Steven Lei 4/24/2025

EE 90

Lab 2 Report

Introduction and Design

The goal of this lab is to design and tune a heater drive circuit to regulate the temperature of a NT0801043600B1F thermistor. The circuit is implemented on a PCB and the component values to set the op-amp gain stage and the transistor current switching are specced out and then soldered on. The circuit also utilizes a MCP4728 DAC to control the voltage of the thermistor (and thus the temperature) and a ADS1115 ADC to measure the voltage of the thermistor. The regulation of the thermistor temperature is implemented through software by a PID controller that interfaces with python code using the MCP2221.

According to the lab instructions, the heater drive circuit should be tuned such that the thermistor will reach a final temperature within ≤ 0.1°K from the setpoint with a 30 min stability of better than 0.1°K. The schematic for the heater drive circuit is provided by the lab instructor:

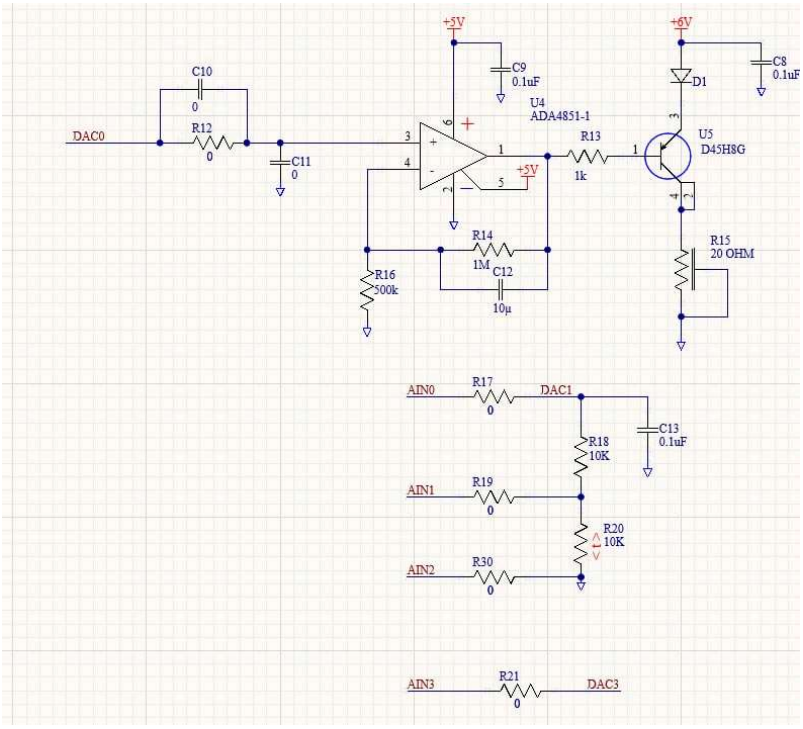

Figure 1: Heater Driver Circuit Schematic

For C10 and C11, I kept them as 0 since R12 already shorts C10. Technically with no R12 we have a DC filter with V+ = Vin * C11/(C10+C11). For the Op Amp, note that it is in non-inverting amplifier configuration, and thus the gain is 1 + (R14/R16) / (R14C12s + 1). R14/R16 determines the gain here and R14 * C12 is our pole. Since we want a low bandwidth, I let the RC constant equal to 10s. For the gain, we want the voltage into the base to be greater than the emitter voltage (6-diode offset) to turn the transistor OFF in a PNP config. So the base voltage has to be > 5.3V, so a gain of (1 + 1.5) for 3V input would be fine. When the transistor is on, the output voltage at the op amp should be 0V. Assuming the Emitter voltage is 6 - 0.7 = 5.3V and we want the transistor in saturation, V_Base should equal 4.7V. According to the ADA4851-1 datasheet, the max current sink is about 90mA, so 4.7/1K = 4.7 mA should be good. The typically allowed power through 0805 resistors is .125W, so 0.0047^2 * 1000 = 0.02209W, which is good.

Since we could not populate our board due to a missing IC, we had to use the TA board. The component values for the TA board are C10 = N/A, C11 = N/A, R16 = 10k Ohm, R14 = 6.8 kOhm, C12 = 1.88 nF, and R13 = 1k Ohm

Setup and Experiment

With the component values chosen, we then soldered SMD 0805 resistors, capacitors, and ICs onto our PCB. ***Discretion again:*** *We unfortunately could not finish soldering onto our own PCB since there were no more MCP4728 IC's left. As such, we used the TA board for our experiment.*

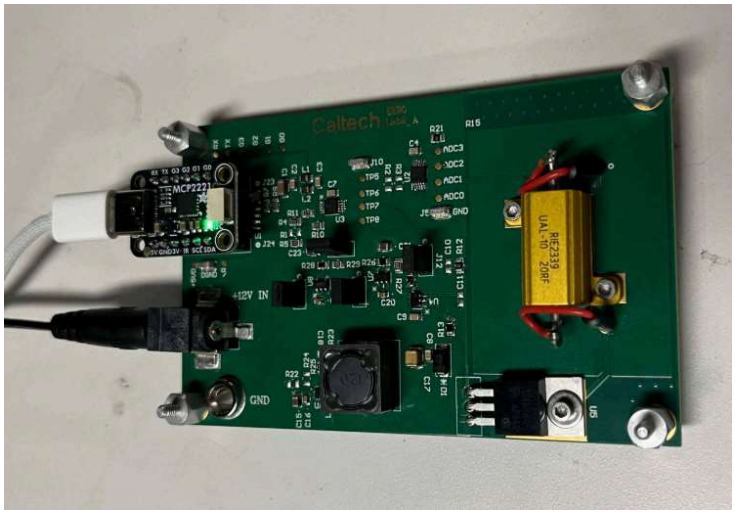

Figure 2: PCB with soldered on components

After populating our components, we then tested the board's voltage regulator units to ensure everything is functional before proceeding. We measured the voltage at pin 1 of J11 to ensure the regulator output was nominally 6V. Then we connected jumpers J11 and J13 to measure the voltage at pin1 of J12 and J14 to ensure they were nominally 5V and 3.3V respectively. Finally, we installed jumpers J12 and J14 to run our circuit.

Before running a full PID controller test, we first need to test the ADC and DAC units independently to ensure that they are working correctly. *See the appendix for the python code*. Our `test_dac()` function sets up the MCP4728 and generates 3.3V on channels A and B, 0.825V on channel C, and 0V on channel D. We measured these voltages using test points 5/6/7/8 on the board to verify the output was correct. For the ADC, our `test_adc()` function sets up the ADS1115 and prints out the channel voltages. Channel 2 output should be 0V, while Channel 0 and 1 should be between 0 and 5V as they are connected to the thermistor.

Next, we ran our `test_on_off()` which set the transistor in the heater driver circuit to fully on for 10 minutes and then fully off for 10 minutes. While doing so, we recorded the voltages at 1 second timestamps to find the maximum and minimum temperatures.

We implemented a standard PID controller in our `pid_controller()`. To tune the constants, we first start with the P term and adjust it such that our controller hits the setpoint until there are no more oscillations after 10 minutes. We observed some steady state error, so we tuned our I term to be fairly small but fast enough to help eliminate steady state error and reach the setpoint. To ensure the polarity is correct, we made sure all term signs were the same and printed the output voltage of the controller on each time step. We did not use a D team since tuning the P and I terms provided enough accuracy. During our tuning process, we added two features to our PID: 1) a deadband region such that if the error is within this deadband value, the PID error is set to 0 to prevent the controller from trying to change the current set value and 2) an integral bound to prevent the integral from winding up to a large value. We found these features to be helpful in preventing steady-state oscillations and increasing stability. After tuning was complete, we verified by running a 30 minute test to hit the 300°K setpoint in our `pid_test()` and then a 2 hour test in `long_test()`. For the 2 hour test, the setpoint for the first hour was set to 75% of the maximum temperature achieved in the on/off test. We interpreted this as MAX_TEMP - 300 * 0.75 since 75% of the actual MAX_TEMP would be well below 300°K. Similarly, the second hour the setpoint was then changed to 25% of the maximum achieved. All the data was recorded in python onto a CSV file and then plotted using matplotlib.

Lastly, we performed three FFT measurements using the FFT function in the oscilloscope. We probed the +6V, +5V, and +3.3V nets from the voltage regulator to analyze the spectral purity. We used the following settings for the FFT measurements:

Window Type: Hanning, **Center Frequency:** 500 kHz, **Span**: 1 MHz, **Scale**: 20 dbV and on the channel we used the following acquisition settings:

Averages: 64 (this helps reduce the noise floor).

Data of the FFT measurement was collected by saving the scope screen as an image.

Results

Voltage regulator measurements:

Voltage at pin 1 of J11/J13: 6.035V (target is 6V)

Voltage at pin 1 of J12: 5.02V (target is 5V)

Voltage at pin 1 of J14: 3.29V (target is 3.3V)

Final PID settings:

P: -1.1, I: -0.035, D: 0

Time step: 1s

Error deadband: ±0.075°K

Integral bound: 250.0

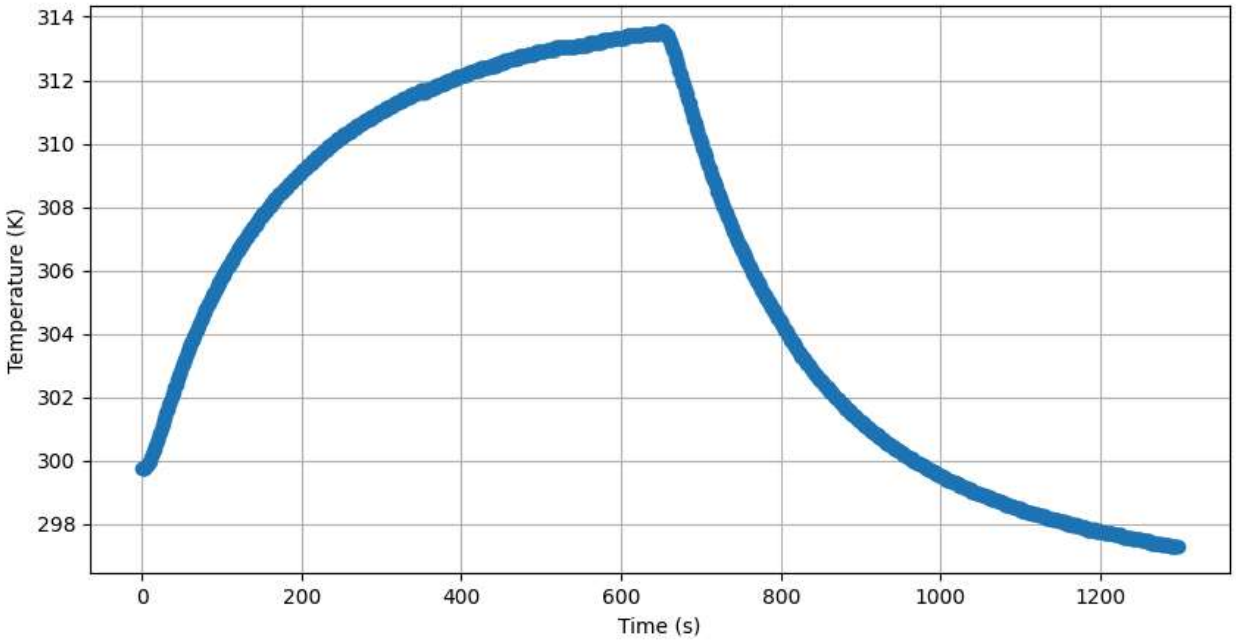

Figure 3: Thermistor ON and OFF (10 min each) temperatures.

Thermistor temperatures:

Thermistor **minimum** temperature: 297.276°K

Thermistor **maximum** temperature: 313.547°K

Thus the **75%** maximum target is (313.547-300) * 0.75 + 300 = 310.161°K

And the **25%** maximum target is (313.547-300) * 0.25 + 300 = 303.388 °K

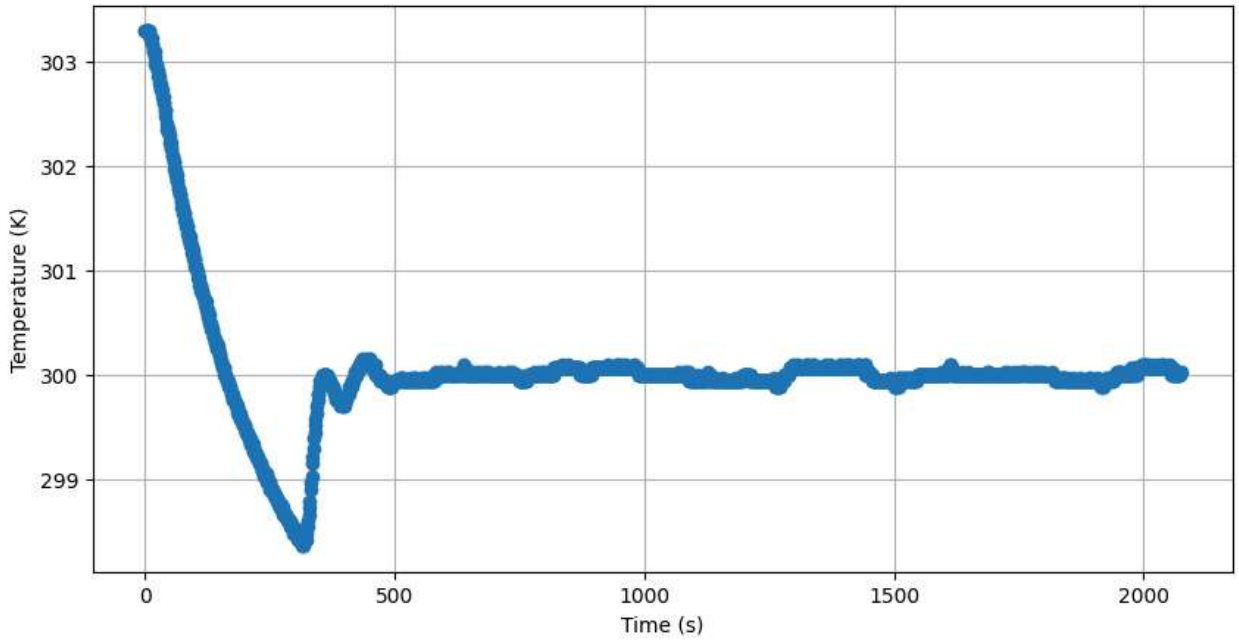

Figure 4 : Thermistor 30 minute response. Setpoint = 300°K

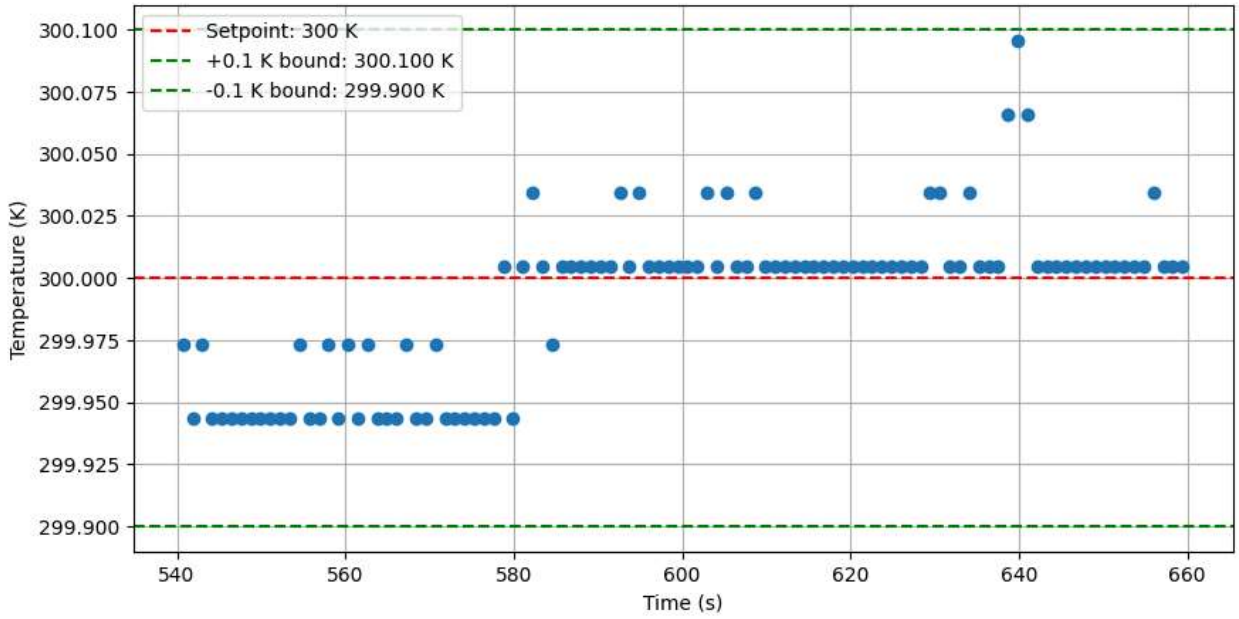

Figure 5: Thermistor 30 minute response, zoomed in on the 9 - 11 minute window. Setpoint = 300°K

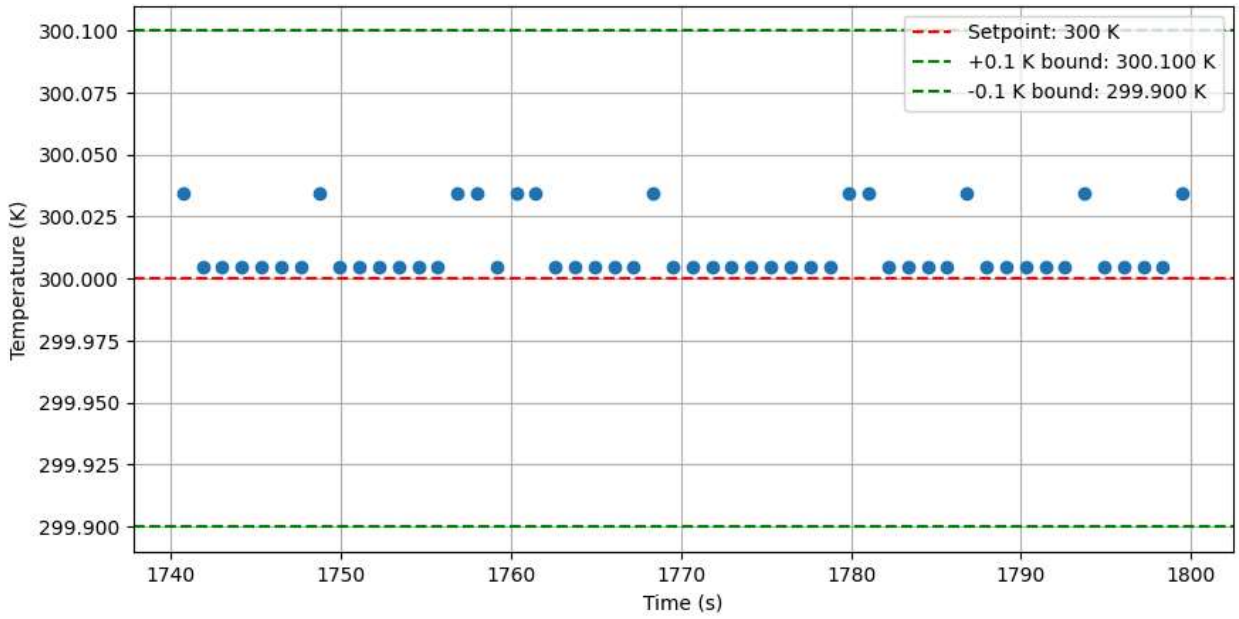

Figure 6: Thermistor 30 minute response, zoomed in on the 29 - 30 minute window. Setpoint = 300°K

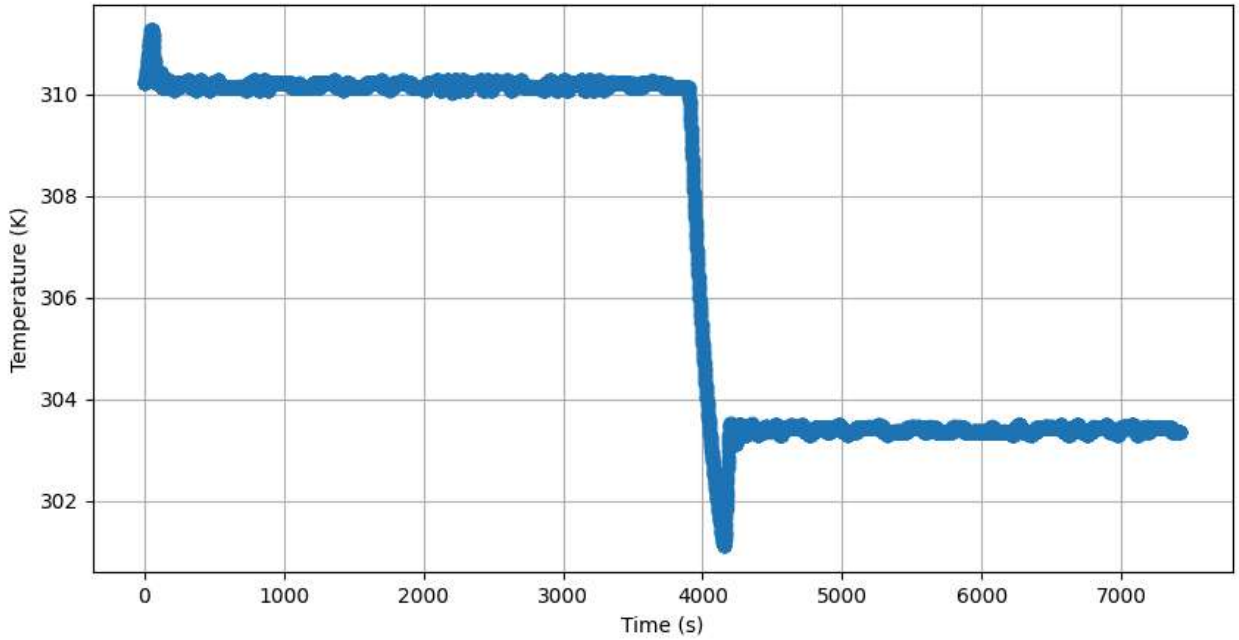

Figure 7 : Thermistor 2-hour response. Hour 1 setpoint: 310.161°K. Hour 2 setpoint: 303.388 °K

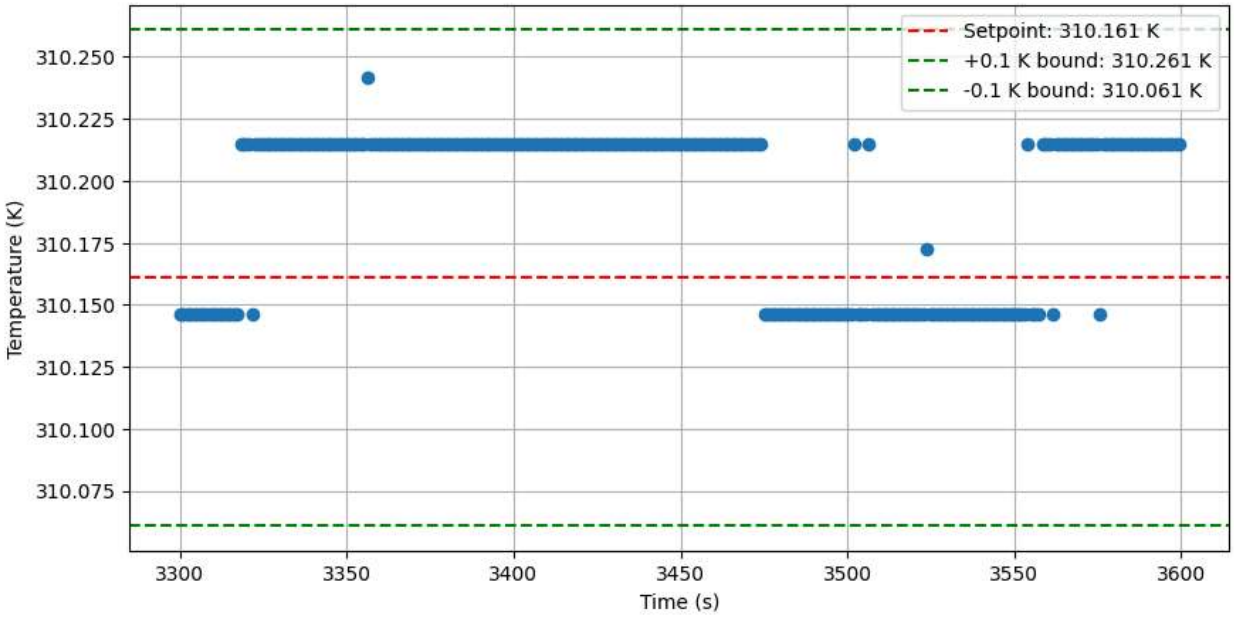

Figure 8 : Thermistor 2-hour response, zoomed in at the 55-60 minute window. Setpoint is 310.161°K.

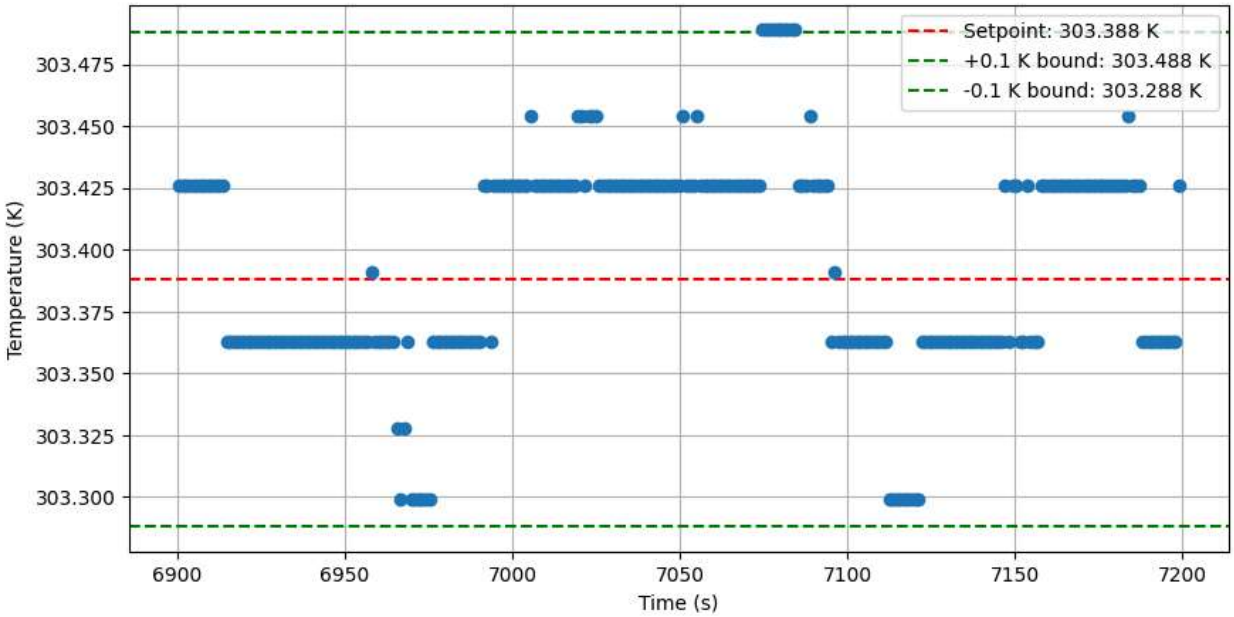

Figure 9: Thermistor 2-hour response, zoomed in at the 115-120 minute window. Setpoint is 303.388°K.

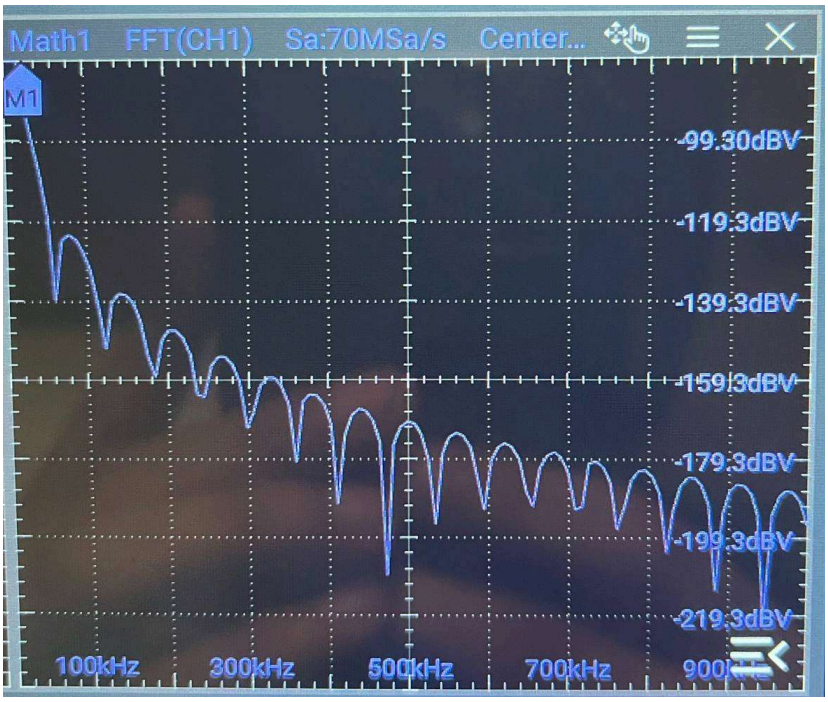

Figure 10: FFT Response of 6V Net at 1MHz range.

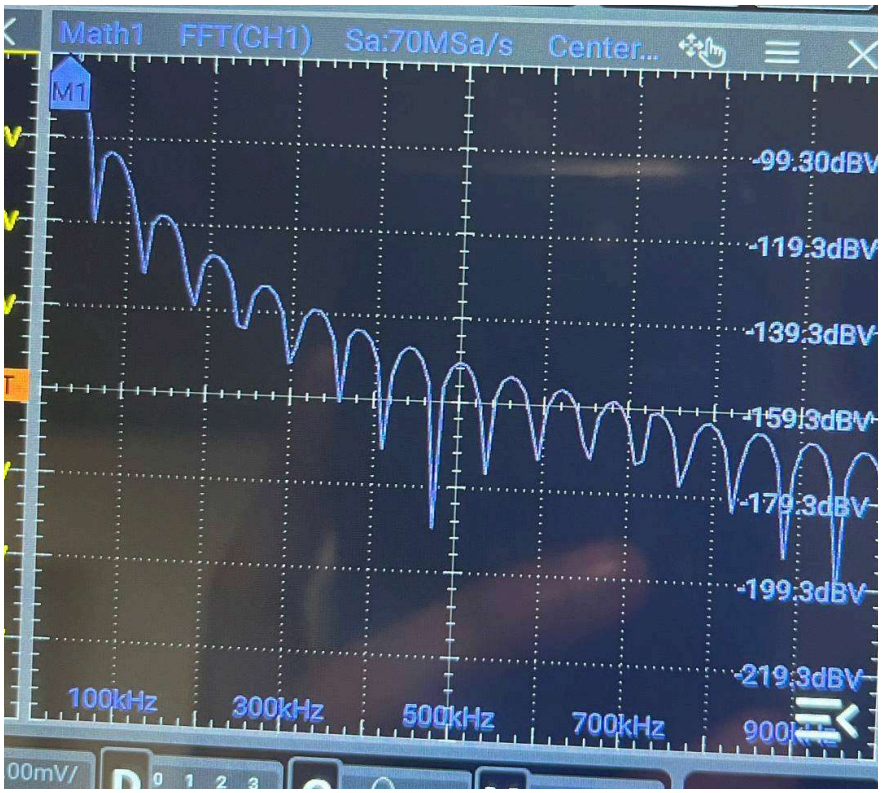

Figure 11: FFT response of 5V Net at 1MHz range

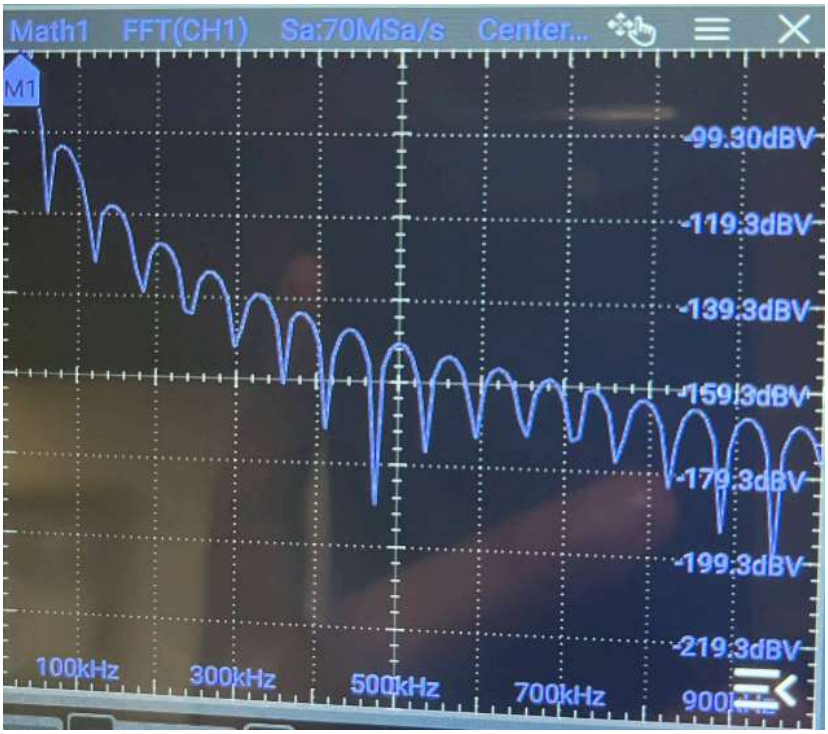

Figure 12: FFT Response of 3.3V Net at 1MHz range

Analysis

Thermistor Response

From our data, we can conclude that our tuned PID controller reaches our requirements of < 0.1°K accuracy within 10 minutes and <0.1°K stability within 30 minutes at a 300°K setpoint. We were also able to hit the 75% and 25% max temperature setpoints in the two hour test with the same accuracy and stability. Note that for most graphs, at steady state the PID the temperature still oscillates between two values, e.g. for 300°K test, between 300.034°K and 300.06°K. From printing the direct bit values of the ADC readings in testing, we realized that these temperatures matched to two ADC values off by 1 bit. This suggests that the oscillation could be caused by quantization, e.g. the true temperature lies in between the two. We noticed that this means the maximum resolution for our controller is 60 mK per bit in the ADC. In the two hour test with setpoint 303.388°K, the temperature oscillates almost by +-180mV, which is 3 bits in each direction. This could be caused by small sensor or temperature noise inducing the controller to react, despite us having a deadband implemented in the controller. There could also be some circuit noise changing the splitter excitation voltage, which is measured by the ADC to calculate the temperature. We can also consider that the DAC resolution is only 12 bits, meaning that if we wanted to obtain a desired temperature and that directly corresponds to a specific voltage output, we may not be able to exactly output that voltage due to digital conversion loss.

FFT and Noise

From our FFT, we can see that there is a potential switching artifact roughly at ~500 KHz as shown by the sharp peak in all three graphs. Since we are using the TPS62932DRLR buck converter as the first stage in the regulator, we will have some switching frequency. The Altium schematic shows that we have left the RT pin that sets the switching frequency floating, and the TPS62932 data sheet shows that the switching frequency with RT floating is about ~505 KHz at ambient temperature. We see this artifact also in the 5V and 3V3 regulators because they use the 6V output from the buck converter.

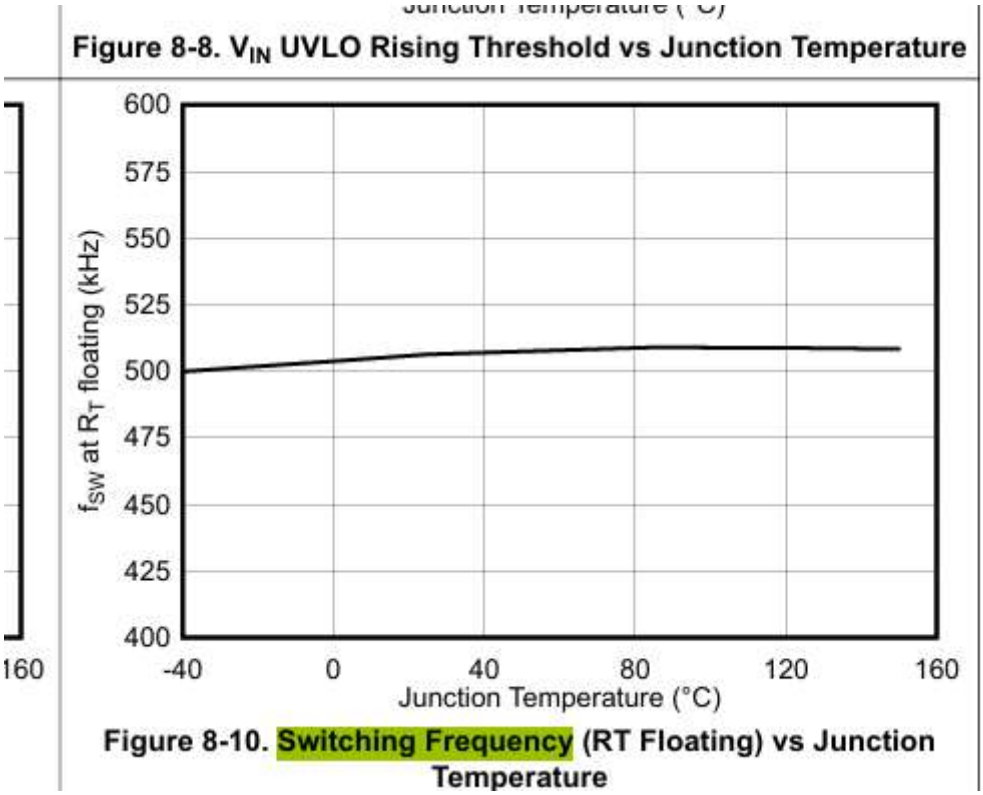

Figure 13: TPS62932 switching frequency vs temperature

Conclusion and Summary

In this lab, we were able to design and tune a heater driver circuit to regulate the temperature of a thermistor to the desired specifications of ≤ 0.1°K from the setpoint with a 30 min stability of better than 0.1°K. We first solve for component values to determine the op-amp gain and the current limiting resistor to switch the BJT on and off. Since we wanted our circuit to behave as a low frequency control loop, we also designed to ensure the RC time constant was on order of seconds.

We then populated our PCB with components. However, due to component shortages, we could not finish the PCB population and instead used the TA board. We first tested the voltage regulators to ensure their accuracy and then wrote python code to test the ADC and DAC units. After, we ran a test to get the maximum and minimum temperatures of our thermistor in our ambient room temperature. We then wrote a PID controller to regulate the temperature of the thermistor to a setpoint, and implemented extra features such as a deadband region and integral bounds. We conducted a 30 minute test for stability and then a 2 hour test where the setpoints changed each hour. From our data, we can see that we were able to achieve the design

specifications. Our analysis showed that although we achieved design specifications, the oscillations we saw in our temperature control could possibly be to DAC/ADC quantization and data loss.

Lastly, we performed FFTs on all three voltage regulator nets to validate the spectral purity. We noticed that there was a potential ~500 kHz switching artifact, which most likely comes from the buck converter TPS62932DRLR as quoted in the data sheet.

Some further measurements we can do is to further verify the ADC/DAC resolution. Since the 60 mK per bit resolution value is derived from our PID test only, we can run a test to step the DAC output bit by bit and record the output voltage. We can then calculate the change in temperature per change in voltage step to find the true resolution from ADC/DAC bits to temperature. We can also compare the output voltage we read from the ADC with a voltmeter to measure the data loss (should be extremely low since 12 bit resolution is still large). To confirm the switching artifact, we can try setting the RT pin of the buck converter to get a desired frequency and see if the artifact appears in FFTs at that frequency. Lastly, we can try running the experiment using the component values we designed and calculated and assess our controller performance relative to the TA board.

Appendix

See this github link for Python code and .csv files.