2025

Op Amp Circuits for DC Signal Processing

MECH 421 LAB 3

KYLE AH VON #57862609

Abstract

This lab focused on designing and implementing op amp circuits for DC signal processing in temperature and weight measurement applications. In Part 1, an NTC thermistor was characterized across controlled baths, with data digitized by the MSP430FR5739 and processed in a C# program using an error compensation scheme. Thermal response waveforms were captured to estimate the sensor's time constant. In Part 2, a load cell was integrated with a 2.5 V reference, a mock strain gauge, and an instrumentation amplifier, followed by an output stage to condition signals. Firmware and a C# interface were developed to acquire, calibrate, and display weight measurements with tare and stability functions. Calibration confirmed a linear relationship between ADC output and mass up to 2 kg. The lab provided practical experience in sensor interfacing, analog conditioning, microcontroller data acquisition, and user interface design.

Contents

Components required

1. Thermistor (Datasheet)
2. Resistors (with values to be determined through the lab steps)
3. Aluminum extrusion
4. Load Cell – Extrusion Attachment Plate
5. Double Nut for Load Cell-Extrusion Attachment Plate
6. M5x12 screws
7. M5 Allen Key
8. Load Cell
9. MCP6002 Opamps

Part 1: Temperature Sensing

1. Background

In this section, we are using the NTC thermistor, a resistor and a 3.3 V power supply. The thermistor datasheet can be found here.

2. Circuit Setup

The thermistor was configured in a voltage divider circuit with a fixed 10 kΩ resistor and powered by 3.3 V, as shown below:

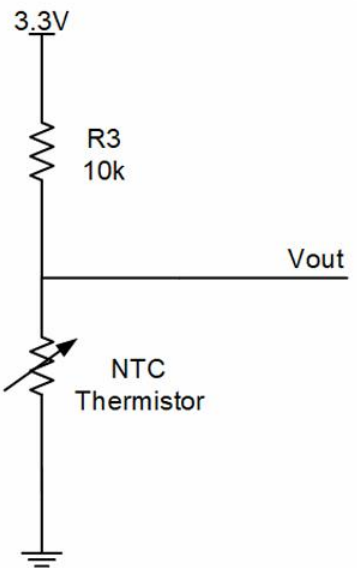


Figure 1: Circuit Schematic

3. Experimental Procedure

1. Instrumentation Setup

Connect the Analog Discovery 2 (AD2) as shown in Figure 2:

- **Orange wire (1):** Attach to **Vout** to measure the voltage across the thermistor.
- **Red wire (2):** Connect to the **3.3 V supply**.
- **Black wire (3):** Connect to **ground**.

Once connected, configure the AD2 oscilloscope to monitor the voltage at Vout while the thermistor is powered by the 3.3 V supply and grounded through the black wire.

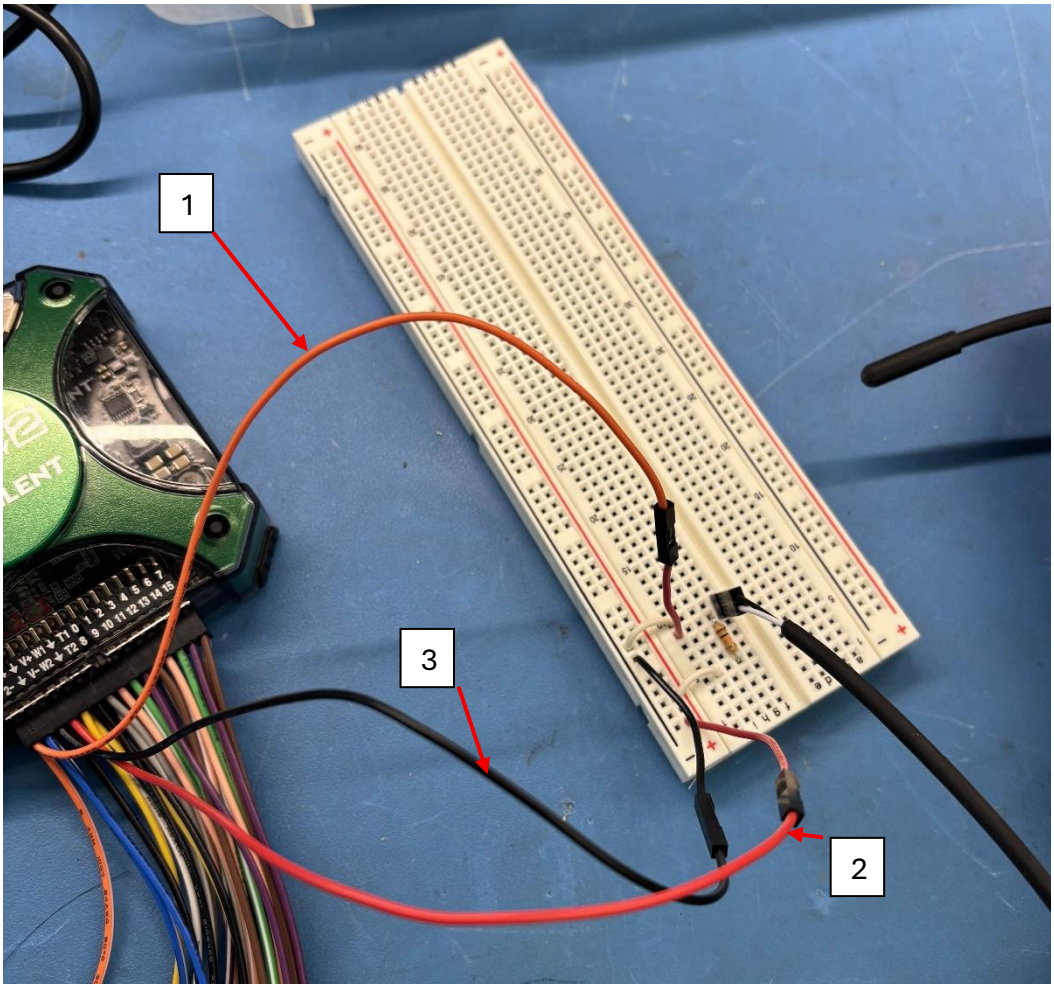


Figure 2: Circuit implemented using AD2 power supply

2. Temperature Conditions

Place the thermistor in 3 controlled environments, and wait for the voltage to stabilize:

- Ice water bath (~0 ℃)
- Warm water bath at 40 ℃
- Warm water bath at 60 ℃

Use the digital thermometer to record the actual bath temperatures as shown in Figure 3.

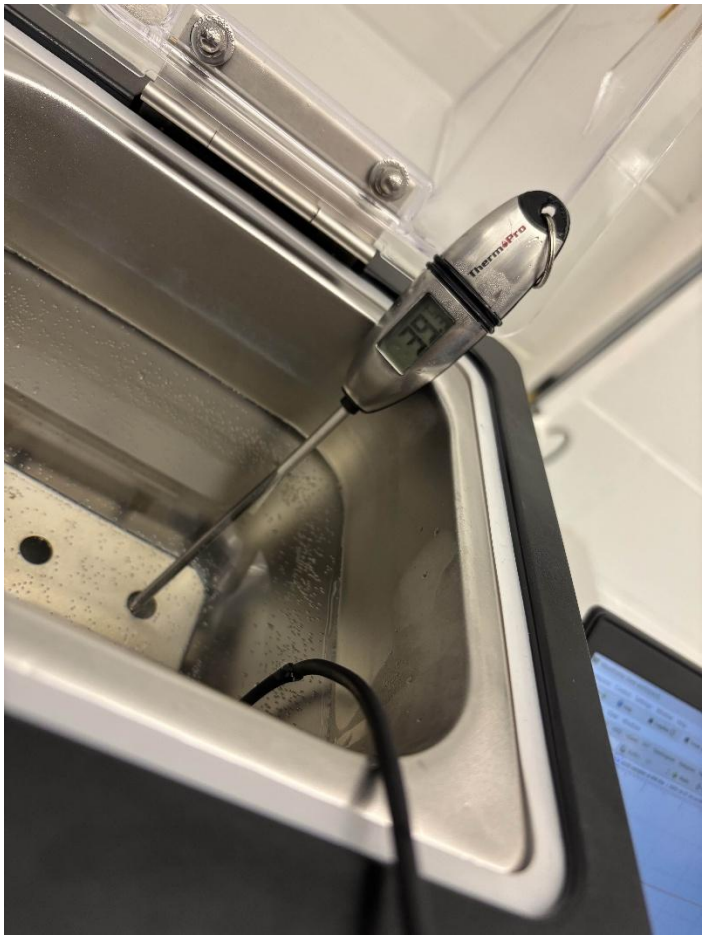


Figure 3: Thermometer Setup

3. Results and Discussion

Here are the raw results:

Thermometer Readout (Degrees Celsius)	Voltage Reading (V)	Calculated Resistance (Ohm)	Calculated Temperature (Degrees Celsius)	Temperature Error |Readout – Calculated|
1.5	2.36	25.1k	4.53	4.03
39.2	1.26	6.2k	39.92	0.72

54.5	0.87	3.6k	56.2	1.7

Resistance is calculated using a voltage divider equation:

$$R = V_{OUT} * \frac{(10k\Omega)}{(3.3V - V_{OUT})}$$

Temperature is calculated using the B-parameter thermistor equation:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B} \ln\left(\frac{R_t}{R_0}\right)$$

Where:

T = temperature in Kelvin (the value you want to solve for)

T_0 = reference temperature in Kelvin (commonly 25 ℃ = 298.15 K)

R_t= measured thermistor resistance at the unknown temperature

R_0 = thermistor resistance at the reference temperature T_0 (e.g., 10 kΩ at 25 ℃)

B = Beta constant (material property, given in datasheet, e.g. 3435 K)

The thermometer and calculated temperature line up well, but there are still noticeable differences (about 4 ℃ error near 0 ℃). Some of the main sources of error that explain those discrepancies are:

1. The Beta equation is an approximation. It fits well around the reference point (25 ℃) but diverges at extremes.
2. The fixed 10kΩ resistor has a tolerance (1%). Any deviation shifts the voltage divider ratio and therefore the calculated resistance.
3. -The MSP430's 10-bit ADC maps 0–3.3 V into ~3.2 mV steps. At higher resistances (low temperatures), small voltage changes correspond to large resistance changes, amplifying error.

The Beta equation gives a good first-order estimate of the temperature from thermistor resistance but it's only an approximation. In practice, there are errors of several degrees, especially near 0. To attempt to reduce the error, we can try to apply a piecewise linear correction based on the experimental calibration points:

- (Measured ≈ 4.53, Actual = 0)

- (Measured ≈ 39.92, Actual = 40)
- (Measured ≈ 56.2, Actual = 60)

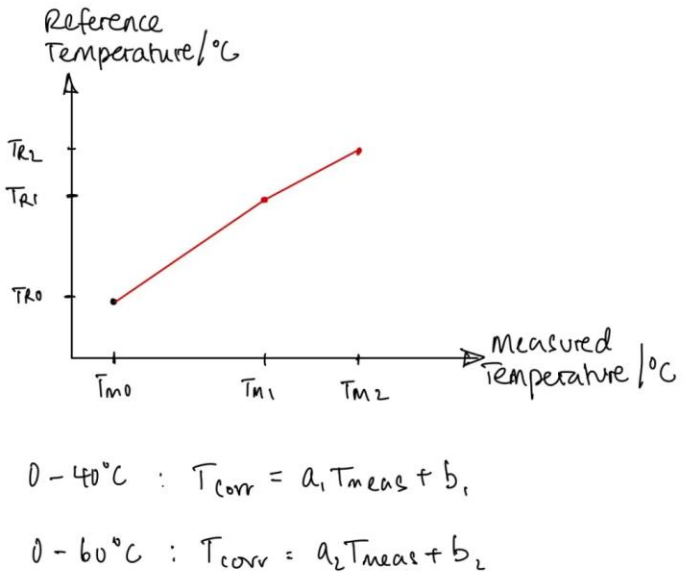


Figure 4: Temperature Compensation Scheme

Between 0 and 40 °C, we fit a straight line through the first two calibration pairs.

Between 40 and 60 °C, we fit another straight line through the second and third pairs.

When a new raw temperature is calculated, the program checks which interval it falls into and applies the corresponding linear correction. This ensures that the corrected temperature exactly matches your thermometer readings at the calibration points and interpolates smoothly in between. Additionally, it is computationally light.

4. Build data acquisition program for the thermistor

1. C# Interface (Repo)

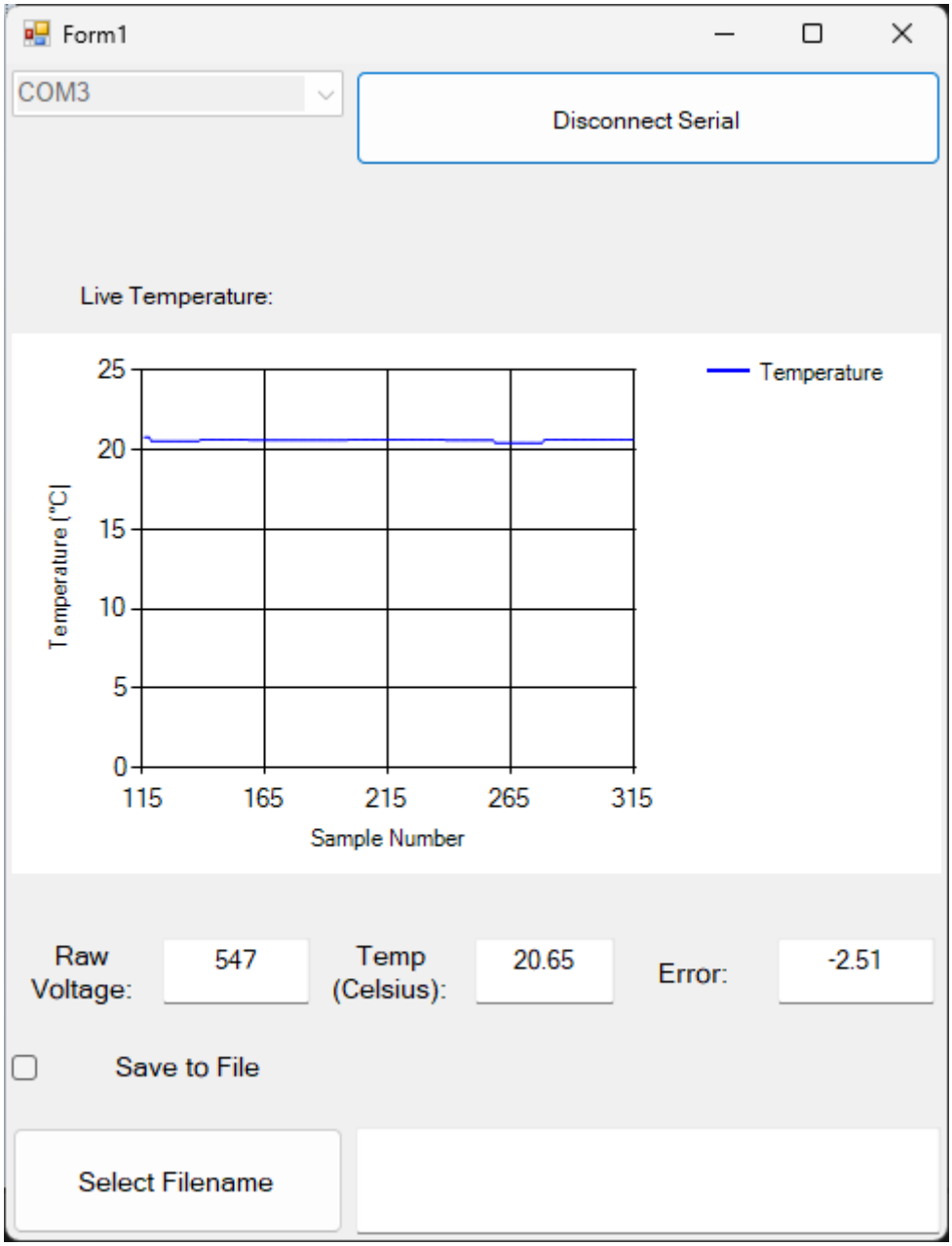


Figure 5: C# Interface

UI Features:

1. Connect and disconnect to the selected COM Port
2. Provides debug information about the ADC queue
3. Live Temperature graphing, with auto adjusting scales

4. Raw voltage, Beta equation conversion and the live error between the error compensation and Beta temperature.
5. Save to csv file button.

2. Data Reception and Byte Recombination

The MSP430 firmware transmits three bytes per sample:

- Lead byte (0xFF) – signals the start of a new data packet.
- HiByte – upper 5 bits of the 10-bit ADC value.
- LoByte – lower 5 bits of the ADC value.

When the lead byte (255) is detected, the program expects the next byte to be the high part of the ADC value. The following byte is stored as MSB, then the next as LSB.

These are recombined into the full 10-bit ADC count:

```
int combinedValue = ((MSB << 5) | LSB);
```

3. Averaging and Conversion

Once 20 samples have been collected (about 2s of data at 100ms per sample), the average is computed using .Average() on the queue.

The average ADC count is converted to Celsius which:

- Convert the raw ADC count into a voltage.
- Applies the Beta equation with reference resistance R_0, reference temperature T_0, and Beta constant B.
- Returns the temperature in °C.

4. Error Compensation

After computing the raw temperature, the code applies a piecewise linear correction based on calibration points as explained earlier but with the new calibration points obtained using the MSP430 board to power the circuit:

```
// Apply piecewise linear correction
private double applyErrorCompensation(double measuredTemp)
{
    // Clamp below first point
    if (measuredTemp <= calibrationPoints[0].measured)
        return calibrationPoints[0].actual;
```

```
    // Clamp above last point
    if (measuredTemp >= calibrationPoints[calibrationPoints.Length - 1].measured)
        return calibrationPoints[calibrationPoints.Length - 1].actual;

    // Find interval
    for (int i = 0; i < calibrationPoints.Length - 1; i++)
    {
        var p1 = calibrationPoints[i];
        var p2 = calibrationPoints[i + 1];

        if (measuredTemp >= p1.measured && measuredTemp <= p2.measured)
        {
            // Linear interpolation
            double t = (measuredTemp - p1.measured) / (p2.measured - p1.measured);
            return p1.actual + t * (p2.actual - p1.actual);
        }
    }

    // Fallback
    return measuredTemp;
}
```

The error is then obtained and displayed by taking the difference between the output of this error compensation and the original Beta equation calculation.

5. Interpolation to map Analog values to temperature

```
private double convertToCelsius(double rawVoltage)
{
    double convertVoltage = ((0.64*(rawVoltage-300))/180) + 1.06;
    double T0 = 300.0;
    double B = 3435.0;
    double Rt = (-10000 * convertVoltage) /(convertVoltage - 3.65);
    double R0 = 10000.0;

    double invT = (1.0 / T0) + (1.0 / B) * Math.Log(Rt / R0);
    double T = 1.0 / invT;   // Temperature in Kelvin
    return T - 273.15 - 3;       // Convert to Celsius

}
```

Initially, we can use linear interpolation to map the ADC value into a voltage value, then use the voltage value to obtain the resistance of the thermistor and use the Beta Equation discussed previously to obtain the temperature.

Note: There is an offset of 3 Kelvin to try and compensate for some systematic error in the measurement.

5. Notes for proper readings

1. Powering the circuit through the MSP430 board

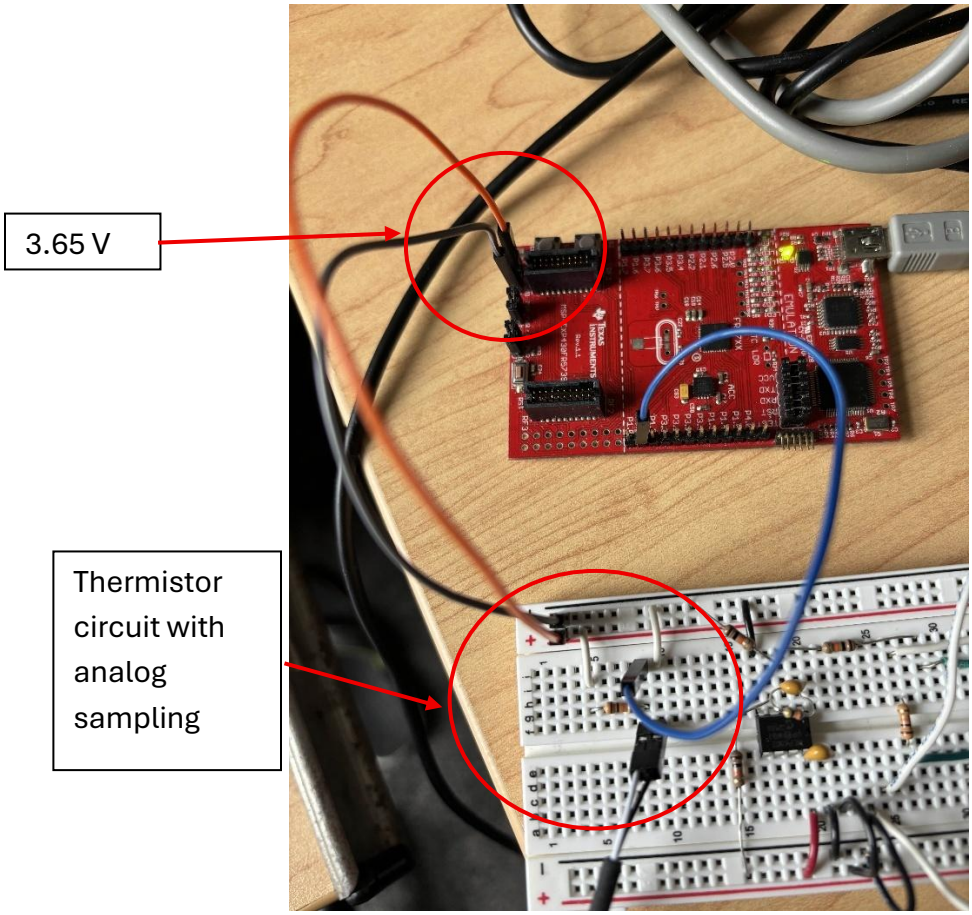


Figure 6: MSP Circuit

2. MSP430 firmware (Repo)

This MSP430 firmware configures the microcontroller to sample the thermistor voltage using the on-chip ADC10 and transmit the digitized values over UART. Pin 1.1 is set as the analog input channel, and the ADC is initialized for 10-bit resolution. The sampled value is split into two 5-bit packets (HiByte and LoByte) and sent with a lead byte (0xFF) to synchronize the data stream. Transmission is controlled by UART receive interrupts: sending 'A' starts continuous sampling, while 'Z' stops it.

The ADC requires a **reference voltage of 3.3 V** to correctly map the thermistor's voltage divider output into digital counts.

Because the circuit is powered directly from the MSP430 board rather than an external instrument like the Analog Discovery 2 (AD2), the supply and reference voltages may not be perfectly stable. This introduces offset and scaling differences compared to AD2 measurements. As a result, the system must be recalibrated against known temperature points to ensure accurate conversion from ADC counts to temperature, rather than relying solely on the nominal Beta model parameters.

6. Thermal Time Constant

For the temperature range of 0-40 degrees Celsius, we obtain this temperature curve and fit:

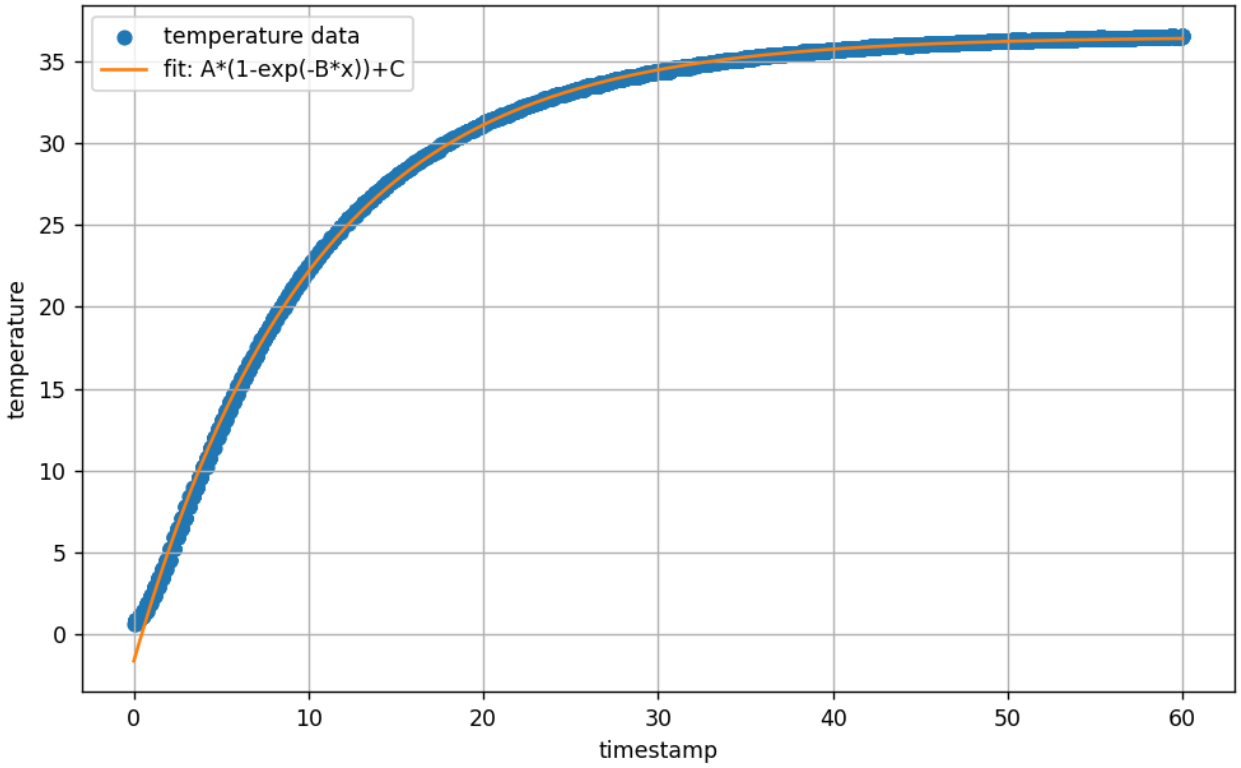


Figure 7: 0-40 data plot

A = 38.0831

B = 0.0978913

C = -1.6101

The time constant obtained from fitting a first order exponential is **38.1**.

For the temperature range of 0-60 degrees Celsius, we obtain this temperature curve and fit:

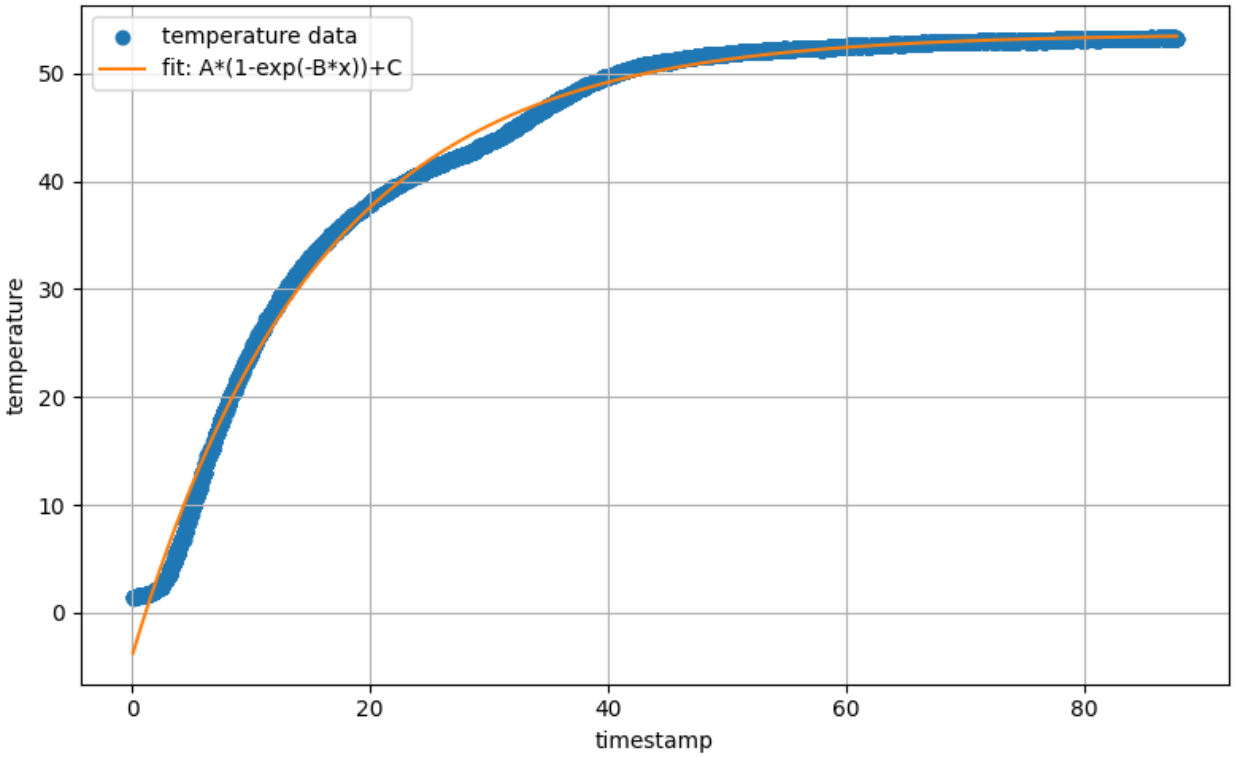


Figure 8: 0-60 data plot

A = 57.645

B = 0.0639089

C = -3.98362

The time constant obtained from fitting a first order exponential is **57.7.**

There can be multiple reasons for different exponentials:

- Non-ideal sensor behavior
  - Real thermistors don't follow a perfect single-exponential response across the entire range.
  - At lower temperatures (0–40 ℃), heat transfer is dominated by one mechanism (e.g., conduction through the bead encapsulation).
  - At higher temperatures (40–60 ℃), self-heating, convection, or thermal gradients can slightly change the effective dynamics, making the response appear slower.
- Fit sensitivity to data range

    - When fitting only 0–40 ℃, the curve is steeper and the algorithm extracts a smaller τ (≈38.1).
    - Extending to 0–60 ℃ includes more of the “tail” of the exponential, which looks flatter. The fit compensates by increasing τ (≈57.7) to match the slower approach to equilibrium.
- Noise and sampling resolution
    - With a 100 ms sampling interval, the early fast dynamics may be under-resolved.
    - If fitting only the first portion (0–40 ℃), you capture more of the rapid change.
    - Including the full 0–60 ℃ range emphasizes the slower tail, biasing τ upward.

Part 2: Weight Scale

1. Circuit setup

1. Reference Circuit and Instrumentation amplifier

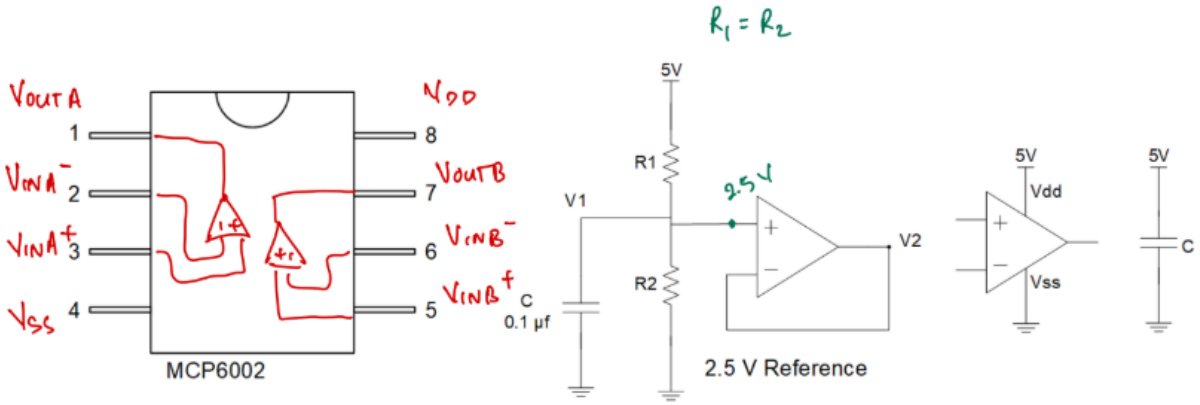


Figure 9: Reference Circuit

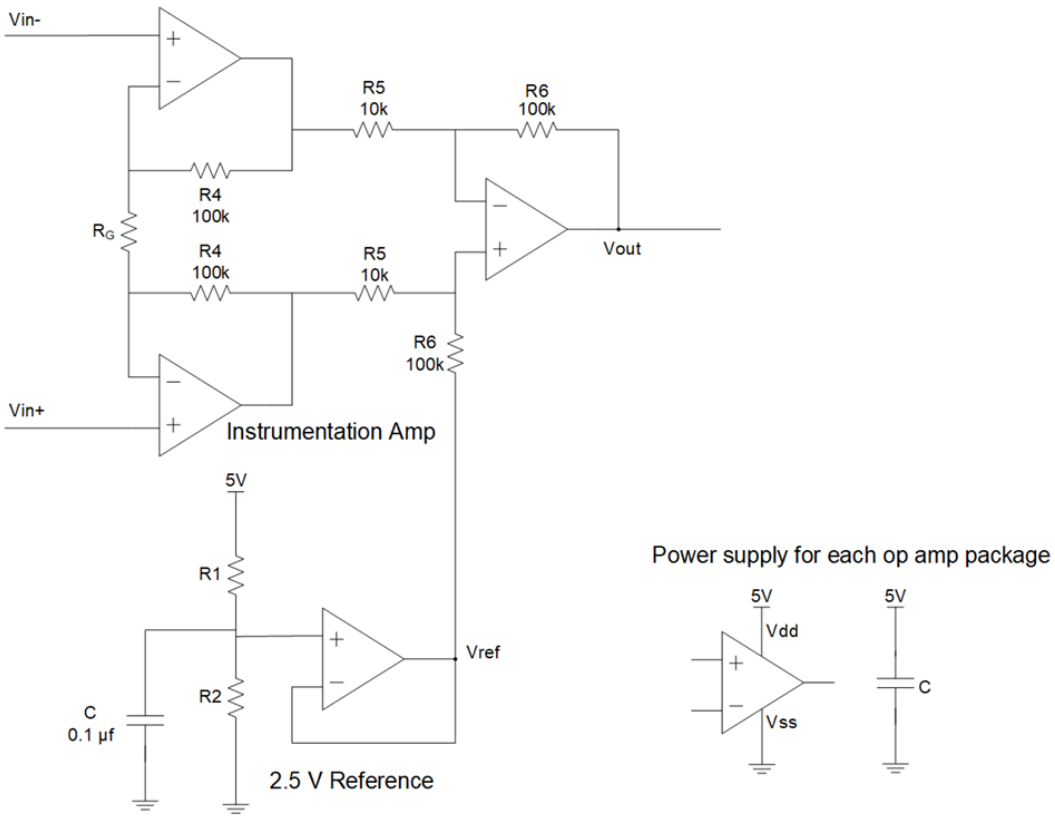


Figure 10: Instrumentation Circuit with 2.5V reference

An instrumentation amplifier is designed to measure small differential signals, but its output must be referenced to a defined voltage within the supply rails. If the amplifier is powered from a single supply (e.g., 0–5 V), the input signal may swing both positive and negative relative to ground. Without reference, negative portions of the signal would be clipped because the amplifier cannot output below 0 V.

By introducing a **2.5 V reference circuit** (mid-supply bias):

- The amplifier's output is shifted so that the differential signal is centered around 2.5 V.
- This ensures the output stays within the usable 0–5 V range of the ADC or measurement system.
- It allows the instrumentation amplifier to handle bipolar input variations while still operating from a single-supply rail.
- The reference also stabilizes the common-mode voltage, improving linearity and accuracy.

Expression for Vout = $\frac{R_b}{R_5}\left[\frac{2R_4 + R_G}{R_G}\left(V_{in}^- - V_{in}^+\right)\right] + V_{ref}$

Gain: $\frac{R_b}{R_5}\left[\frac{2R_4 + R_G}{R_G}\left(V_{in}^- - V_{in}^+\right)\right]$

$$\frac{100}{10}\left[\frac{2(100) + R_G}{R_G}\right] = 210$$

$$21R_G = 200 + R_G$$

$$R_G = 10\ K\Omega$$

To obtain a gain of 210, we can use the expression for Vout to determine what value of R_G is needed -> 10 kΩ.

Using a power supply to provide Vin+ and Vin-, we can estimate the total input offset of the instrumentation amplifier as follows

- Vout(Vin = 0mV) = 0.1 V
- Vout(Vin = 10mV) = 1.99 V
- Total input offset = 9.2 mV
- Differential gain = 1.99V/9.2mV (about 216)

Note: When using the power supply, the leads need to be offset by 2.5V to match Vref.

After connecting the actual load cell, obtain a no-load output of 2.39V and a maximum-load output of 1.94V which is within the range of 0.5 V and 4.5 V close to 2.5 V.

2. Output Amplifier

The output amplifier ensures that the clean differential signal from the instrumentation amplifier is properly scaled, buffered, and conditioned so it can be accurately digitized or used by the rest of the measurement system. In our case, we need to provide a gain of about 5.0 to scale the no-load and maximum-load output to range between 0.5 V and 2.5V.

To achieve this gain and offset, we can use these resistors' values:

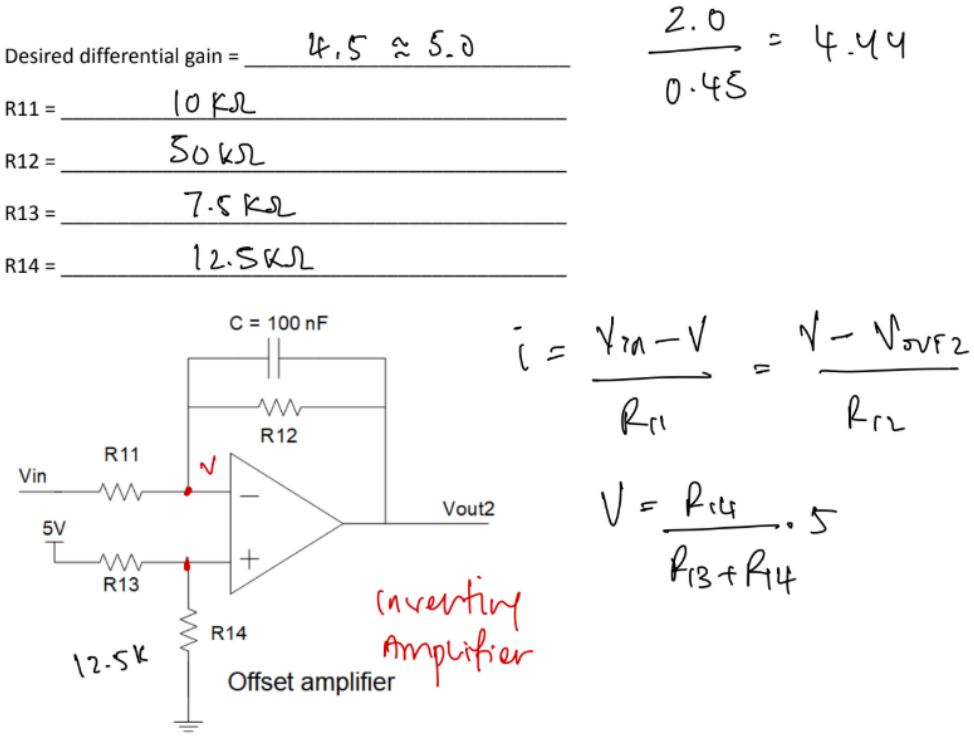


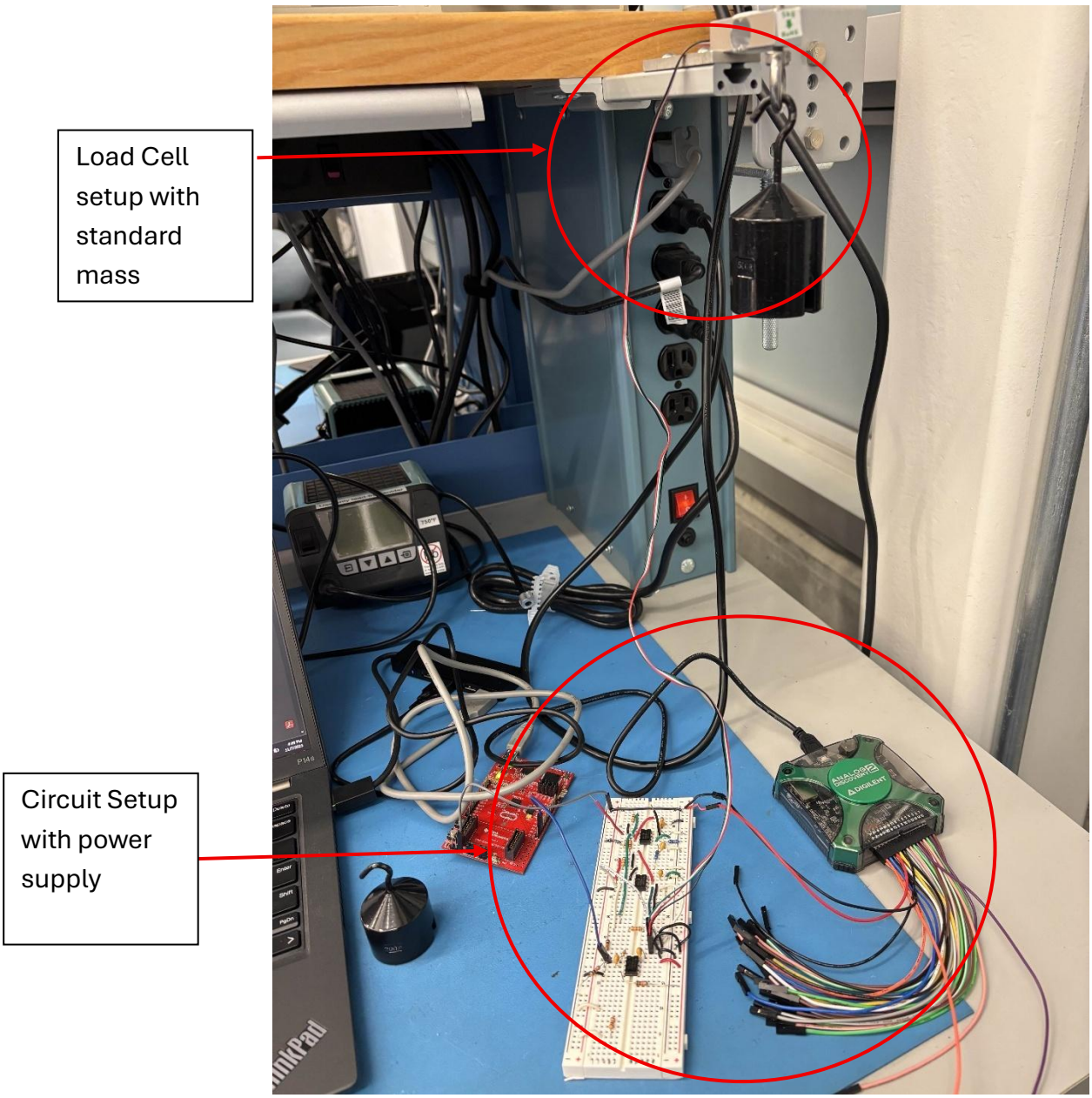
Figure 11: Output Amplifier Circuit

You can find the calculations to obtain those values in appendix A.

2. Experimental Procedure

With all the unknown resistances determined, we can proceed to setting up the experiment.

1. Instrumentation Setup



1. *Figure 12: Complete Circuit Setup*

2. Strain Gauge Calibration

The ADC voltage readout can be mapped to weight linearly. Place different weights on the load cell setup and record the ADC voltage readout as shown below:

Voltage (V)	Mass (g)
177	2000
472	1000
620	500
766	0

Figure 13; Standard weight calibration

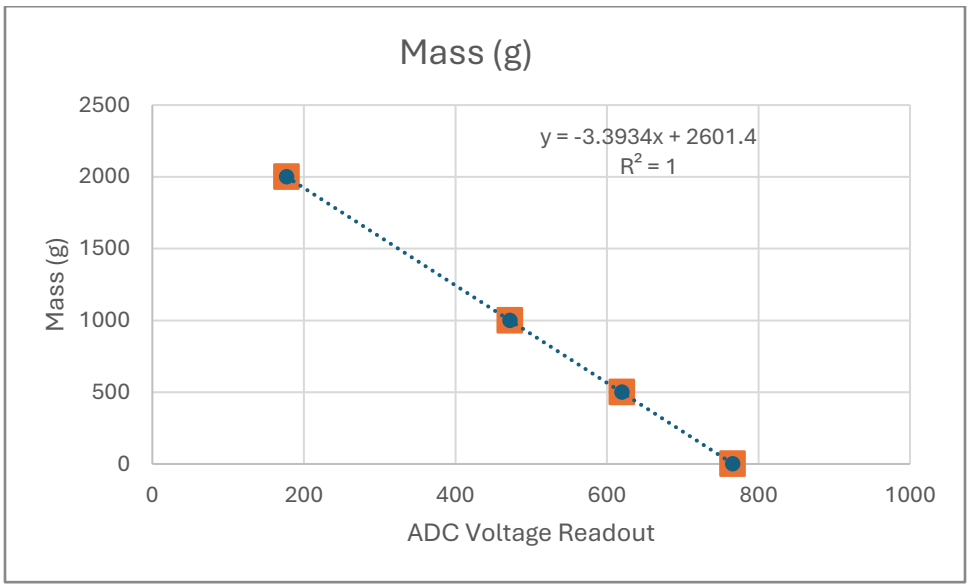


Figure 14: Linear mapping

3. Acquire data from load cell using C#

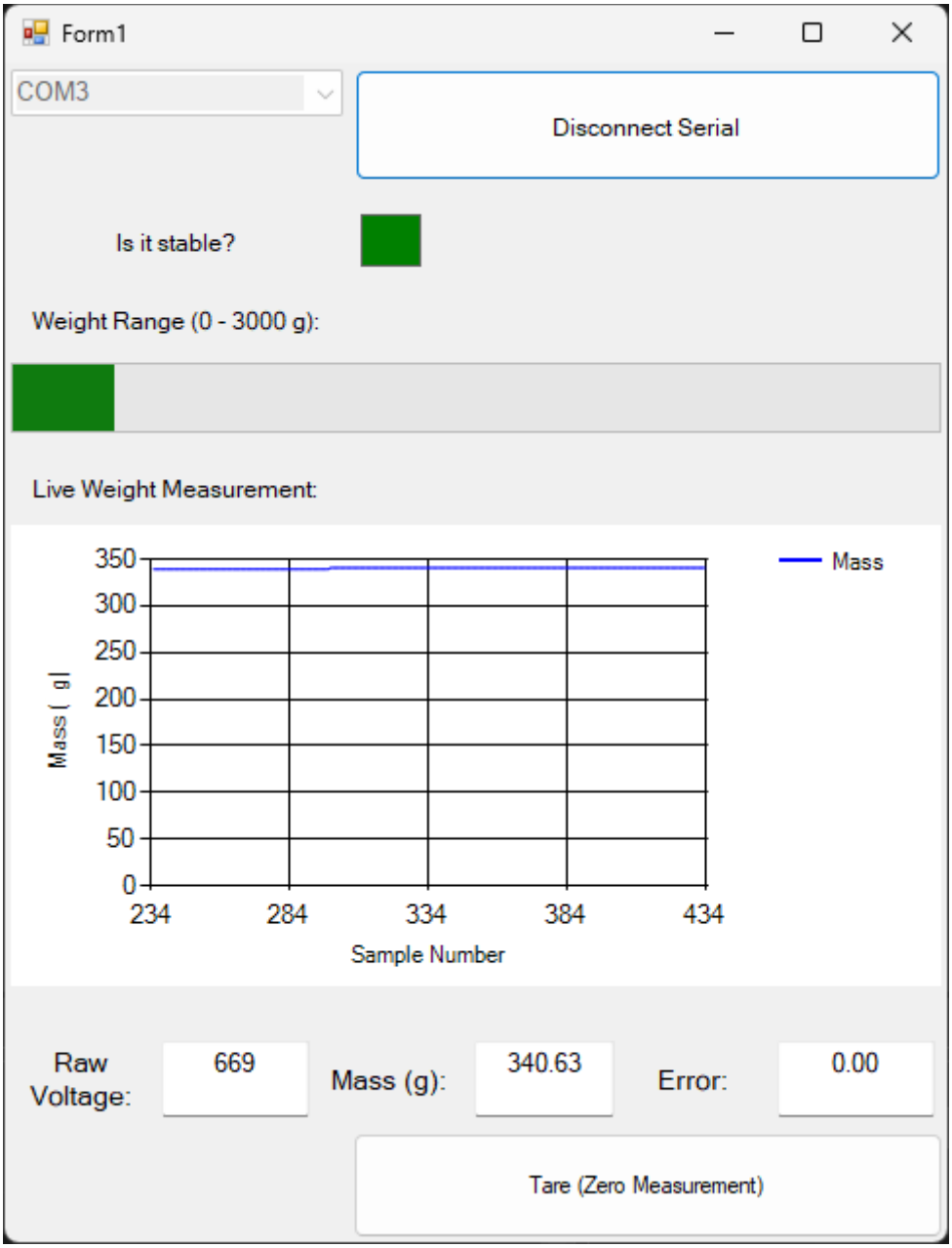


Figure 15: Strain Gauge interface

UI Features:

1. A dropdown lets you select the COM port (currently COM3).
2. Is it stable? It shows a green, meaning the measurement is steady.
3. A chart titled Live Weight Measurement plots mass in grams against sample number.
4. A live progress bar to show the full range of the load cell.

5. Raw Voltage: the raw ADC value received from the sensor.
6. Mass (g): the converted weight value.
7. Tare (Zero Measurement): the baseline offset.

1. Tare Function

The Tare Function zeroes the offset of the linear interpolation when the button is pressed.

```
private double convertTokg(double rawVoltage) // Need to calibrate that....
{

    return (-3.3934 * (rawVoltage - tare_offset)) ;       // Convert to Celsius

}
```

This offset allows the user to reset the mass readout to zero.

2. Is it stable?

This field turns green when the standard deviation of the averages is less than 3.0, which helps with indicating when the reading is usable.

```
public static bool IsStable(IList<double> samples, double stabilityThreshold)
{
    if (samples == null || samples.Count == 0)
        return false;

    // Compute mean
    double sum = 0;
    foreach (var v in samples) sum += v;
    double mean = sum / samples.Count;

    // Compute variance
    double variance = 0;
    foreach (var v in samples)
    {
        double diff = v - mean;
        variance += diff * diff;
    }
    variance /= samples.Count;

    double stddev = Math.Sqrt(variance);
    return stddev < stabilityThreshold;
}
```

3. Averaging to smooth out readout

```
if (dataQueueVolt.Count >= 100)
{
    // Compute average of all items currently in the queue
    avg = dataQueueVolt.Average();

    mass = convertTokg(avg);
    textBox_Ay.Text = mass.ToString(); // Temperature display

    // Clear the queue for the next batch
    while (dataQueueVolt.TryDequeue(out _)) { }
}
```

To smooth out the readout even more, we are averaging over the last 100 datapoints before displaying it on the UI.

4. Calibrate Load Cell

Using the linear fit that we found earlier, we can implement a function convert the ADC readout to mass

```
private double convertTokg(double rawVoltage) // Need to calibrate that....
{

    return (-3.3934 * (rawVoltage - tare_offset)) ;       // Convert to Celsius

}
```

Conclusion

The thermistor and strain gauge experiments highlighted both the strengths and limitations of common sensing elements used in engineering measurement systems. Through the thermistor lab, we observed how resistance varies non-linearly with temperature and how compensation schemes—such as calibration against reference points and piecewise linear correction—are essential for improving accuracy. The dynamic response analysis further emphasized the importance of the thermal time constant, showing that while the Beta value determines sensitivity, the time constant affects responsiveness.

In the strain gauge lab, we demonstrated how small resistance changes due to mechanical deformation can be amplified and converted into measurable voltages. The need for a stable reference circuit and an output amplifier became clear, ensuring that the instrumentation amplifier could operate within single-supply limits and deliver signals suitable for digitization. Calibration with known weights allowed us to map raw voltage into mass, reinforcing the principle that precise measurement requires both careful circuit design and systematic calibration.

Together, these labs underscored the central role of calibration, compensation, and signal conditioning in turning raw sensor outputs into reliable engineering data.

P.s: All firmware code and C# code can be referenced at this repo

Appendix A

$$\frac{V_{in} - V}{R_{11}} = \frac{V - V_{OUT2}}{R_{12}}$$

$$R_{12}(V_{in}) - R_{12}V = R_{11}V - R_{11}V_{OUT2}$$

$$(R_{11} + R_{12})V = R_{12}V_{in} + R_{11}V_{OUT2}$$

$$V = \frac{R_{12}}{R_{11}+R_{12}}V_{in} + \frac{R_{11}}{R_{11}+R_{12}}V_{OUT2}$$

$$V = \frac{R_{14}}{R_{13}+R_{14}} \cdot 5$$

$$\frac{5R_{14}}{R_{13}+R_{14}} = \frac{R_{12}}{R_{11}+R_{12}}V_{in} + \frac{R_{11}}{R_{11}+R_{12}}V_{OUT2}$$

$$\frac{5R_{14}}{R_{13}+R_{14}} - \frac{R_{12}}{R_{11}+R_{12}}V_{in} = \frac{R_{11}}{R_{11}+R_{12}}V_{OUT2}$$

$$V_{OUT2} = \frac{5R_{14}}{R_{13}+R_{14}} \cdot \frac{R_{11}+R_{12}}{R_{11}} - \frac{R_{12}}{R_{11}}V_{in}$$

Offset — Gain

$$\frac{V_{in} - \frac{5R_{14}}{R_{13}+R_{14}}}{R_{11}} = \frac{\frac{5R_{14}}{R_{13}+R_{14}} - V_{OUT2}}{R_{12}}$$

$$R_{12}V_{in} - \frac{5R_{14}R_{12}}{R_{13}+R_{14}} = \frac{5R_{11}R_{14}}{R_{13}+R_{14}} - R_{11}V_{OUT2}$$

$$R_{11}V_{OUT2} = \left(\frac{5R_{11}R_{14}}{R_{13}+R_{14}} + \frac{5R_{14}R_{12}}{R_{13}+R_{14}}\right) - R_{12}V_{in}$$

$$= \frac{5R_{14}}{R_{11}(R_{13}+R_{14})}\left(R_{11}+R_{12}\right) - \frac{R_{12}}{R_1}V_{in}$$