Nichols method. Here the response of the system to a step change in the control signal is observed, and the value of the control signal (χ_0), the change in temperature (K_m), the dead time before the system responds to the control signal (t_d), and the time constant for the system response (τ_m) are used to calculate the ideal values using the expressions given in Table 1. below.

	K_p	K_i
P-Control	$\frac{\chi_0}{K_m}\frac{\tau_m}{t_d}$	0
PI-Control	$0.9\frac{\chi_0}{K_m}\frac{\tau_m}{t_d}$	$\frac{K_p}{3.3t_d}$

Table 1: Formulas for Ziegler Nichols open-loop parameters, taken from http://216.92.172.113/courses/phys39/Thermal%20Control/PID/07-PID%20Controller.pdf.

II. Experimental Setup

Our experimental setup can be separated into three parts; the temperature measurement and control hardware and its associated analogue circuitry, the Arduino Uno that performs the necessary calculations, and the connected laptop running the Matlab App which serves as our human interface.

a) Hardware Setup

To monitor and regulate the temperature of the block we used a thermistor and a thermoelectric cooler, controlled by an Arduino Uno micro-controller. A full diagram of the control circuit can be seen in Fig. 1 below.

a.1) Temperature Monitoring

Temperature monitoring was accomplished using a thermistor, which was placed in a small hole drilled in the metal block. As shown in the circuit diagram, it was wired in series with a resistor to create a voltage divider and then the voltage drop over the thermistor was measured using one of the Arduino's analogue

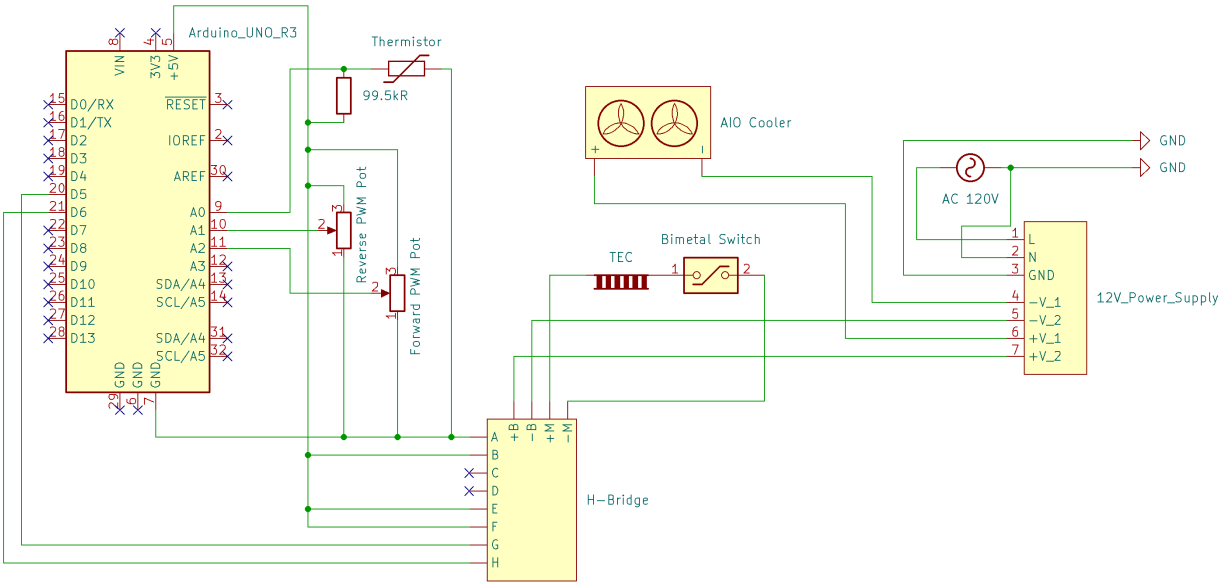

Figure 1: Diagram of the full temperature control circuit

inputs. Knowing the resistance of the other resistor and the total voltage supplied allowed us to calculate the current through the divider and thus the resistance of the thermistor. This in turn allowed us to measure temperature as the temperature, T, and resistance, R, of a thermistor are related by the B-parameterized Steinhart-Hart equation

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln \frac{R}{R_0} \tag{4}$$

where T_0 is a reference temperature, in our case 25°C. R_0 is the resistance at that reference temperature, and B another parameter, values for which were determined from the data sheet for the thermistor to be 100 kΩ and 4540K respectively. Hence, by rearranging the above equation, we may compute the temperature of the thermistor via

$$T = \frac{B}{ln(R/R_0) + B/T_0} \tag{5}$$

a.2) Temperature Control

For temperature regulation a thermoelectric cooler was connected to the metal block to supply, or remove, heat. The other side of the TEC was attached to an AIO liquid cooler, which used a water block with an integrated reservoir and a pump to transport heat to or from a radiator with two attached fans, where it was dissipated into the room, or vice versa depending on the mode of operation.

The TEC itself contains a series of alternating pieces of differing electrical conductivity in a sheet, with conductors placed on either side as seen in Fig. 3 below.

When an electrical current flows across the cooler, the peltier effect moves heat from one surface to another to establish a temperature gradient between them. This effect is fully reversible by reversing the direction of current flow, allowing the TEC to either heat or cool the block as desired. More specifically, the heat per unit time supplied to the hot side of the TEC is given by

$$Q_h = S_m T_h I + \frac{1}{2} I^2 R_m - K_m (T_h - T_c) \tag{6}$$

where I is the electric current supplied, S_m is the Seebeck Coefficient of the TEC, R_m its resistance, K_m its thermal conductance, and T_h and T_c the temperatures of its hot and cold sides. The first term on the left hand side accounts for the heat moved to the hot side by the Peltier effect, the second for the resistive

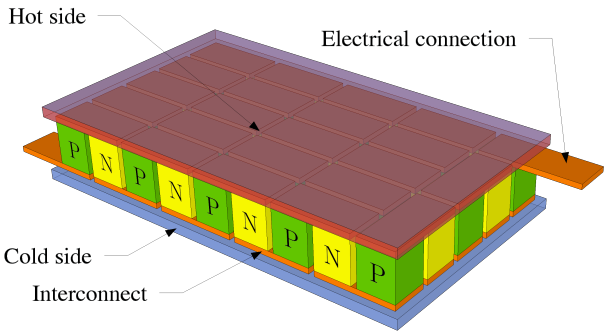

Figure 2: Peltier Effect Cooler, image from
https://upload.wikimedia.org/wikipedia/commons/a/a2/Peltierelement.png

heating due to current flow within the device, and the third for heat lost to conduction across device. As a result, the heat per unit time supplied to the cold side is not $Q_c = -Q_h$ but rather

$$Q_c = -S_m T_h I + \frac{1}{2} I^2 R_m + K_m (T_h - T_c) \tag{7}$$

because resistive heating occurs equally on both sides of the TEC.

Assuming that the temperature of the block is effectively uniform, either $T = T_h$ or $T = T_c$ depending on whether the TEC is being used to heat or cool the block. In steady state the heat supplied by the TEC will be exactly canceled out by the net heat the block exchanges with its environment, so that $Q_{h/c} = hA(T - T_{room})$ per Newton's law of cooling. Here h and A are the heat transfer coefficient and the exposed surface area of the block. The steady state conditions, for heating and cooling, are thus

$$hA(T - T_{room}) = S_m T I + \frac{1}{2} I^2 R_m - K_m (T - T_c) \tag{8}$$

$$hA(T - T_{room}) = -S_m T_h I + \frac{1}{2} I^2 R_m + K_m (T_h - T) \tag{9}$$

and hence, rearranging, in steady state the temperature is

$$T = \frac{hAT_{room} + \frac{1}{2} I^2 R_m + K_m T_c}{hA + K_m - S_m I} \tag{10}$$

when the block is being heated, and

$$T = \frac{hAT_{room} + \frac{1}{2} I^2 R_m + K_m T_h - S_m T_h I}{hA + K_m} \tag{11}$$

when it is being cooled. Because the linear current term is always negative and in the numerator when for heating and in the denominator for cooling, we can expect that so long as $hA + K_m > S_m I \frac{dT}{dI}$ will be significantly larger for heating than for cooling. We may expected this to generally be true, as $K_m = 0.5\ \mathrm{W\,m\,K^{-1}}$ while $S_m = 0.024\ \mathrm{V\,K^{-1}}$.[1] Conceptually, this asymmetry results from the different effect of resistive heating on the heating and cooling of the TEC.

a.3) Additional Hardware

As the power output of the Arduino is insufficient to drive the TEC directly, power was supplied to it via an H-Bridge connected to a separate power supply. The H-Bridge, wired as shown in Fig. 1 above, acts as an amplifier capable of generating a voltage difference in either direction across the M+ and M- pins to

[1] Values from http://216.92.172.113/courses/phys39/Thermal%20Control/hardware.html

allow either heating or cooling. The voltage difference across those pins is proportional to the inputs on the on the forward and reserve control pins, G and H, which were connected to two of the Arduino's analogue outputs, where the voltage could be adjusted between 0 and 5V by adjusting the duty cycle of the pulse width modulation. Hence, utilizing Ohm's law, the current across the TEC was given by

$$I = \frac{V_s}{R_m}\frac{\text{PWM}}{255} \tag{12}$$

where V_s is the maximum voltage supplied by the power supply to the H-Bridge, in our case 12V. Thus, we can see that $I \propto \text{PWM}$ so the argument given above for $\frac{dT}{dI}$ holds for $\frac{dT}{d\text{PWM}}$ and we may expect that $\frac{dT}{d\text{PWM}}$ will be larger for heating than for cooling due to the asymmetry produced by resistive heating.

Additionally, a bi-metal switch was placed in series with the TEC and embedded within the block to act as a hardware safety. The switch contains two strips of metal with different thermal expansion coefficients so that as it heats above a certain temperature – in our case 70°C – the circuit would be broken and the TEC deactivated. This prevents the it from overheating and destroying itself, as a result of software errors.

b) Arduino Software Overview

The Arduino has much lower software overhead than Matlab, and so runs much faster. For this reason as much of the control functionality as possible was implemented on the Arduino microcontroller. Only the human interface was done in Matlab on the laptop, which was connected to the Arduino's USB-b port. The two communicated with one another via serial.

The Arduino program[2] had two key functions, namely temperature monitoring and control. For the first, it calculated the temperature of the block from the voltage drop over the thermistor, using equation (5) and sent that across the serial connection to Matlab.

The temperature control functions of the Arduino depended on the control mode it was set to by the Matlab interface. If set to hardware manual control, the analogue outputs which control the H-Bridge were set manually by turning the two potentiometer, each of which was wired as a voltage divider and connected to an analogue input. The Arduino simply mapped the 0-1023 input onto the 0-255 PWM range and then wrote it to the H-Bridge outputs. If in software manual mode, the Arduino instead read the PWM value fed to it by the Matlab interface and wrote it to the H-Bridge outputs.

When set in the two automatic control modes, the Arduino read the set temperature, K_p, and K_i values sent by Matlab, then calculated the control signal. For proportional mode equation (1) was used directly, while for proportional-integral mode

$$y(t) = K_p e(t) + \frac{K_i}{2}\frac{e(t) - e(t-\Delta t)}{\Delta t} + I(t) \tag{13}$$

was used, where $e(t)$ is the error function, Δt the time elapsed since the last time the calculation was run, and $I(t)$ the running sum of all previous K_i terms. This is a trapezoidal-Reimann sum of the error function and thus, because the calculation was done many times a second, produced a very close approximation of equation (3). To prevent integral windup in PI mode, the $I(t)$ term was zeroed out every time the temperature crossed the set point if it was above a set thresh-hold. This control signal was then written to the forward analogue output if it was negative and the reverse output if it was positive, after being clipped if any higher than 255.

Regardless of the control mode, two software safeties were implemented. If the temperature of the block was calculated at over 60°C or nonzero values were to be written to both the H and G pins of the H-Bridge, both outputs were set to zero so that the TEC was inactive. The temperature safety prevented us from tripping the hardware safety and having to wait for the block to cool off before it could be used again. The other prevented a shoot through, whereby writing to both the forward and reverse H-Bride inputs causes a short which destroys the component.

[2]The Arduino code utilized may be found at https://github.com/WilliamSorger/Brandeis-PHYS39A.

c) Matlab Software Overview

A Matlab app,[3] running on a laptop plugged into the Arduino's USB-b port and interacting with it via serial, functioned as our human interface. The GUI can be seen is Fig. 3 below.

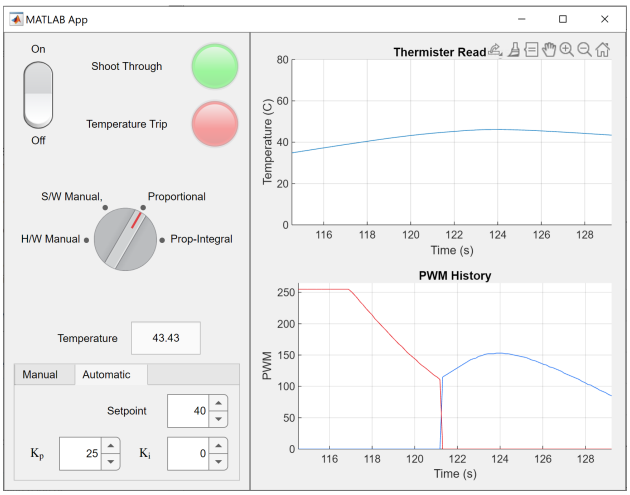

Figure 3: The Matlab interface

The two lights were used to indicated when the software safeties had tripped, the current block temperature as calculated in Arduino was displayed in one of the text boxes, and the last ten seconds of temperature data and PWM control signals were graphed on the two axes. The knob was used to select the control mode, and a series of text boxes allowed the setting of K_p, K_i, the set temperature, and the software manual PWM value. The temperature and PWM control signal data was also archived as the application ran, and then saved to csv file when it was closed.

III. Results

a) Heating-Cooling Asymmetry

To characterize our thermal control system, we began by using the software manual mode to set a series of constant values for the Arduino PWM outputs – and thus the power supplied to the TEC.[4] Broadly speaking the system response was similar in each of our five heating and five cooling tests. The temperature of the block followed an s-curve, beginning at room temperature and rising or falling quickly for a brief period before leveling off at an equilibrium temperature determined by the PWM value. Two examples of the transient dynamics, for PWM = 22 in heating mode and PWM = 121 in cooling mode, are shown in Fig. 4 below.

We then plotted the change in temperature produced by the TEC as a function of the set PWM values for both heating and cooling, and found the trend-line via linear regression. The resulting plots are Fig. 5 and Fig. 6 below. The change in temperature was used rather than the final system temperature because several of the tests were conducted on different days with room temperatures varying by several degrees. For all cooling tests and most heating tests the system was at thermal equilibrium when the tests began, although for a few of the latter the system had leveled off after being cooled from the previous test but not quite reached equilibrium. In all cases the last temperature measurement before the PWM value was set was used to calculate the temperature difference.

As can be observed in these plots, when the the system was in heating mode the magnitude of the slope of the trend-line was significantly greater than when it in cooling mode; 0.406°C/PWM as compared to

[3]The Matlab utilized may be found at https://github.com/WilliamSorger/Brandeis-PHYS39A.

[4]The data produced by this and other tests, as well as the python code utilized to generate the graphs and fit trend-lines may be found at https://github.com/WilliamSorger/Brandeis-PHYS39A.

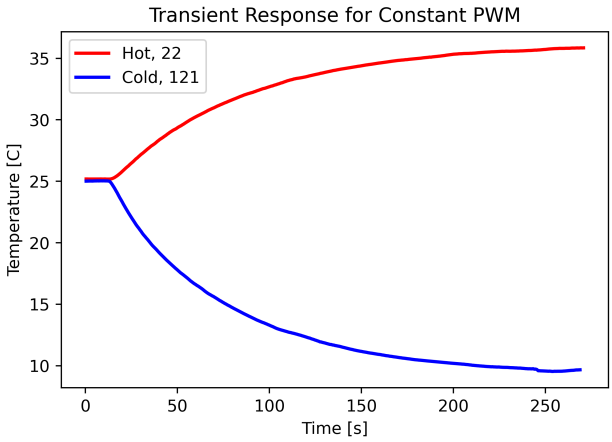

Figure 4: Transient response for PWM $= 22$ and PWM $= -121$

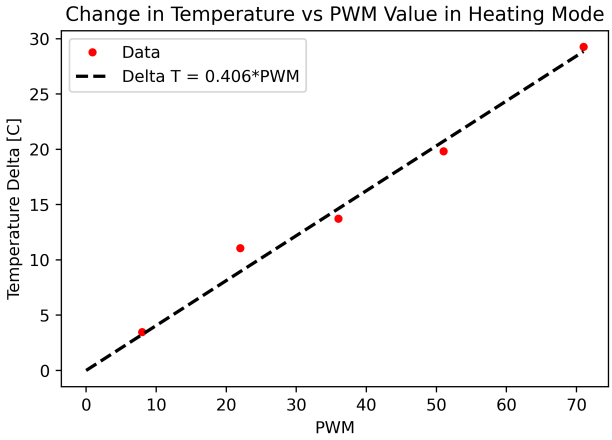

Figure 5: Plot of final temperatures vs. set PWM in heating mode

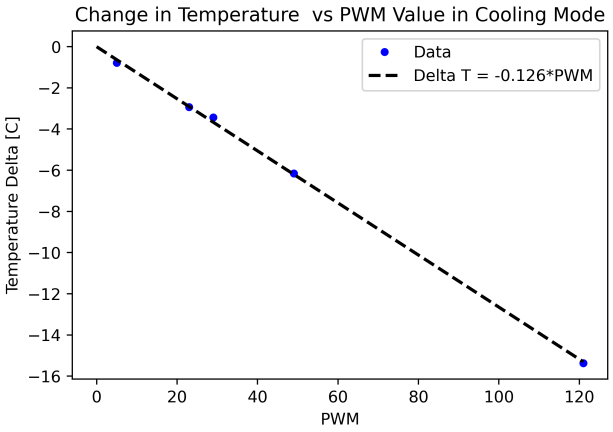

Figure 6: Plot of final temperature vs. set PWM in cooling mode

$-0.126°$C/PWM. This was exactly the behavior we expected, given the discussion of $\frac{dT}{dI}$ and $\frac{dT}{d\text{PWM}}$ above. Because resistive heating heats both the hot and cool sides of the TEC, the rate at which heat was produced on the hot side was larger than the rate at which it was removed form the cool side. Interestingly, the fit of the trend line is significantly better for the cooling data. The standard deviation of the error on the slope was 0.00105 for cooling and 0.0130 for heating. This may be explained by the additional error introduced by the uncertainty on the room temperature for some heating tests. Regardless, both plots are remarkably linear given the non-linear forms found for $T(I)$ in equations (10) and (11).

b) Automatic System Response

To further characterize the control system, we then tested both automatic control modes – proportional and integral – with several different K_p and K_i values in heating and cooling modes. The temperature responses for the various values, as well as the control signals during each test can be seen in Figures 7 through 14.

Several aspects of these results are notable. In the pure proportional control tests, the results were as expected, namely a large temperature droop for low K_ps which shrinks as it increases and concurrently increasing oscillation in both the temperature response and the control signal. For heating we found that $K_p = 30$ provided a reasonable balance of minimal droop and minimal oscillation, while for cooling $K_p = 50$ seemed about right. These produced no oscillation, although the droop was larger than we desired. The

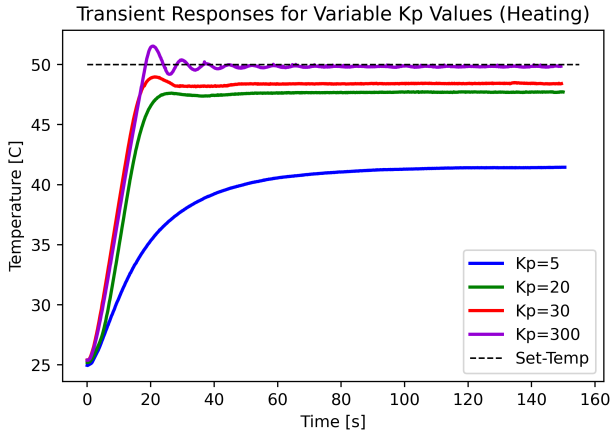

Figure 7: Transient response for various K_p values in heating mode

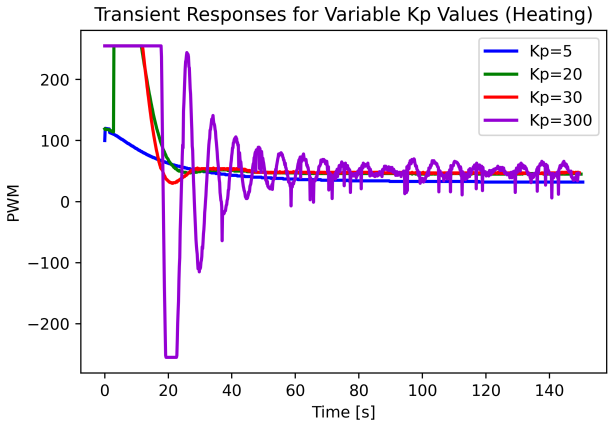

Figure 8: Plot of the control signals for K_p tests

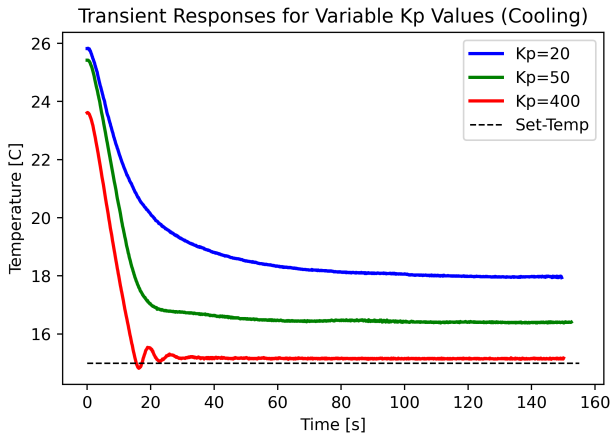

Figure 9: Transient response for various K_p values in cooling mode

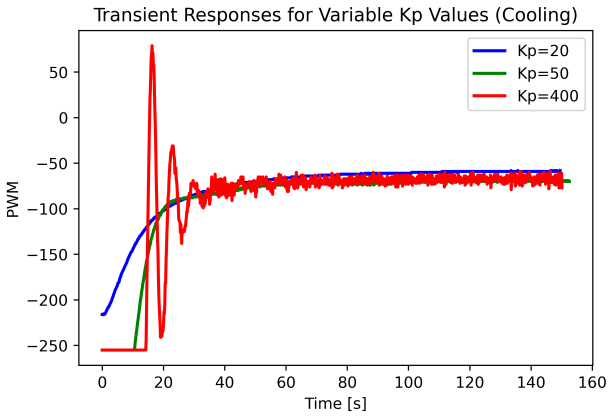

Figure 10: Plot of the control signals for K_p tests

larger ideal value for cooling reflects the heating-cooling asymmetry with respect to $\frac{dT}{d\text{PWM}}$ discussed above.

For testing our integral control system, $K_p = 52$ for heating and $K_p = 198$ for cooling were used for all respective tests, for reasons explain below. As expected, droop was suppressed almost entirely in all cases, though the time it took to reach the set temperature decreased with increasing K_i. The only outlier was the $K_i = 50$ test for heating, where the system instead leveled off at 49°C. Although the data file for that test listed the set point as 50°C, it is plausible that data for that test was mistakenly taken with a set point of 49°C. This which would explain the result, especially as the steady state temperature was exactly 49°C.

Finally, for several of our tests a large amount of noise was present, particularly in the PWM data. This is particularly notable in the $K_p = 300$ and $K_p = 400$ calibration tests and the K_i calibration tests for cooling. Given the high frequency and amplitude of the noise it seems unlikely to have resulted from actual temperature fluctuations in the block. Rather, electrical noise seems the most likely explanation. Small variations in the voltage supplied by the Arduino to the thermistor-voltage divider, perhaps arising from a loose power or ground wire, may have perturbed the current and voltage drop, effecting the resistance calculations and through them the temperature measurements. In turn, the small fluctuations in the temperature measurement would have influence the error and the control signal. That the noise is primarily visible in tests with large values for the control parameters, which would have amplified the effect of small errors in the block temperature, makes this the explanation more convincing. Rounding errors or other small variations

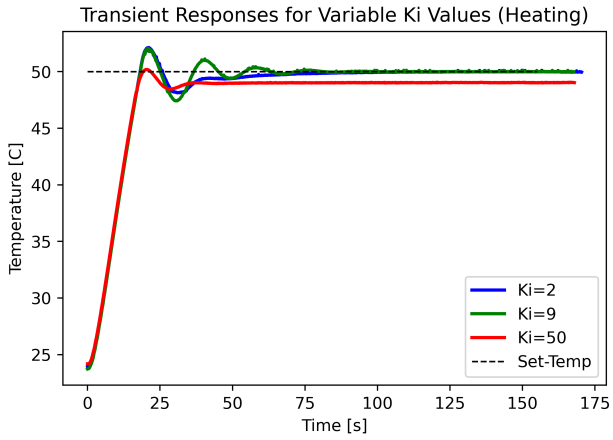

Figure 11: Transient response for various K_i values in cooling mode

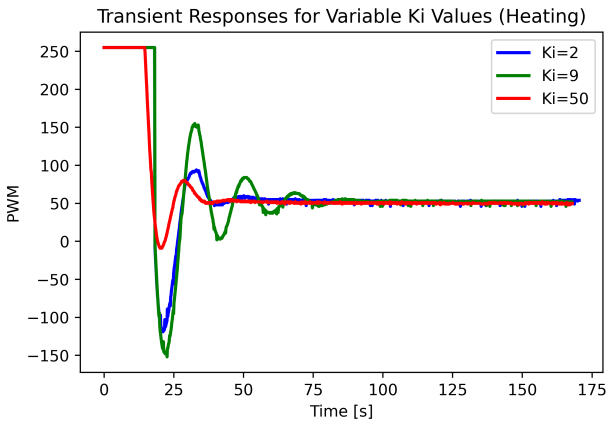

Figure 12: Plot of the control signals for K_i tests

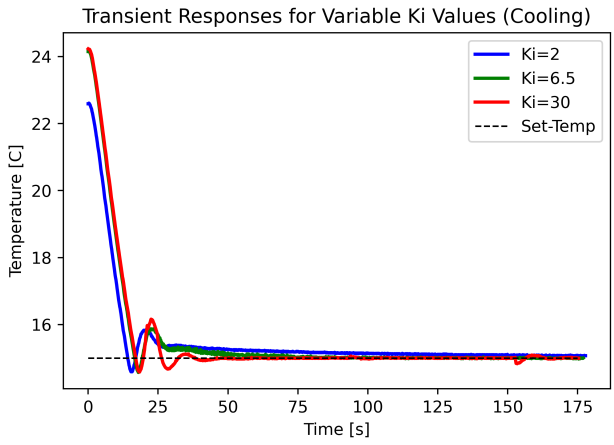

Figure 13: Transient response for various K_i values in cooling mode

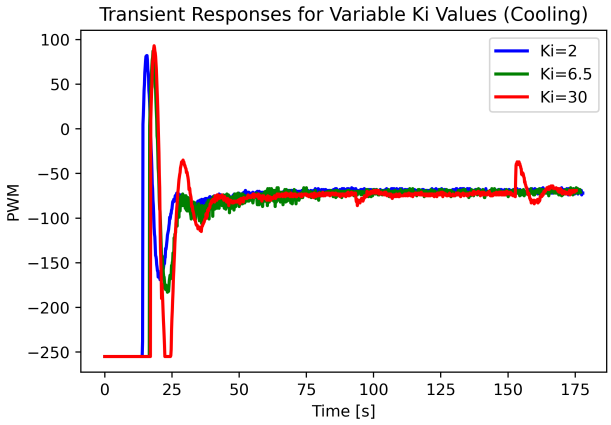

Figure 14: Plot of the control signals for K_i tests

due to the use of floating point arithmetic in the Arduino's calculations, for example, would not explain the increase of noise in large-parameter tests. A loose wire would also help explain the sharp temperature fluctuation just after 150 seconds into the $K_i = 30$ cooling test, visible directly in Fig. 13 and via its effect on the control signal in Fig. 14.

c) Ziegler Nichols Tuning

To helps us determine the ideal control parameters more rigorously, we utilized the open loop Ziegler Nichols method with the data from the two constant PWM test shown in Fig. 4 above. For χ_0 we used the PWM control signals, 22 and -121, while t_d was found manually by subtracting the time at which the PWM was changed from zero from the time at which the temperature of the system began to respond. After that, each group of five measurements was averaged to reduce noise, the data was differentiated numerically, and the largest slope was found. This largest slope of the step response give K_m/τ_m. Utilizing the expressions in Table 1, for the proportional algorithm we determined that the ideal values of K_p were 58 for heating and 220 for cooling. For integral control, meanwhile, we found that the ideal values were $K_p = 52$, $K_i = 6$ for heating and $K_p = 198$, $K_i = 31$ for cooling. The larger parameter values for cooling once again reflect the asymmetry with respect to $\frac{dT}{d\mathrm{PWM}}$ discussed above.

To test these parameter values, we then used them for our integral control calibration tests, shown in Fig. 11 through Fig. 14. ~~Unfortunately, due to a calculation error only discovered while writing this report we originally identified the ideal K_i values as 9 for heating and 6.5 for cooling, and so tested those rather than the values given above.~~ Nonetheless, the closest values we tested to the correct Ziegler Nichols parameters – $K_i = 9$ for heating and $K_i = 30$ for cooling – worked reasonably well, effectively minimizing the time the system took to reach equilibrium and the temperature droop. They produced more oscillation than we had tolerated with our Goldilocks calibration method, but the oscillations were transient. As expected, the Ziegler Nichols parameters were not large enough to render the system unstable in steady state.