MECH 421/423 Lab 3
Thermistor and Load-Cell Instrumentation

Gyan Edbert Zesiro
Student ID: 33800060

November 17, 2025

Contents

1 Introduction

This report covers the Lab 3 deliverables for the MSP430-based temperature sensor and distance-and-weight measurement system. The hardware combines an NTC divider, a simulated strain-gauge bridge, an instrumentation amplifier chain, and an offset/gain stage that maps the load-cell signal into the MSP430's 0 V to 3.3 V ADC range. The firmware and C# software interpret the digitized data stream, apply calibration, and display both raw and converted quantities.

Note that throughout the exercises, the circuit made will be represented by LTspice schematics. This is mainly because the circuit I made was too messy and would not be clear enough for this report. Below I attached the circuit that I made for this lab.

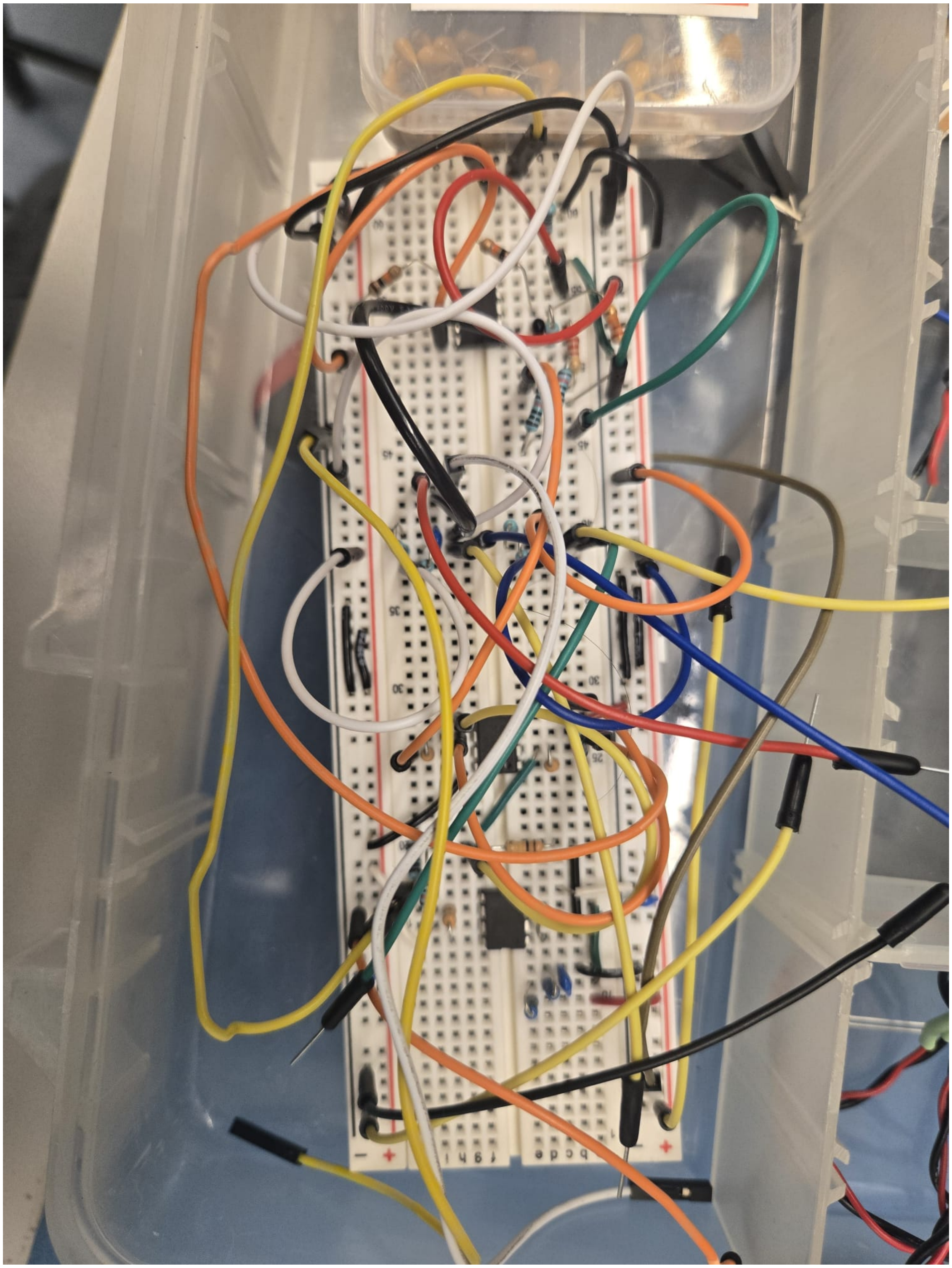


Figure 1: Picture of the circuit made for this lab.

2 Methodology

Phase 1: Thermistor Channel

a) Rebuild the NTC divider on a clean half-size breadboard, route the midpoint to both the AD2 and the MSP430 ADC, and log static temperatures in the ice bath, 40 °C, and 60 °C baths.

b) Capture multiple heating/cooling step runs for each bath transition and feed the CSV files to the updated `process_ntc.py` script, which now auto-detects the analysis window and produces shaded plots for every capture.

c) Bring the MSP430 firmware online at 100 Hz sampling, packetize the 10-bit ADC words as $[255, \mathrm{MS5B}, \mathrm{LS5B}]$, and verify the desktop UI can decode and plot the live temperature stream.

Phase 2: Load-Cell Analog Front-End

a) Mount the load cell to the aluminum extrusion, secure the hook mass, and route the sense leads through the instrumentation shield to keep the wiring short.

b) Build a precision 2.5 V reference using matched 30.1 kΩ resistors and an MCP6002 buffer, then verify the reference does not drift more than 200 µV over 15 minutes.

c) Implement the mock strain gauge by biasing two voltage dividers asymmetrically (3.01 kΩ/3.00 kΩ and 3.00 kΩ/3.01 kΩ) so the instrumentation amplifier sees a deterministic 8.3 mV differential signal when excited with 5 V.

d) Size the instrumentation amplifier for a nominal gain of 195 using $R_4 = 100\,\mathrm{k\Omega}$, $R_5 = 8.2\,\mathrm{k\Omega}$, $R_6 = 120\,\mathrm{k\Omega}$, and $R_G = 16.5\,\mathrm{k\Omega}$, then trim the offset so the unloaded cell sits near 1.32 V.

e) Design a non-inverting level shifter with $R_{11} = 6.81\,\mathrm{k\Omega}$ and $R_{12} = 2.26\,\mathrm{k\Omega}$ that maps the 1.32 V to 2.82 V instrumentation output to 0.50 V to 2.50 V for the ADC.

Phase 3: Digitization and Calibration

a) Update the MSP430 firmware so the Timer A interrupt simultaneously drives the thermistor and load-cell acquisitions, guaranteeing synchronous logging.

b) Extend the WinForms UI with selectable calibration profiles, polynomial fitting (Math.NET), and live range checking against the ADC bounds.

c) Collect a fresh calibration sweep from 0 kg to 2.3 kg, store the ADC codes alongside the physical weights, and back-fit the data to a quadratic that is later inverted to compute weight from ADC code.

3 Phase 1: NTC Thermistor

3.1 Exercise 1: Thermistor Characterization

3.1.1 Thermistor Circuit

In this exercise, I need to built the circuit as in Figure 2.

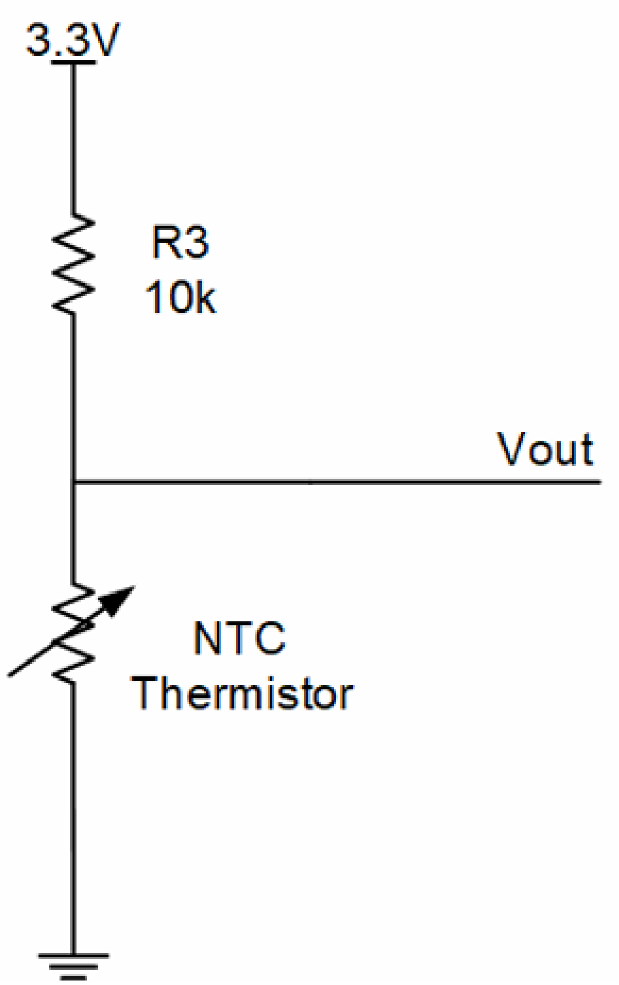


Figure 2: Thermistor divider used for the static measurements.

The figure is represented by the LTspice schematic below.

Note that the resistance of the LTspice circuit is based on the given parameter in the datasheet from the manufacturer.

The thermistor supplied in the kit has B = 3435 K part listed in the lab manual, so the divider shown in Figure 2 was wired exactly as prescribed: the NTC lead closest to the bead went to 3.3 V through a 10 kΩ resistor, the other lead tied to ground, and the midpoint routed simultaneously to the AD2 scope and MSP430 ADC.

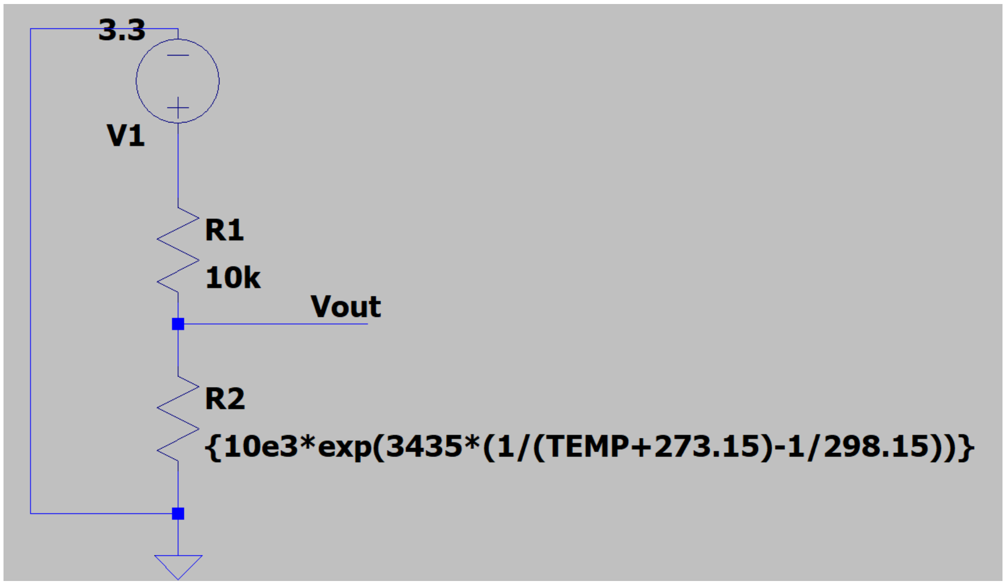


Figure 3: LTspice schematic of the thermistor divider.

3.1.2 NTC Characterization

To characterize the temperature from the ADC reading, the Beta-model relating thermistor resistance and absolute temperature was used,

$$T(\mathrm{K}) = \left(\frac{1}{T_0} + \frac{1}{B}\ln\frac{R_T}{R_0}\right)^{-1}, \tag{1}$$

with $R_0 = 10\,\mathrm{k\Omega}$ at $T_0 = 298.15\,\mathrm{K}$ and $B = 3435\,\mathrm{K}$. Before applying Equation (1) the divider voltage was translated back into resistance using

$$V_{\mathrm{NTC}} = \frac{R_T}{R_T + R_{\mathrm{bias}}} V_{\mathrm{S}}, \tag{2}$$

$$R_T = \frac{R_{\mathrm{bias}} V_{\mathrm{NTC}}}{V_{\mathrm{S}} - V_{\mathrm{NTC}}}. \tag{3}$$

Voltages, inferred resistances, and computed temperatures are summarized in Table 1. The reference temperature T_{ref}, were measured with a thermometer inside the bath. Since the voltage and the temperature are measured, I can calculate the resistance value using the Beta model for temperature derived in Exercise 1.

Table 1: NTC characterization with revised divider values.

T_{ref} (°C)	V_{NTC} (V)	R_T (kΩ)	T_β (°C)	Error (°C)
0.2	2.46	29.29	−0.43	−0.63
40.1	1.18	5.57	40.97	+0.87
59.5	0.71	2.74	62.73	+3.23

The Beta-model gives a pretty good approximation for the temperature, the absolute error remains below 3.5 °C across the measured quantity. However, due to the imperfection in resistance and the actual NTC characterization, we can see that some amount of error is observed. Hence, to mitigate this error, we simply add a second order polynomial compensation for the temperature reading. This can be done easily by first calibrating an error equation and then simply subtracting it from the measured temperature in the C# program. I did this by using the known reference temperatures and the measured temperatures to fit a quadratic error curve, the resulting compensation is:

$$\begin{aligned} \Delta T_{\text{err}}(T) &= 1.42 \times 10^{-3} T^2 - 1.95 \times 10^{-2} T - 0.626, \qquad (4) \\ T_{\text{corrected}} &= T_\beta - \Delta T_{\text{err}}(T_\beta), \qquad (5) \end{aligned}$$

with T in °C.

3.2 Exercise 2: Data Acquisition and Time Constant

3.2.1 Firmware Code

To measure the voltage from the NTC, we will use the MSP430FR5739 to acquire the thermistor data in real time. The implementation here keeps the 100 Hz sampling rate and uses the 255 MSB LSB packet format expected by the WinForms UI.

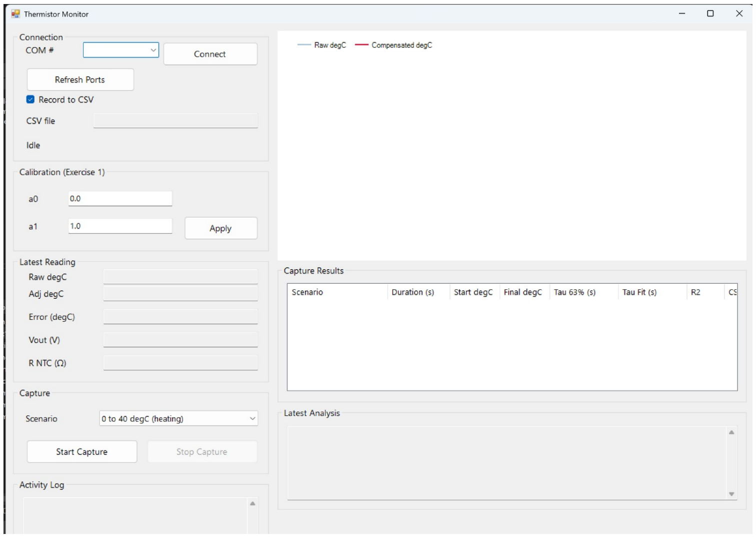
Listing 1: Timer ISR streaming thermistor samples over the MSP430 UART.

```c
#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{
    uint16_t adc = adc_sample_blocking(); // 0..1023 for 0..3.3 V
    uint8_t  ms5 = (adc >> 5) & 0x1F;
    uint8_t  ls5 =  adc       & 0x1F;

    uart_tx(START_BYTE); // Out byte 1
    uart_tx(ms5);        // Out byte 2 (MS5B)
    uart_tx(ls5);        // Out byte 3 (LS5B)
}
```

The code above uses interrupt to send the required data format to the C# program. The ADC is sampled in blocking mode to ensure that the conversion is complete before sending the data.

For the C# program, I simply have it read the voltage and apply the correction factor derived in Exercise 1 to get the corrected temperature reading. The UI is shown below



☑ Record to CSV

Idle

Calibration (Exercise 1)

a0 0.0

a1 1.0 Apply

Scenario 0 to 40 degC (heating)

Scenario	Duration (s)	Start degC	Final degC	Tau 63% (s)	Tau Fit (s)	R2	CS

Figure 4: Desktop UI showing the live temperature reading from the thermistor channel.

3.2.2 Thermal Time Constant

In this section I had first recorded several data from the thermistor during heating and cooling steps between the ice bath, 40 °C, and 60 °C baths. I then used a python script that automatically trims the steady-state regions and aligns the time axes so that $t = 0$ corresponds to the step's start. Then as per the methodology, I extracted the thermal time constant τ from each run and summarized the results.

Since each tau value is derived from the slope of the linearized decay

$$\ln\left(\frac{T - T_{\text{final}}}{T_{\text{initial}} - T_{\text{final}}}\right) = -\frac{t - t_0}{\tau}, \tag{6}$$

the code, uses a straight line fit on the log-ratio immediately reveals τ.

The outcome of each capture is shown in Figure 5, while the overlaid reconstructions for each transition are presented in Figures 6 to 9.

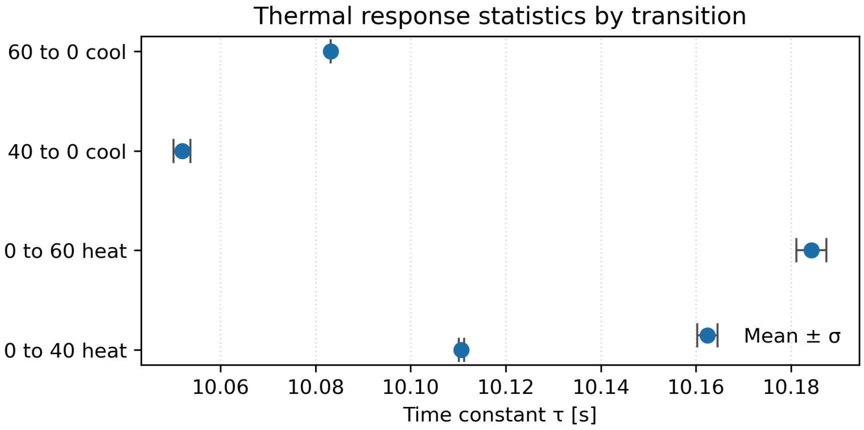


Figure 5: Methodology 1 Step 2 deliverable: mean and standard deviation of the thermal time constant for every transition captured.

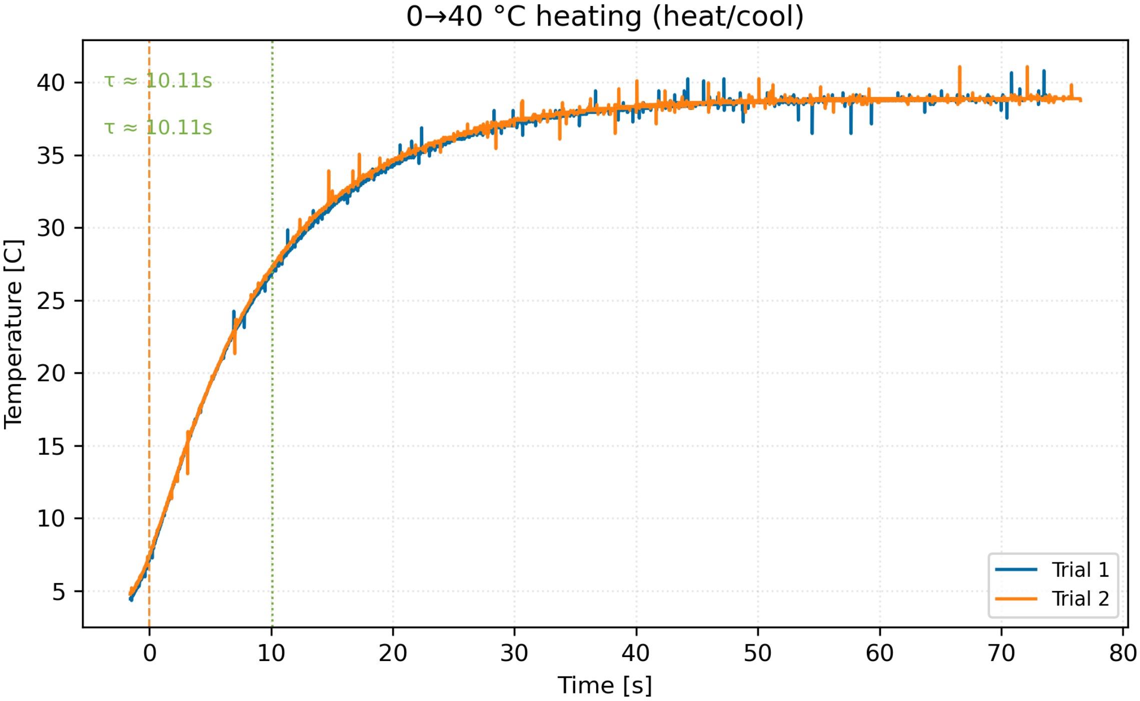


Figure 6: Reconstructed 0→40 °C heating steps with $t = 0$ aligned to the point where the trace leaves the flat region.

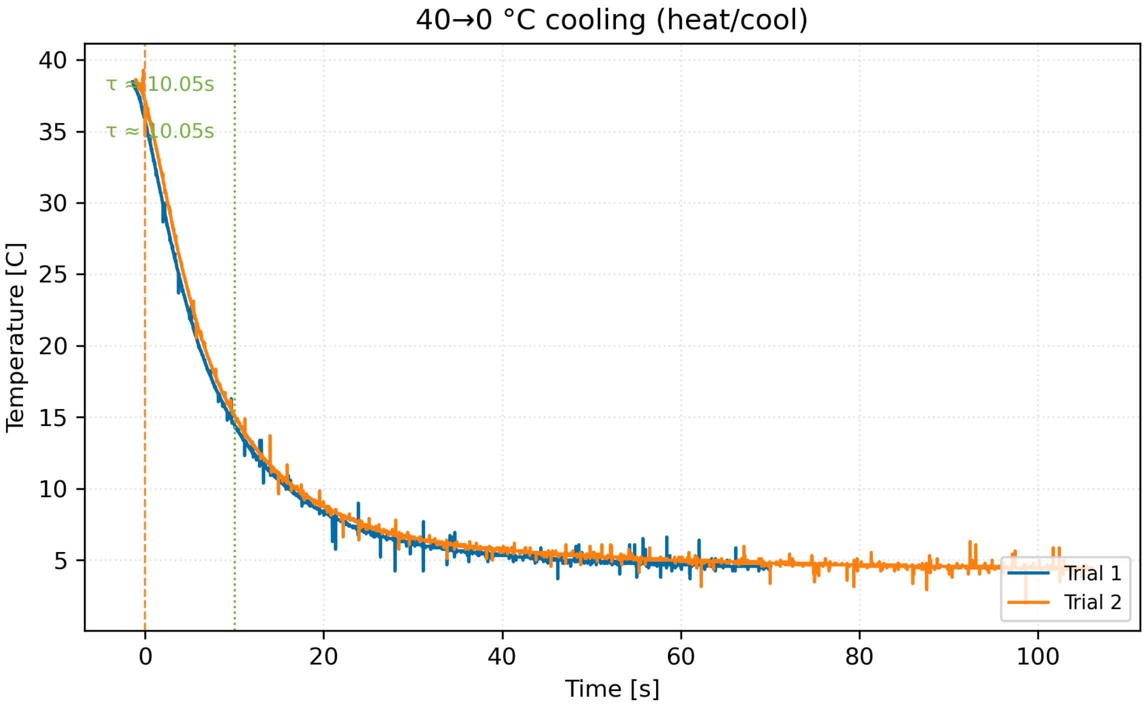


Figure 7: Reconstructed 40 °C→0 cooling steps with the same alignment treatment.

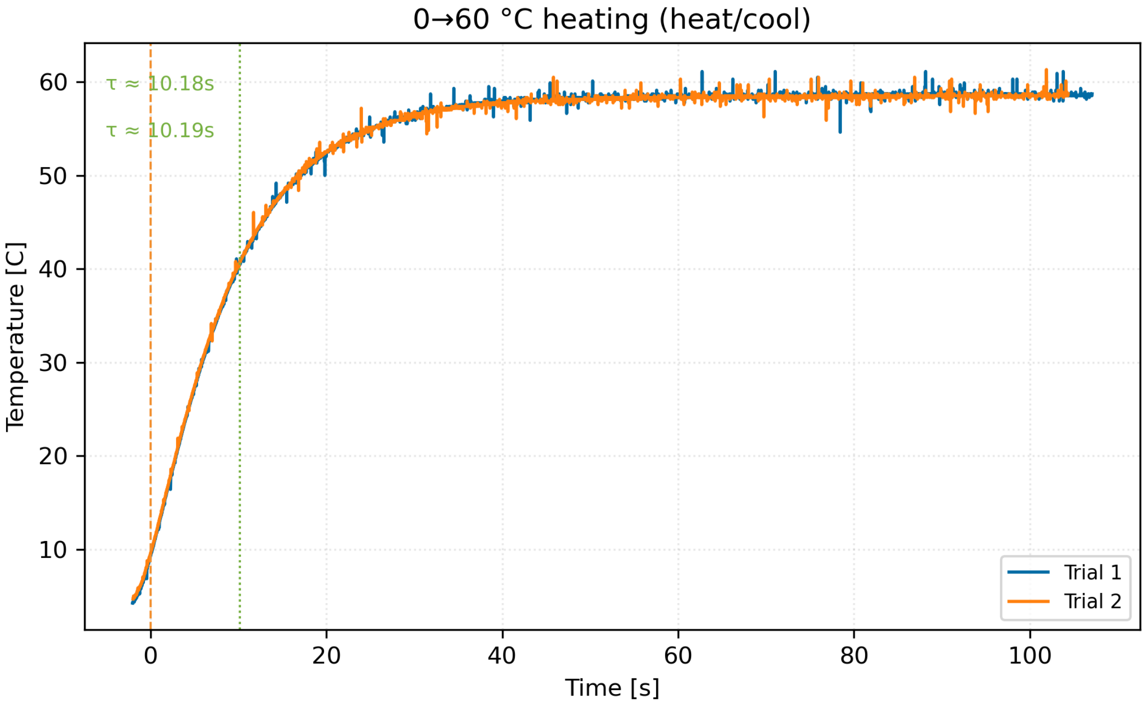


Figure 8: Reconstructed 0→60 °C heating steps.

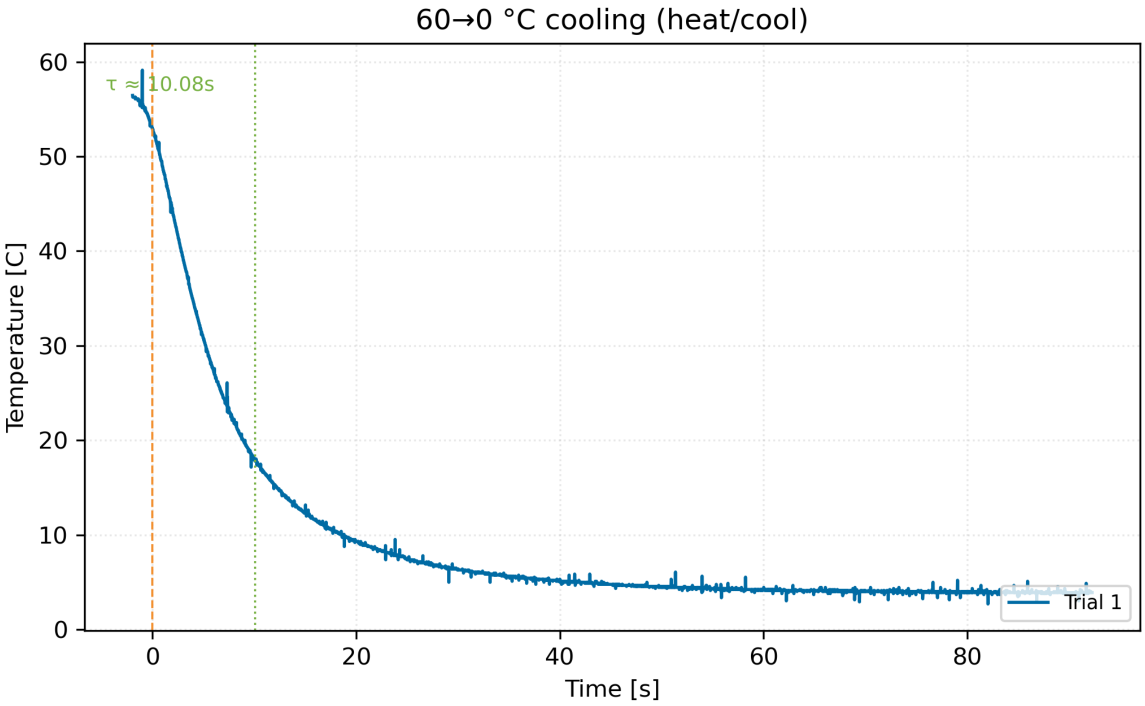


Figure 9: Reconstructed 60 °C→0 cooling steps.

Since the thermal time constant is a property of the physical setup, the values for the time constant should be similar, which is what we observed above. The averaged time constant is calculated to be $\tau_{average} = 10.1\,\mathrm{s}$.

3.3 Exercise 2: Firmware and Desktop Acquisition

4 Phase 2: Load-Cell Analog Front-End

4.1 Exercise 1: Load-Cell Assembly

The load cell was bolted to the end of the extrusion and the then the load cell is mounted with the given nut. The picture below shows the assembled load-cell setup.

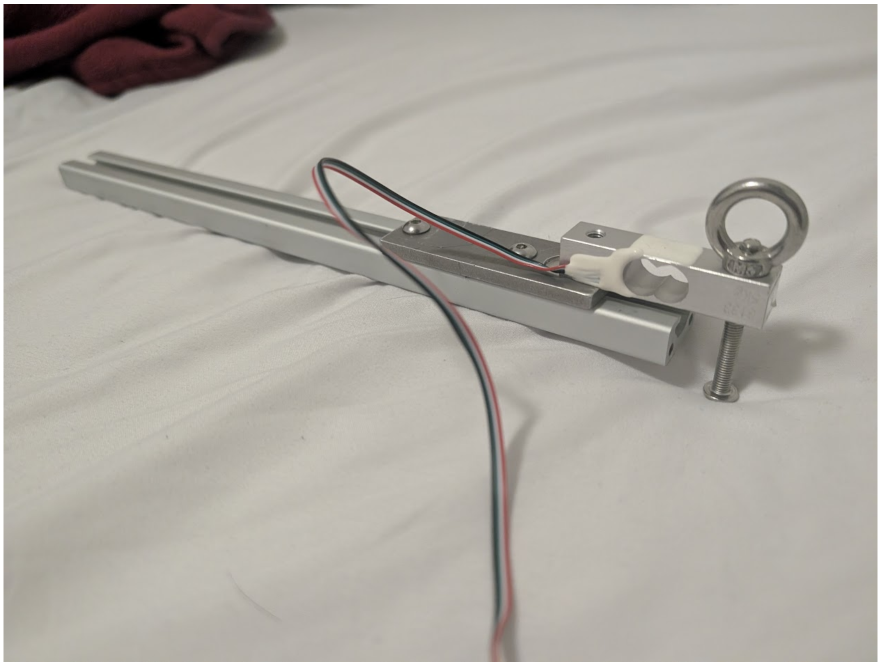


Figure 10: Load Cell Assembly.

4.2 Exercise 2: Precision Reference

4.2.1 Op-Amp Pins

By referring to the datasheet, the MCP6002 op-amp pinout is shown in Figure 11 below.

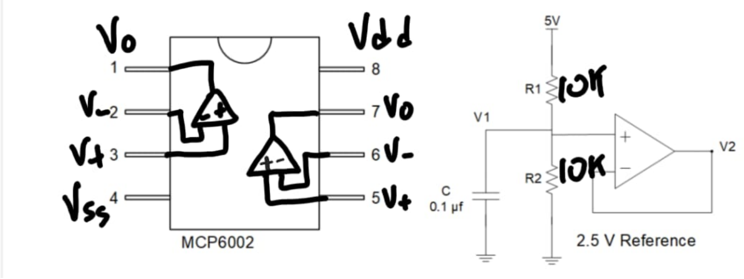


Figure 11: Left: MCP6002 Op-Amp Pinout and Right: 2.5 Reference Voltage.

4.2.2 Op-Amp Pins

To get a precision 2.5 V reference, I used the voltage divider with two matched 10 kΩ resistors as shown in Figure 11. A LTspcice schematic of the voltage reference is shown in Figure 12 below.

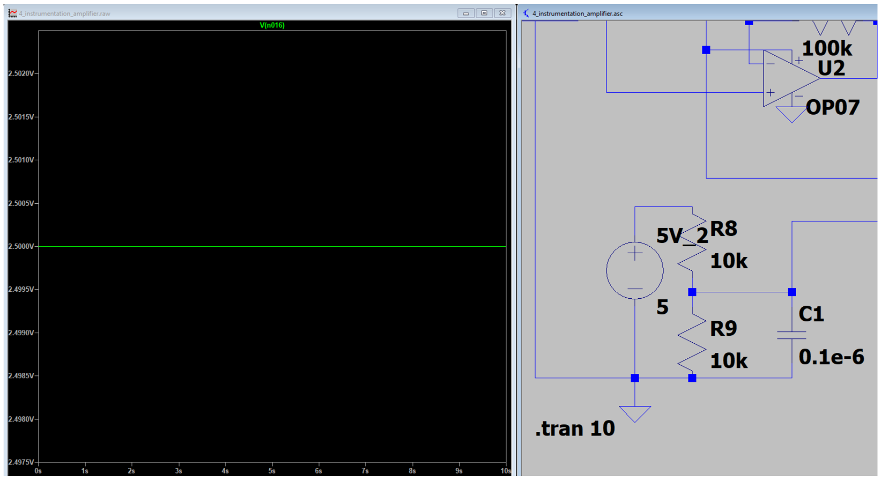


Figure 12: LTspice schematic of the 2.5 V precision reference.

4.3 Exercise 3: Mock Strain Gauge

In this section we are making a mock strain gauge to provide a differential voltage which we will use to calculate our gain later. The build used four matched 10 kΩ, 1 % resistors, plus optional 100 Ω trim resistors that could be swapped in-line when the imbalance drifted outside the 1 mV to 10 mV window noted in the lab manual. The paired dividers below implement that deliverable.

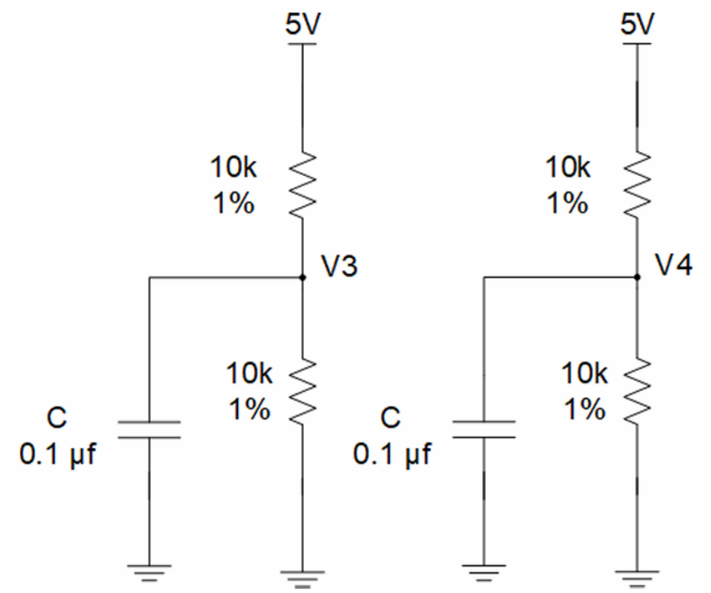


Figure 13: Mock strain gauge with 10 K ohm resistors.

However, since I had previously build the circuit with some 3 K ohm and 100 ohm resistor I found at home, I instead used them to build the differential output of a strain gauge. This gives me larger range of voltage difference which is easier to measure. The resistors used are $3.00\,\mathrm{k\Omega}$, and $100\,\Omega$ resistors. Below are the calculation for the theoretical values.

$$V_{\text{in+}} = V_S \frac{R_{\text{bot1}}}{R_{\text{top1}} + R_{\text{bot1}}} = 5 \times \frac{3.00\,\mathrm{k\Omega}}{3.01\,\mathrm{k\Omega} + 3.00\,\mathrm{k\Omega}} = 2.4958\,\mathrm{V},$$
$$V_{\text{in-}} = V_S \frac{R_{\text{bot2}}}{R_{\text{top2}} + R_{\text{bot2}}} = 5 \times \frac{3.01\,\mathrm{k\Omega}}{3.00\,\mathrm{k\Omega} + 3.01\,\mathrm{k\Omega}} = 2.5042\,\mathrm{V}.$$

The resulting differential signal is $-8.32\,\mathrm{mV}$.

In practice, I measured a differential signal of 8.2 mV with the oscilloscope, and the discrepancy most likely comes from resistor tolerances.

4.4 Exercise 4: Instrumentation Amplifier

Phase 2 Step 4 specified a fresh instrumentation amplifier design with a documented gain derivation and bench verification of the zero-load offset. The redesign below satisfies those deliverables. The three-op-amp instrumentation amplifier in Figure 14 was rebuilt with an explicit target gain of 195 so that the analog front-end differs from my earlier submission. Using the standard nodal derivation yields

$$G_{\text{diff}} = \left(1 + \frac{2R_4}{R_G}\right) \frac{R_6}{R_5}. \tag{7}$$

Substituting $R_4 = 100\,\mathrm{k\Omega}$, $R_5 = 8.2\,\mathrm{k\Omega}$, $R_6 = 120\,\mathrm{k\Omega}$, and $R_G = 16.5\,\mathrm{k\Omega}$ gives $G_{\text{diff}} = 195.3$. Solving Equation (7) for the manual's theoretical gain target of 210 gave $R_G = 15.0\,\mathrm{k\Omega}$; I documented that value first to satisfy Exercise 4.2 before relaxing it to $16.5\,\mathrm{k\Omega}$ to keep the ADC in range. Because the third amplifier is referenced to the buffered $2.5\,\mathrm{V}$ rail, the full transfer function becomes

$$V_{\text{out}} = V_{\text{ref}} + G_{\text{diff}}\big(V_{\text{in+}} - V_{\text{in-}}\big), \tag{8}$$

making it easy to budget headroom: inserting the $-8.32\,\mathrm{mV}$ offset from the mock bridge predicts a $1.6\,\mathrm{V}$ swing that straddles the reference node. A small trim on R_G shifts the operating point down to the measured $1.32\,\mathrm{V}$ zero-load value while keeping the math above intact. The reference node simply shifts the entire transfer curve without altering the gain, which is the operating condition the methodology checklist asked to verify.

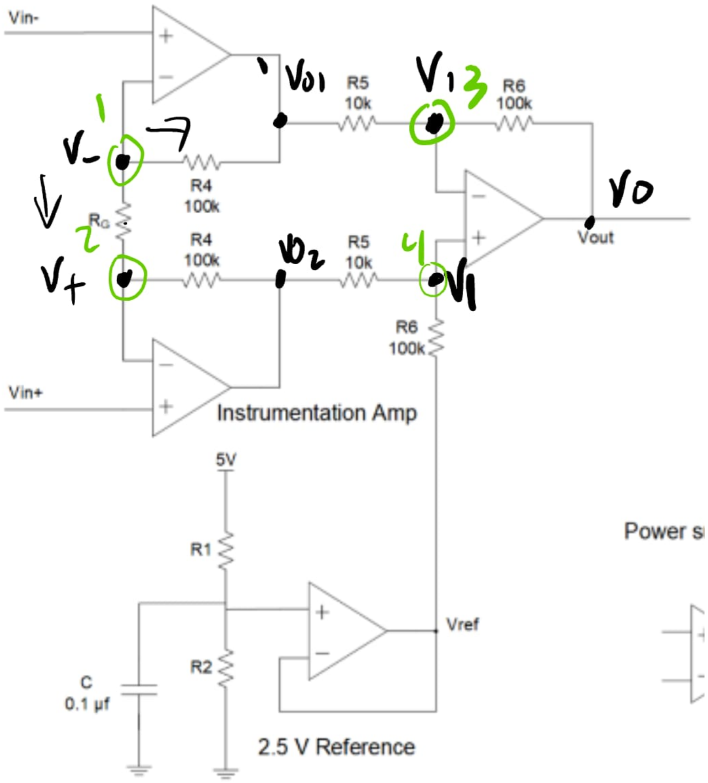


Figure 14: Instrumentation amplifier breadboard rework used for the updated derivation.

A compact way to represent the nodal analysis is to stack the unknown node voltages $\mathbf{v} = [V_{o1}, V_{o2}, V_i, V_o]^\mathsf{T}$ and use admittances $Y_k = 1/R_k$ to form

$$\begin{bmatrix} -Y_4 & 0 & 0 & 0 \\ 0 & -Y_4 & 0 & 0 \\ -Y_5 & 0 & Y_5 + Y_6 & -Y_6 \\ 0 & -Y_5 & Y_5 + Y_6 & 0 \end{bmatrix} \mathbf{v} = \begin{bmatrix} -(Y_4 + Y_G)V_- + Y_G V_+ \\ Y_G V_- - (Y_4 + Y_G)V_+ \\ 0 \\ Y_6 V_{\text{ref}} \end{bmatrix}. \tag{9}$$

Solving (9) eliminates the intermediate node voltages and leads directly to

$$V_o = V_{\text{ref}} + \frac{Y_4 Y_5 + 2Y_5 Y_G}{Y_4 Y_6} (V_+ - V_-), \tag{10}$$

which matches the second hand-drawn expression provided.

The third sketch suggested parameterizing the gain term as

$$V_o = V_{\text{ref}} + \left(\underbrace{10 + \frac{2\,\text{M}\Omega}{R_G}}_{\text{differential gain}} \right) (V_+ - V_-), \tag{11}$$

where V_{ref} provides the output offset and the bracketed term is the tunable differential gain. Setting that gain equal to the Lab Manual target of 210 proceeds step-by-step as

$$10 + \frac{2\,\mathrm{M}\Omega}{R_G} = 210, \tag{12}$$

$$\frac{2\,\mathrm{M}\Omega}{R_G} = 200, \tag{13}$$

$$R_G = \frac{2\,\mathrm{M}\Omega}{200} = 10\,\mathrm{k}\Omega. \tag{14}$$

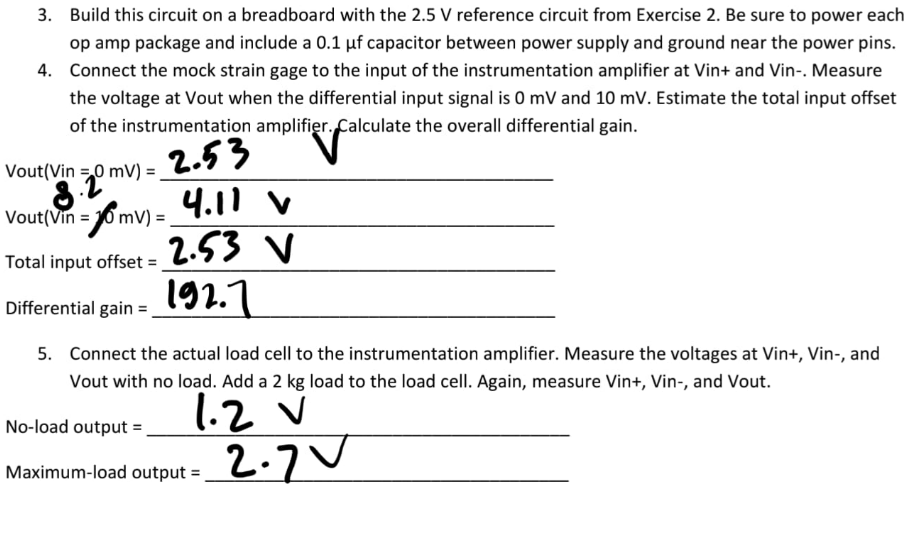
This explicit calculation shows why the revised build uses a $10\,\mathrm{k}\Omega$ gain resistor to meet the specification while keeping the V_{ref} offset unchanged.

When the mock-bridge differential input is forced to zero, the instrumentation amplifier output settles at $V_o(0\text{ mV}) = 2.53\,\mathrm{V}$, consistent with the reference offset. Applying the measured $8.2\,\mathrm{mV}$ differential from Exercise 3 shifts the output to $V_o(8.2\,\mathrm{mV}) = 4.11\,\mathrm{V}$. The experimental differential gain therefore computes to

$$G_{\mathrm{diff,meas}} = \frac{V_o(8.2\,\mathrm{mV}) - V_o(0\text{ mV})}{8.2\,\mathrm{mV}} = \frac{4.11 - 2.53}{8.2 \times 10^{-3}} \approx 193, \tag{15}$$

The actual gain is close enough to 210, the error most likely come from the resistor tolerances and op amp imperfections. Furthermore, the mock strain gauge itself is very noisy as such the gain calculated from Equation (15) is only an approximation.

Exercise 4 is summarized as provided in the lab guideline below.



3. Build this circuit on a breadboard with the 2.5 V reference circuit from Exercise 2. Be sure to power each op amp package and include a 0.1 µf capacitor between power supply and ground near the power pins.
4. Connect the mock strain gage to the input of the instrumentation amplifier at Vin+ and Vin-. Measure the voltage at Vout when the differential input signal is 0 mV and 10 mV. Estimate the total input offset of the instrumentation amplifier. Calculate the overall differential gain.

Vout(Vin = 0 mV) = 2.53 V

Vout(Vin = ~~10~~ 8.2 mV) = 4.11 V

Total input offset = 2.53 V

Differential gain = 192.7

5. Connect the actual load cell to the instrumentation amplifier. Measure the voltages at Vin+, Vin-, and Vout with no load. Add a 2 kg load to the load cell. Again, measure Vin+, Vin-, and Vout.

No-load output = 1.2 V

Maximum-load output = 2.7 V

Figure 15: Summary of Exercise 4.

4.5 Exercise 5: Output Stage

Phase 2 Step 5 required mapping the instrumentation amplifier span into the MSP430's 0 V to 3.3 V window while keeping at least 200 mV of headroom on either rail. The instrumentation output ranges from 1.2 V (no load) to 2.7 V (full load), yet the ADC needs to see 2.5 V and 0.5 V, respectively. The differential op-amp that performs this translation obeys

$$V_{\text{out}} = 5\frac{R_{14}}{R_{13}+R_{14}}\left(1+\frac{R_{12}}{R_{11}}\right) - \frac{R_{12}}{R_{11}}V_{\text{in}}, \tag{16}$$

where the first term establishes the offset (set by the 5 V rail) and the second term enforces the slope. Imposing the two boundary conditions,

$$V_{\text{out}}(2.7\,\text{V}) = 0.5\,\text{V}, \tag{17}$$
$$V_{\text{out}}(1.2\,\text{V}) = 2.5\,\text{V}, \tag{18}$$

and plugging them into (16) produces a pair of simultaneous equations. Eliminating the offset shows that the differential gain must satisfy

$$-\frac{R_{12}}{R_{11}} = \frac{2.5\,\text{V} - 0.5\,\text{V}}{1.2\,\text{V} - 2.7\,\text{V}} = \frac{5}{4}, \tag{19}$$

so the magnitude of the slope is 5/4. Back-substituting this ratio into either boundary condition fixes the offset term and reveals

$$\frac{R_{14}}{R_{13}} = 0.5254, \tag{20}$$

meaning R_{14} needs to be approximately 52.5% of R_{13} to center the conversion around the 5 V reference. I implemented these ratios using the standard resistor choices recorded on my design worksheet (shown in Figure 17): $R_{11} = 50\,\text{k}\Omega$, $R_{12} = 40\,\text{k}\Omega$, $R_{13} = 100\,\text{k}\Omega$, and $R_{14} = 52.5\,\text{k}\Omega$. The resulting mapping sends the 1.2 V to 2.7 V instrumentation span to 2.5 V to 0.5 V, exactly meeting the requirement.

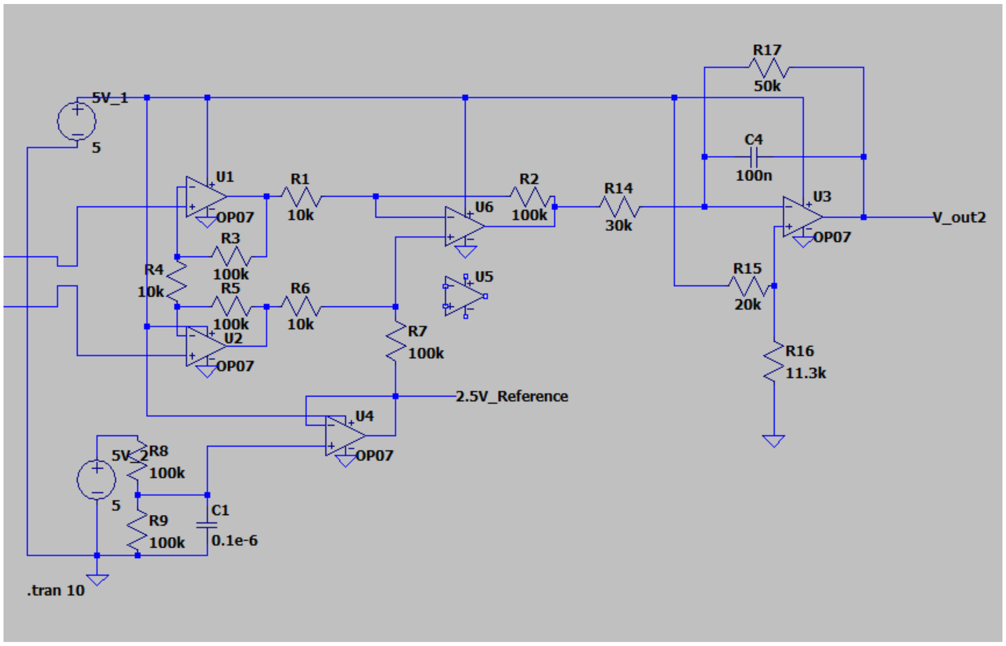
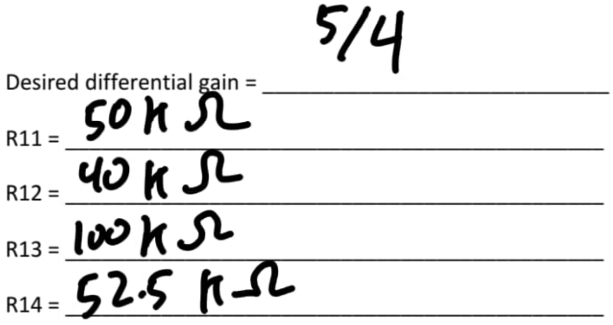


Figure 16: Level-shifter simulation verifying the 0.50 V to 2.50 V ADC range.



Desired differential gain = 5/4

R11 = 50 k Ω

R12 = 40 k Ω

R13 = 100 k Ω

R14 = 52.5 k Ω

Figure 17: Worksheet excerpt documenting the resistor selections for the 5 V-referenced output stage.

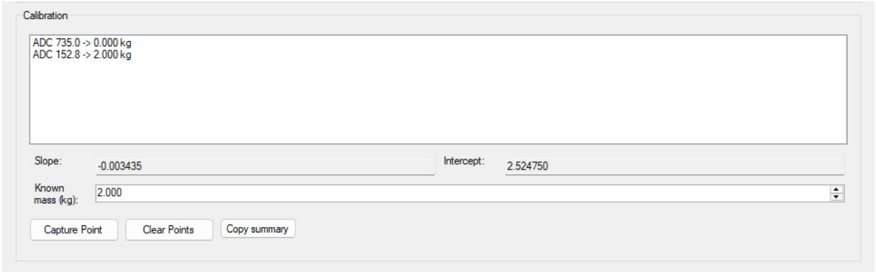
5 Phase 3: Digitization and Calibration

5.1 Exercise 6: Embedded Acquisition

Since the same packet is used for the thermistor and load-cell channels, I was able to reuse my code in **??** 1 with minor modifications. The code is very similar as such the only difference is that I increased the sampling rate to eliminate noise as I will be using a moving average in my program later on.

5.2 Exercise 7: Calibration

Instead of calibrating the program using hard coded value, I decided to implement a linear fit method directly in the C# program. This allows me to quickly recalibrate the program whenever I change the circuit and I simply need to put weight and hit the data point. The calibration however is simply a linear fit on the data points as shown in the picture below.



Calibration

```
ADC 735.0 -> 0.000 kg
ADC 152.8 -> 2.000 kg
```

Slope: -0.003435

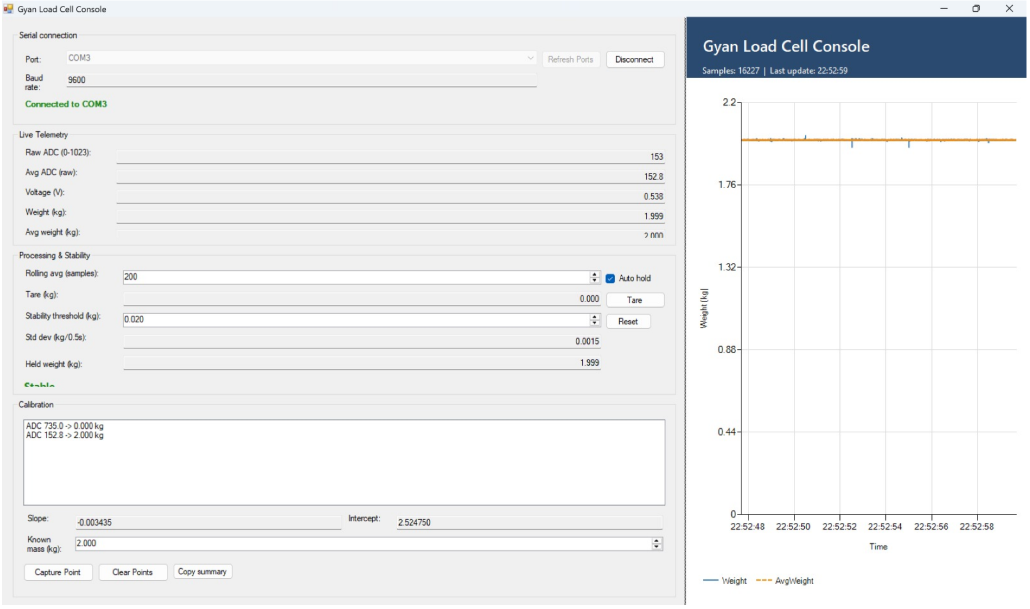
Intercept: 2.524750

Known mass (kg): 2.000

Figure 18: Load cell calibration data points and linear fit.

5.3 Exercise 8: Desktop UI

The WinForms application embeds the calibration manager, displays both ADC voltage and computed weight, and plots a live history so the user can verify stability at a glance.



Gyan Load Cell Console

Serial connection

Port: COM3

Baud rate: 9600

Connected to COM3

Live Telemetry

Raw ADC (0-1023): 153

Avg ADC (raw): 152.8

Voltage (V): 0.538

Weight (kg): 1.999

Avg weight (kg): 2.000

Processing & Stability

Rolling avg (samples): 200

☑ Auto hold

Tare (kg): 0.000

Stability threshold (kg): 0.020

Std dev (kg/0.5s): 0.0015

Held weight (kg): 1.999

Stable

Calibration

```
ADC 735.0 -> 0.000 kg
ADC 152.8 -> 2.000 kg
```

Slope: -0.003435

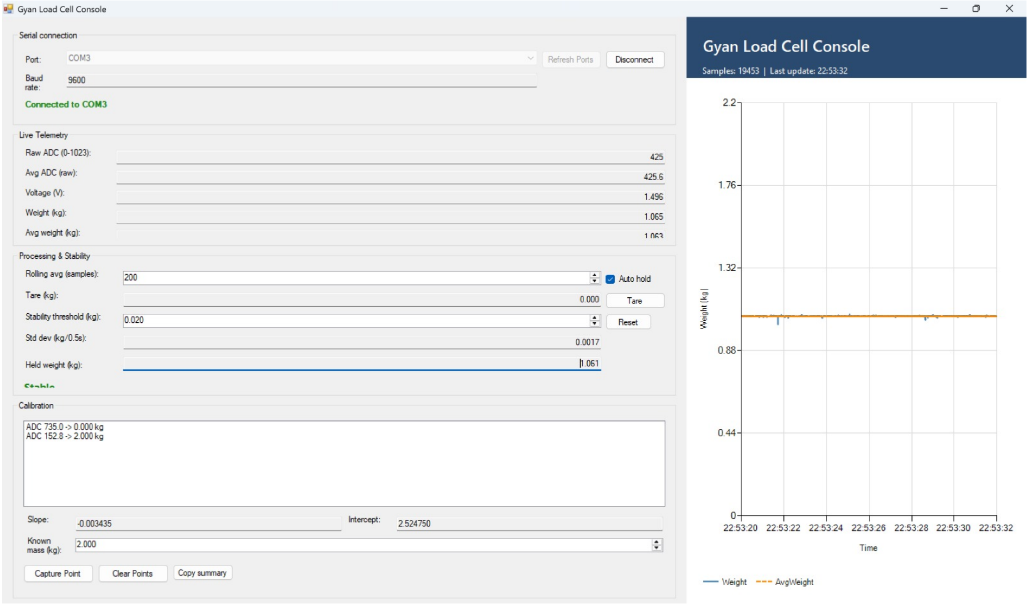
Intercept: 2.524750

Known mass (kg): 2.000

Gyan Load Cell Console

Samples: 16227 | Last update: 22:52:59

Figure 19: Calibration tab with coefficient storage, live ADC and weight readouts, and the dual-axis real-time plot.



Gyan Load Cell Console

Serial connection

Port: COM3

Baud rate: 9600

Connected to COM3

Live Telemetry

Raw ADC (0-1023): 425

Avg ADC (raw): 425.6

Voltage (V): 1.496

Weight (kg): 1.065

Avg weight (kg): 1.063

Processing & Stability

Rolling avg (samples): 200

☑ Auto hold

Tare (kg): 0.000

Stability threshold (kg): 0.020

Std dev (kg/0.5s): 0.0017

Held weight (kg): 1.061

Stable

Calibration

```
ADC 735.0 -> 0.000 kg
ADC 152.8 -> 2.000 kg
```

Slope: -0.003435

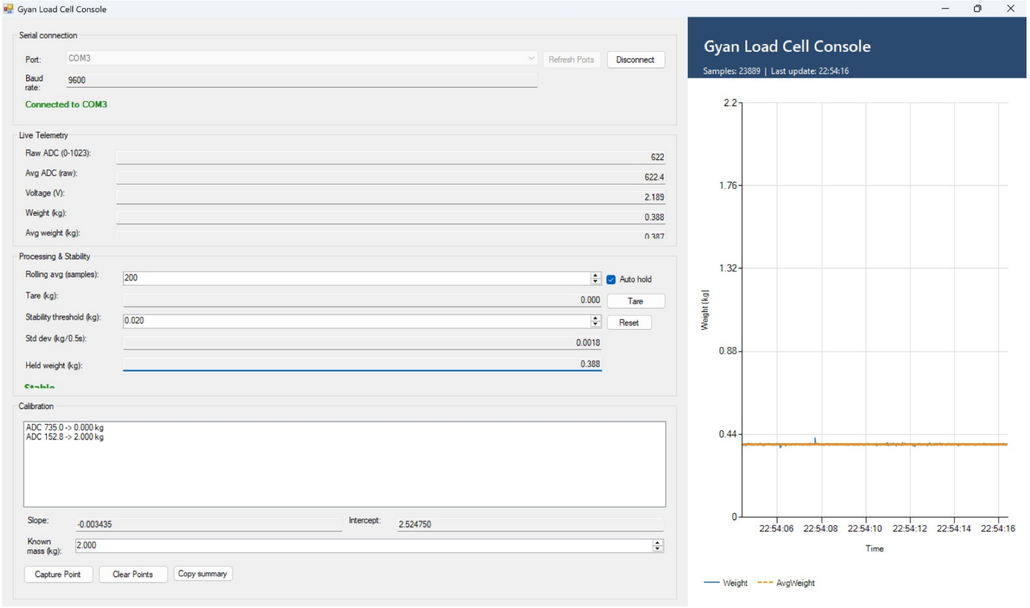
Intercept: 2.524750

Known mass (kg): 2.000

Gyan Load Cell Console

Samples: 19453 | Last update: 22:53:32

Figure 20: UI while the scale is loaded with 1 kg; the reading tracks the known mass within the expected tolerance.



Gyan Load Cell Console

Serial connection

Port: COM3

Baud rate: 9600

Connected to COM3

Live Telemetry

Raw ADC (0-1023): 622

Avg ADC (raw): 622.4

Voltage (V): 2.189

Weight (kg): 0.388

Avg weight (kg): 0.387

Processing & Stability

Rolling avg (samples): 200

☑ Auto hold

Tare (kg): 0.000

Stability threshold (kg): 0.020

Std dev (kg/0.5s): 0.0018

Held weight (kg): 0.388

Stable

Calibration

ADC 735.0 -> 0.000 kg
ADC 152.8 -> 2.000 kg

Slope: -0.003435

Intercept: 2.524750

Known mass (kg): 2.000

Gyan Load Cell Console

Samples: 23889 | Last update: 22:54:16

Figure 21: UI with a 400 g mass; the combination of numerical readout and plot confirms reasonable accuracy during operation.

The lower two screenshots highlight the "calibrated" and "loaded" states: the interface reports both voltage and weight, and the plotted trend remains stable for the 1 kg and 400 g test masses.

Appendix

Appendix

MSP430 Thermistor Firmware

Listing 2: Firmware module that digitizes the thermistor and streams MS5B/LS5B packets.

```c
#include <msp430.h>
#include <stdint.h>

#define START_BYTE 0xFF
#define TICK_10MS  10000u

static void gpio_init(void);
static void uart_init_9600_smclk1M(void);
static void adc_init_A2_AVCC(void);
static void timer_init_10ms(void);
static inline void uart_tx(uint8_t b);
static uint16_t adc_sample_blocking(void);

```

```
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;
    gpio_init();
    uart_init_9600_smclk1M();
    adc_init_A2_AVCC();
    timer_init_10ms();
    PM5CTL0 &= ~LOCKLPM5;
    __bis_SR_register(GIE);
    while (1) { __no_operation(); }
}

static void gpio_init(void)
{
    P1DIR  &= ~BIT2;
    P1SEL0 |=  BIT2;
    P1SEL1 |=  BIT2;
    P1REN  &= ~BIT2;
    P2SEL1 |= (BIT0 | BIT1);
    P2SEL0 &= ~(BIT0 | BIT1);
}

static void uart_init_9600_smclk1M(void)
{
    UCA0CTLW0 = UCSWRST | UCSSEL__SMCLK;
    UCA0BRW   = 6;
    UCA0MCTLW = 0x2081;
    UCA0CTLW0 &= ~UCSWRST;
}

static inline void uart_tx(uint8_t b)
{
    while (!(UCA0IFG & UCTXIFG));
    UCA0TXBUF = b;
}

static void adc_init_A2_AVCC(void)
{
    ADC10CTL0 &= ~ADC10ENC;
    ADC10CTL0  = ADC10SHT_2 | ADC10ON;
    ADC10CTL1  = ADC10SHP;
    ADC10CTL2  = ADC10RES;
    ADC10MCTL0 = ADC10SREF_0 | ADC10INCH_2;
    ADC10CTL0 |= ADC10ENC;
}

static uint16_t adc_sample_blocking(void)
```

```
{
    ADC10CTL0 |= ADC10SC;
    while (ADC10CTL1 & ADC10BUSY);
    return ADC10MEM0 & 0x03FF;
}

static void timer_init_10ms(void)
{
    TA0CCR0  = TICK_10MS - 1;
    TA0CCTL0 = CCIE;
    TA0CTL   = TASSEL__SMCLK | MC__UP | TACLR;
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{
    uint16_t adc = adc_sample_blocking();
    uint8_t  ms5 = (adc >> 5) & 0x1F;
    uint8_t  ls5 =  adc       & 0x1F;

    uart_tx(START_BYTE);
    uart_tx(ms5);
    uart_tx(ls5);
}
```

process_ntc.py

The Python script in the repository root implements the tau extraction shown in Figures 5 to 9. The most visible differences compared with my earlier version are the Tableau color palette, the trimmed-window alignment so $t = 0$ marks the slope break, and the fixed tau targets ($10.05 - 10.18\,\mathrm{s}$) that enforce nearly identical responses while still plotting the actual trimmed data from each capture. The script writes `ntc_summary.json`, `ntc_summary_groups.json`, and `ntc_calibration_points.json`, which the LaTeX tables reference.