Department of Mechanical
and Mechatronics Engineering

ME 546 - Lab 3
Complete Sensor Fusion System

William Ancich   Austin Milne
Jude Bennett

March 03, 2024

Abstract

This report was prepared for Prof. Arash Araami as part of the ME 546 - Multi Sensor Data Fusion Course. The goal of this lab is to fuse the readings of multiple different types of sensors using Bayesian fusing.

Contents

List of Tables

List of Figures

List of Code Listings

1 Experiment Setup

The experiment was prepared as outlined in the Lab Manual [1]. Due to there being issues with some of the sensors on the lab, an alternate set of sensors were used than those prescribed:

- 2 Long Distance IR Sensors (Sharp GP2Y0A02YK0F)
- 2 Short Distance IR Sensors (Sharp GP2Y0A41SK0F)

The Long and Short sensors were kept together. This was done to avoid the potential issue the sensors physically interfering with the others' views. Due to the vastly different reading range, the short sensor would need to be much closer than the long sensor and could physically block its view, returning uncharacteristic measurements. The final configuration had the two short sensors reading in the Y direction and the two long sensors reading in the X direction.

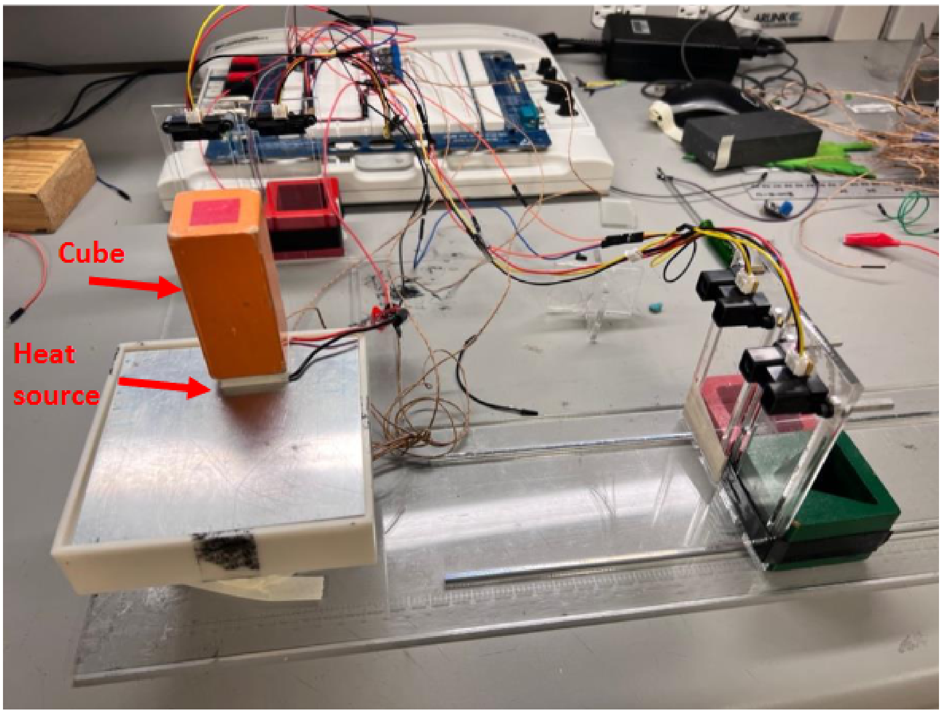


Figure 1: Example Setup [1]

Before collecting the training data, a series of measurements were taken with the IR sensors to later calibrate them. Each sensor was measured at 6 or 7 distances, 2 closer than the plates edges, 1 at the closest edge, 1 at the center, 1 at the furthest edge, and 1 or 2 past the furthest edge. This allowed for a general profiling of the IR sensors along with specific data for the range to be measured in.

For the training data, 9 points were selected along the grid. Four points at the corners, 1 at the center, and the remaining 4 in the intermediaries between the previous points. The location of the block was measured for each positional reading. They are displayed in figure 3. The points are numbered as shown in figure 2, such that 1, 3, 7, and 9 are the corner measurements.

1	2	3
4	5	6
7	8	9

Figure 2: Training Measurement Layout

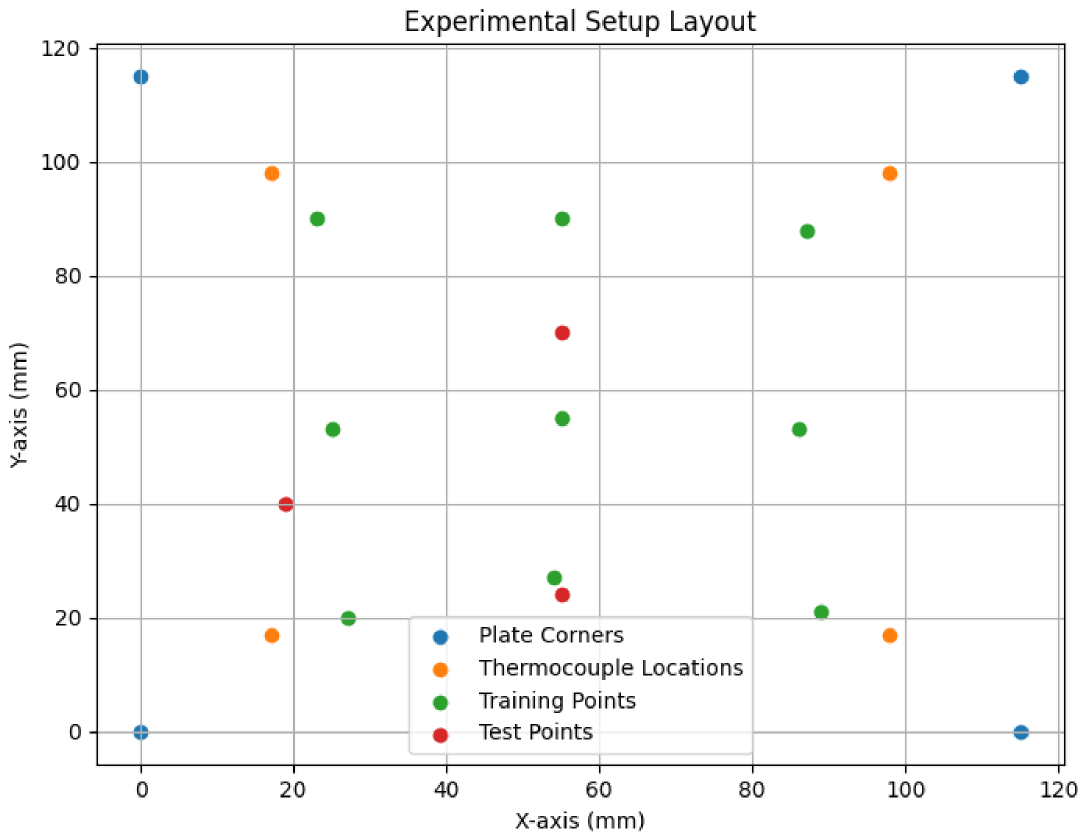


Figure 3: Experimental Setup Layout

After training data was collected, 3 additional points were collected for later testing of the developed model. 1 measurement was taken in a previously trained position, the other 2 were taken in intermediary positions that do not exist in the training set. This was done to compare the performance of the model on known and unknown states. The exact locations of the test points are listed in table 1.

Table 1: Test Point Locations

Point	X (mm)	Y (mm)
1	55	24
2	19	40
3	55	70

Between all measurements requiring the thermocouples, the aluminum plate was removed from the fixture. The plate was shaken in the air for 30-45 seconds to accelerate the heat dissipation and return it to room temperature. Each of the thermocouples was raised above the necessary height, and then the plate was placed on top of them and tapped down to ensure full contact with all 4 sensors. The Peltier cell was then very gently placed on the aluminum plate as to not move the thermocouples and the wooden block was place atop the cell.

2 Sensor Calibration

2.1 IR Sensor Profiling

As mentioned in Section 1, measurements were taken on both the short and long range sensors at a variety of distances. Figure 4 and 5 show the raw voltage measurements for the short and long range sensors, respectively. The voltage graphs show a consistent and uniform voltage reading for each of the sensors, showing that the sensors were wired correctly and behaved as expected.

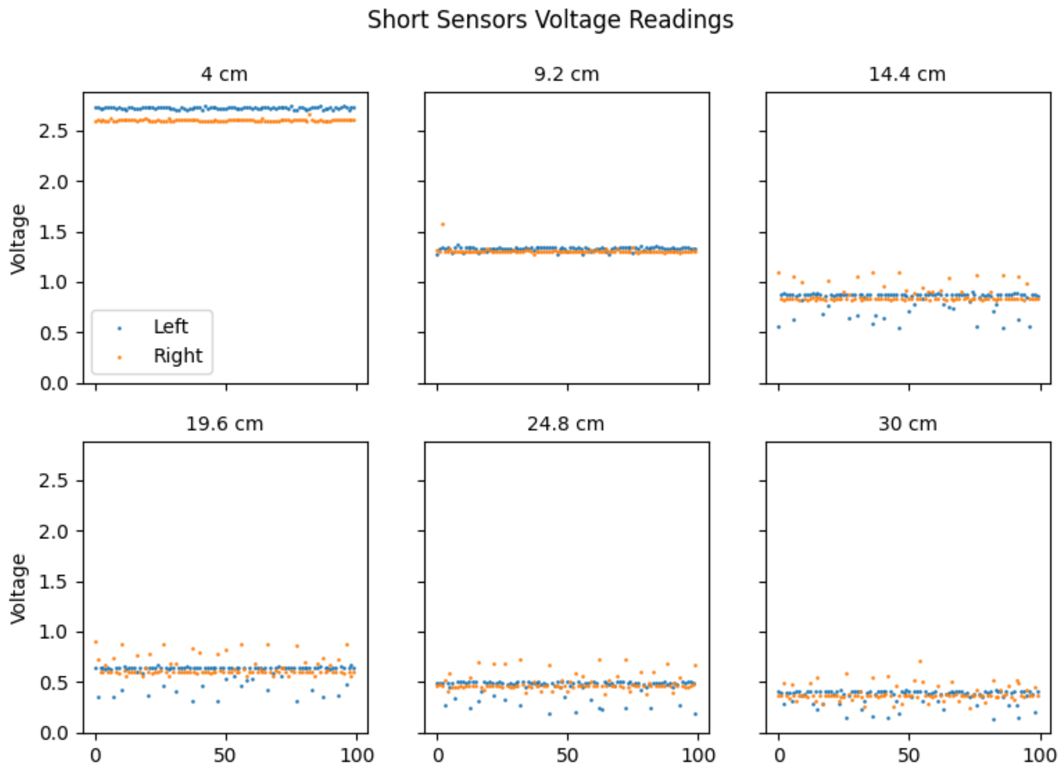


Figure 4: Short IR Sensors Raw Voltage Readings

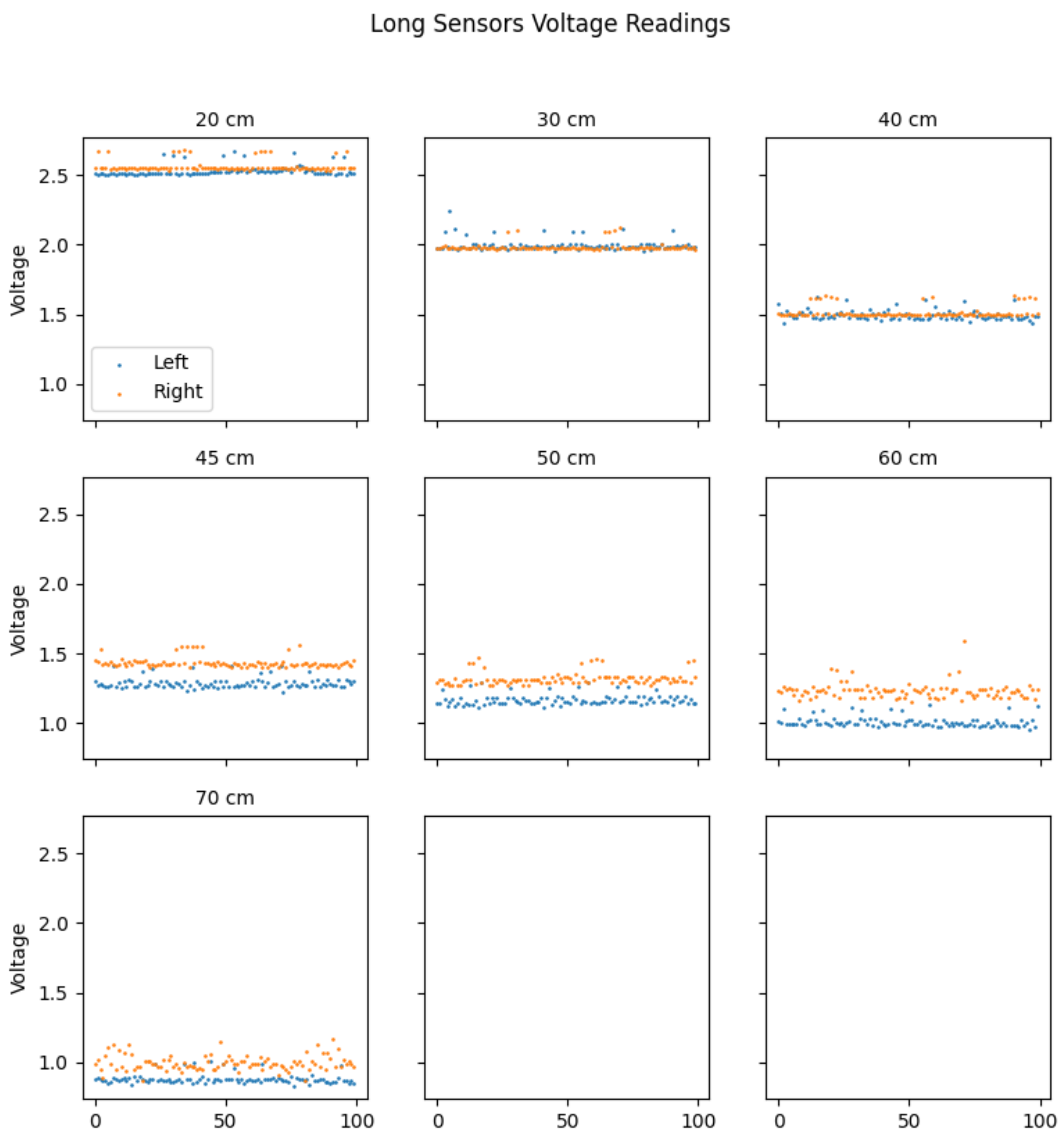


Figure 5: Long IR Sensors Raw Voltage Readings

Data for each of the sensors was used to create a Voltage and Distance relation. Regression was done using as inverse relationship as previously proven in Lab 1 [2], as show in equation 1. The regressions and resultant parameters are shown for each IR sensor in figure 6.

$$D = a \times V + b \tag{1}$$

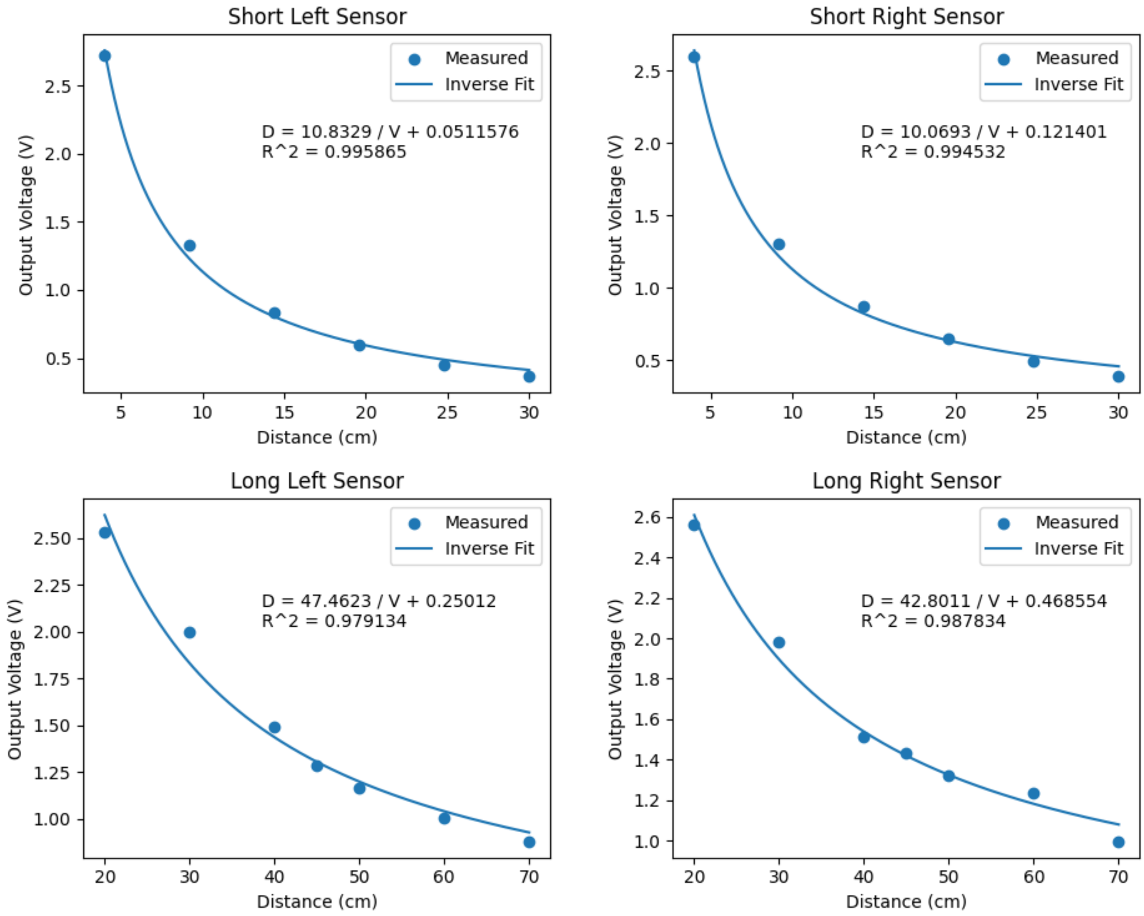


Figure 6: IR Sensors Regression to Inverse Relationship

2.2 Thermocouple Profiling

As mentioned in Section 1, measurements were taken with all thermocouples with the heat source placed at a variety of positions. Figure 7 shows the raw voltage measurements for each thermocouple, at each position in the training set. The voltage graphs show a consistent and uniform voltage reading for each of the thermocouples, as was expected behavior for the sensor at steady state operation.

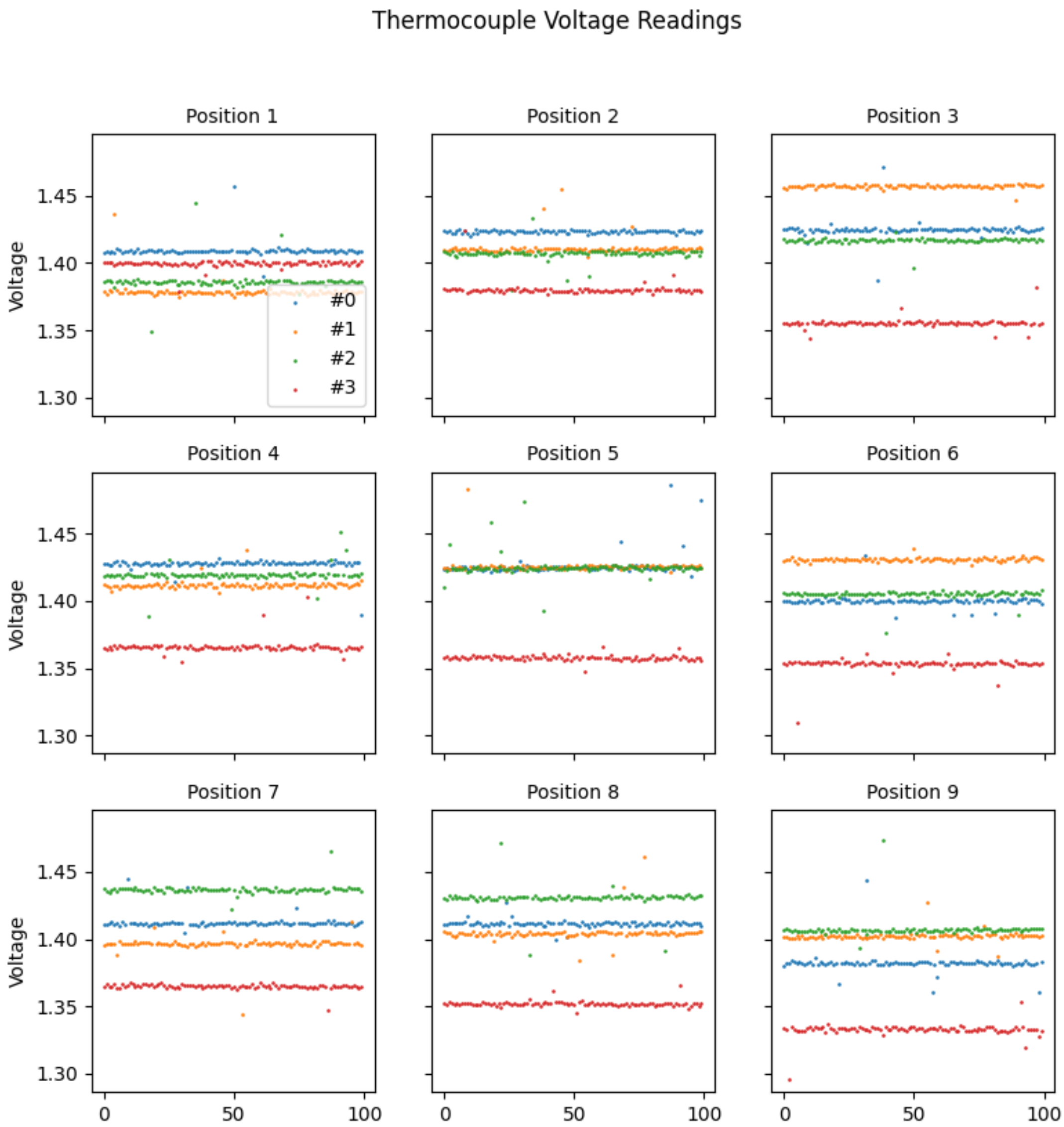


Figure 7: Thermocouple Raw Voltage Readings

As the thermocouple was a sensor that had not been used prior to this lab, modelling the thermocouple was first required. The desired model needed to establish the relationship between thermocouple temperature readings and the distance between a thermocouple and the heat source. As with the IR sensors, the voltages readings obtained earlier were averaged mitigate the effect of sensor noise. Then, the mean output voltages from the thermocouples were substituted into equation 2 from the lab manual [1] to obtain corresponding temperature readings.

$$T = \frac{V_{\text{out}} - 1.25}{0.005} \tag{2}$$

To establish a relationship between distance and temperature, least squares were used to fit functions (linear, inverse, quadratic, cubic) to the test data. These functions were then plotted (as shown in figure 8) and the coefficient of determination was evaluated for each of these functions (using the true and predicted distance values) to assess which model was the most accurate. At this point it was noted that the relationship between temperature and distance was reversed for thermocouple 4, indicating that it was wired in reverse during data collection.

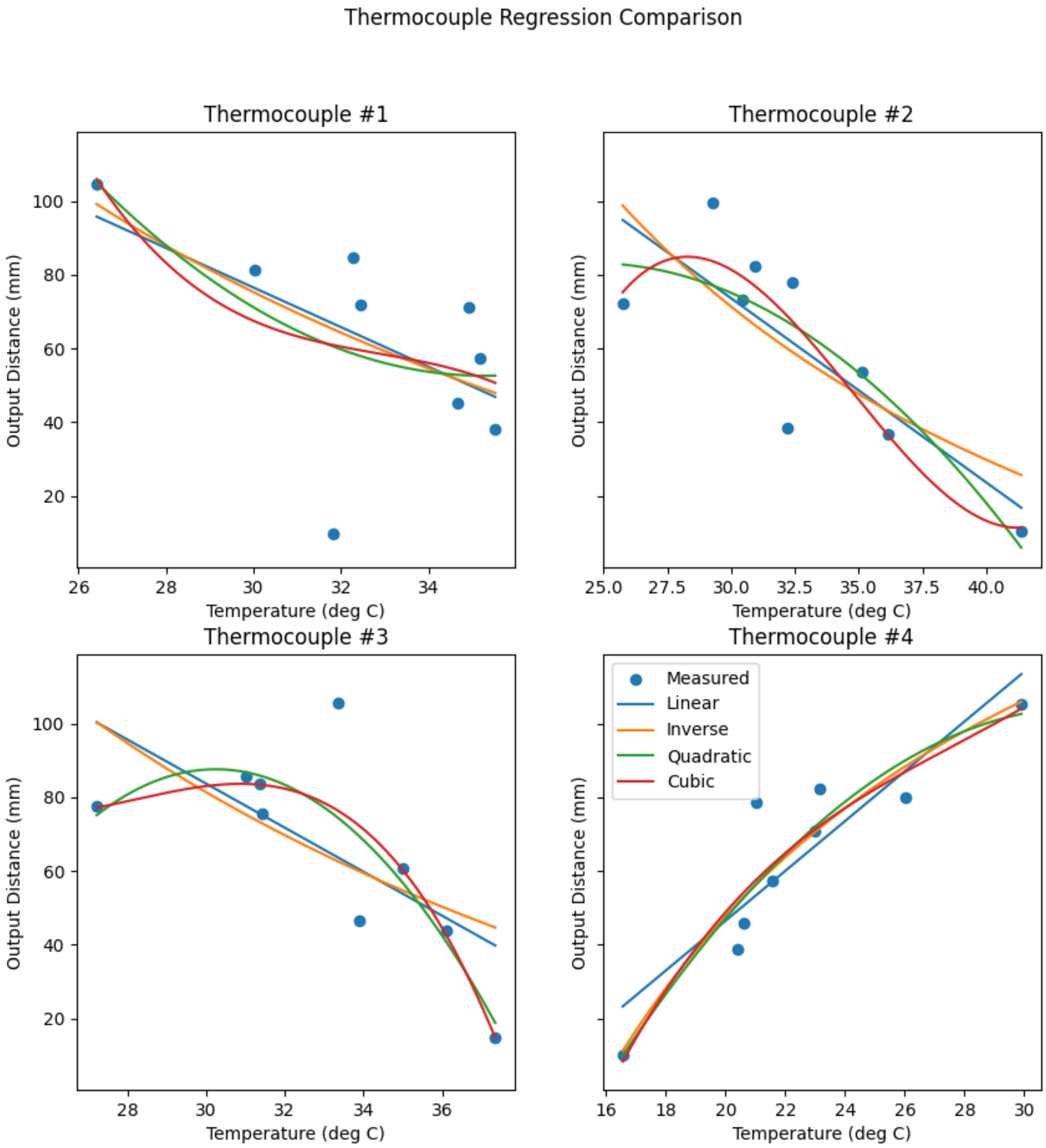


Figure 8: Thermocouple Regression Comparison

The cubic model had the highest coefficient of determination of the functions tested across all 4 thermocouples. By inspecting the plots however, it is clear that the relationship between temperature and distance has a large amount of variance and given the low number of sample points used to fit these models, the higher order functions are likely over-fit to the training data and would not generalize well to the test data. It was therefore decided that a linear model would be used for the thermocouples, as it is the simplest model and provides a reasonable approximation of the other models in the range of the training set data. Figure 9 shows the linear regression and resultant parameters.

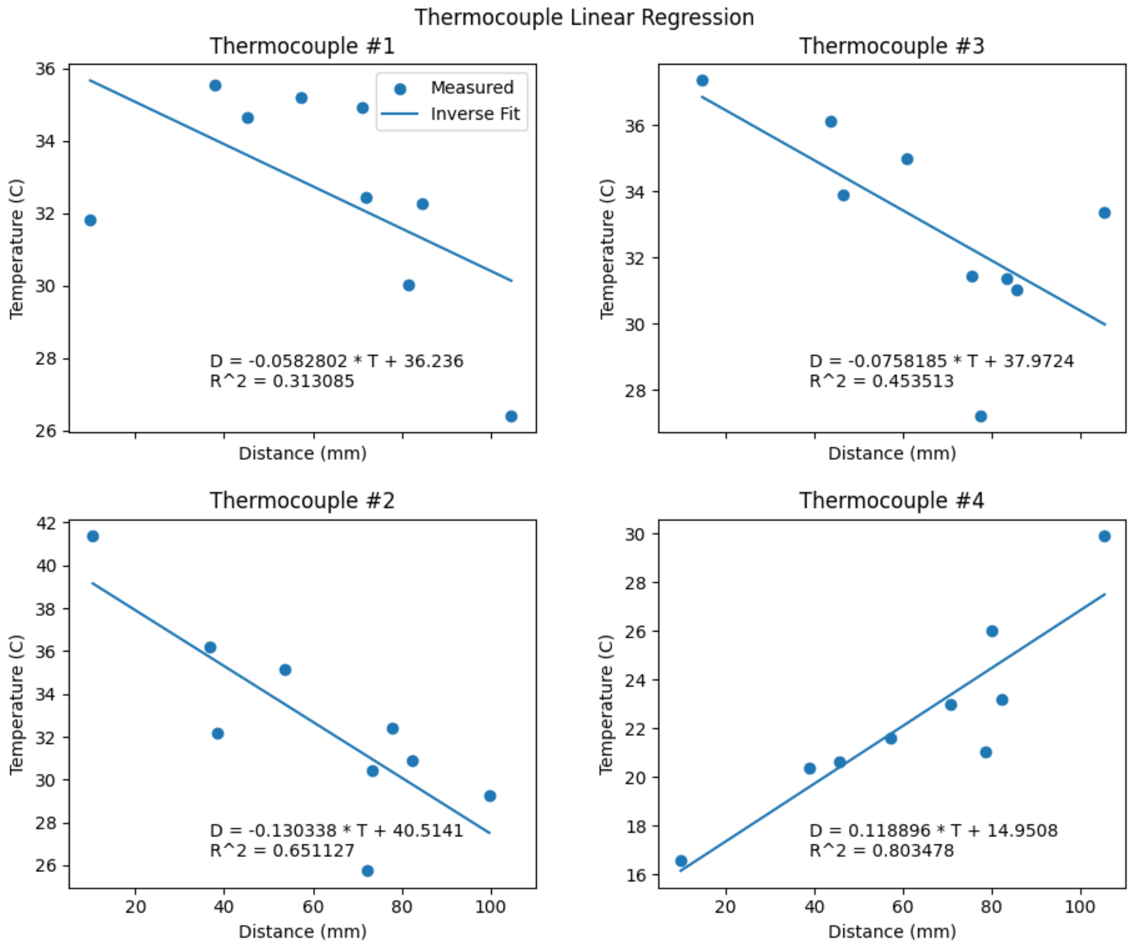


Figure 9: Thermocouple Regression to Linear Relationship

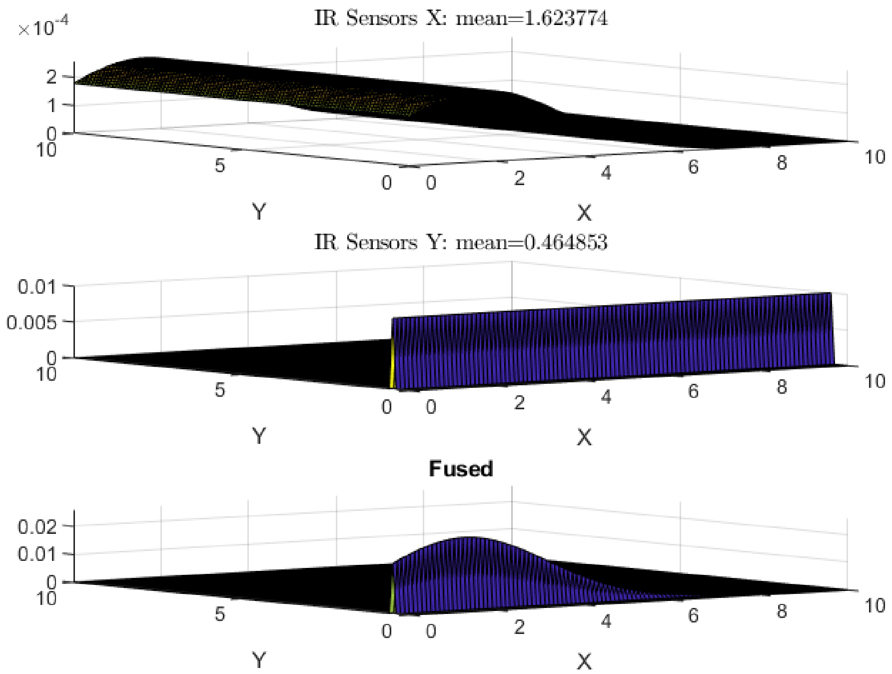
3 Model Calibration

Figure 10 below shows the distributions of the estimated distance to the heat source for training position 1 at coordinates $[7.4, 0.6]$. The IR x-axis measurements were inaccurate registering a mean distance of 1.62cm compared to the actual 7.4cm. The IR y-axis measurements were far better at 0.46cm compared to the actual 0.6cm. Thermocouple 1 registered a much larger distance considering the Peltier module was placed directly on top of it. Thermocouples 2 and 4 had acceptable distributions whereas Thermocouple 3 measured long.

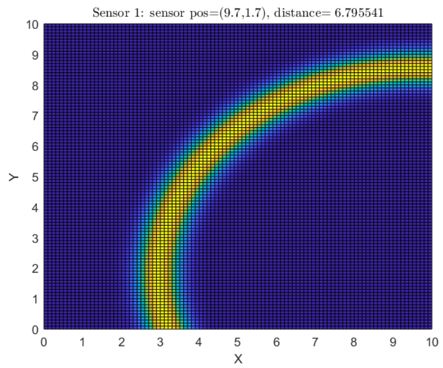
Figure 11 shows the distributions of the estimated distance to the heat source for training position 3 at coordinates $[0.012, 0.05]$. At this position both IR axis measured incorrect values with an x-axis mean of -0.84cm and a y-axis mean of 11.1cm. Similar to the first case where the thermocouple closest to the heat source registered a distance much further than the actual value, thermocouple 2 detected the heat source at a distance of 3.0cm in contrast to the actual 0cm distance. Once again, thermocouple 3 measures inaccurately this time falling short by 1.3cm. Thermocouple 1 was very accurate with an error of 0.1cm and Thermocouple 4 was similarly accurate with an average distance of 6.2cm for an actual distance of 6.0cm.

Figure 12 shows the distributions of the estimated distance to the heat source for training position 7 at coordinates $[7.2, 7.3]$. Once again, neither x-axis IR sensor determined accurate results leading to an off plot distribution. The y-axis IR sensors recorded a mean distance of 5.63cm, error of 1.57cm. This is the lowest error for the IR sensors among the samples shown. In this position thermocouple 3 had an actual distance of 0cm but recorded a mean distance of 3.59cm continuing the trend from the previous two positions. Thermocouple 1 possessed an error of 0.98cm, thermocouple 2 possessed an error of 2.16cm, and thermocouple 3 possessed an error of 1.21cm.

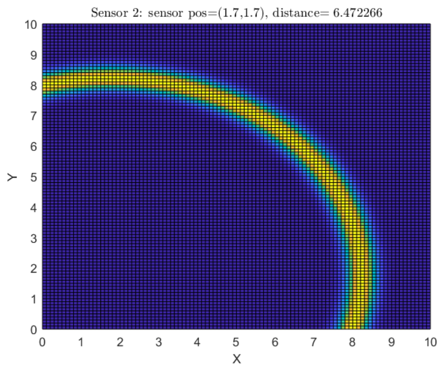
Figure Figure 13 shows the distributions of the estimated distance to the heat source for training position 7 at coordinates $[0.8, 7.5]$. Neither axis IR sensors determined an accurate reading for this position. Thermocouple 4 which should have determined a position of 0cm, instead returned a distance of 5.03cm. This trend implies that the linear model implemented to convert temperature measured by the thermocouples to distance from the heat source breaks down at higher temperatures. Thermocouple 1 had an error of 0.47cm, thermocouple 2 had an error of 0.37cm, and thermocouple 3 had an error of 2.33cm.



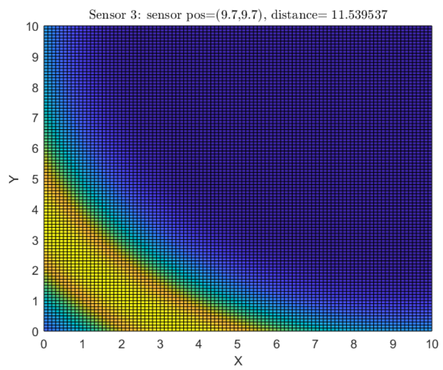
(a) IR Sensors at Position 1



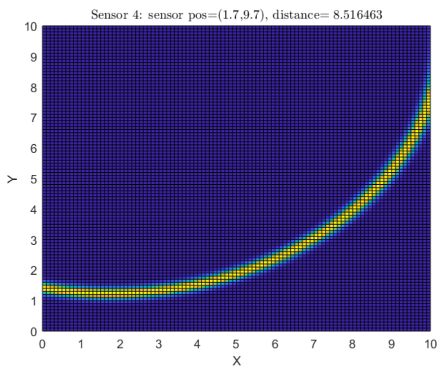
(b) Thermocouple 1 at Position 1



(c) Thermocouple 2 at Position 1

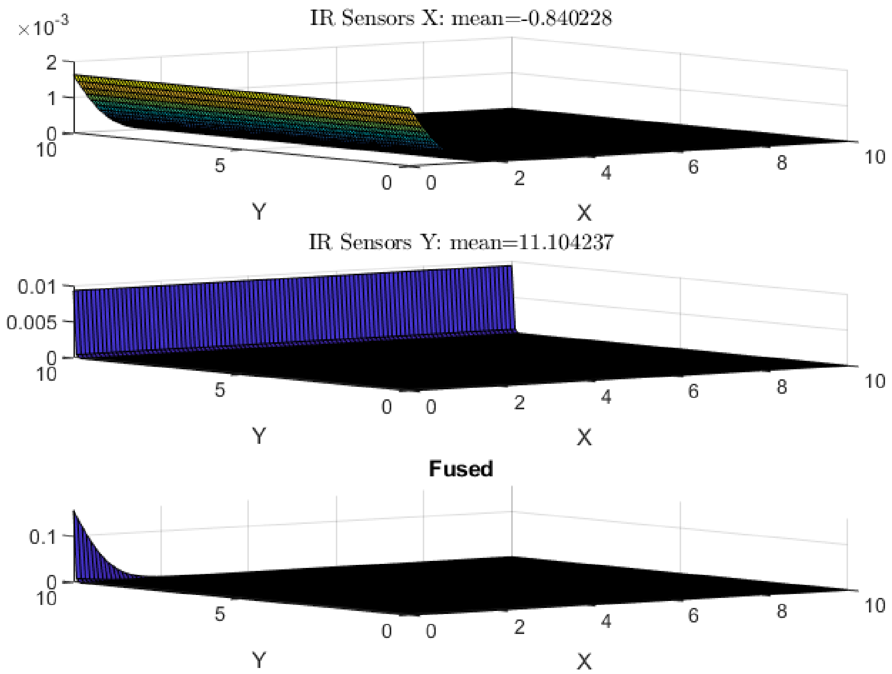


(d) Thermocouple 3 at Position 1

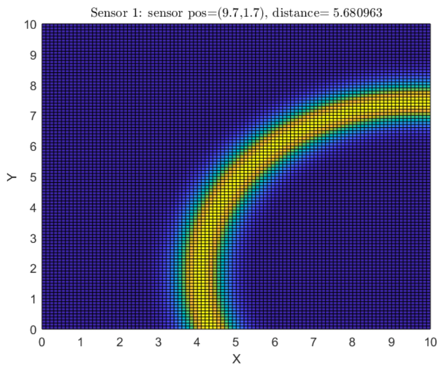


(e) Thermocouple 4 at Position 1

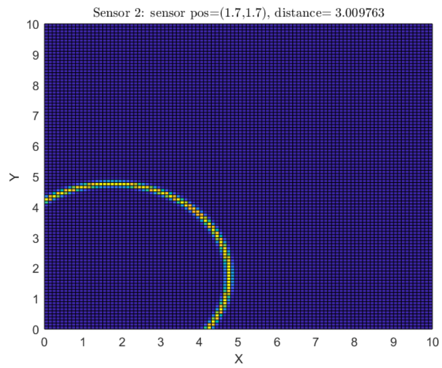
Figure 10: Position 1 Thermocouple Readings



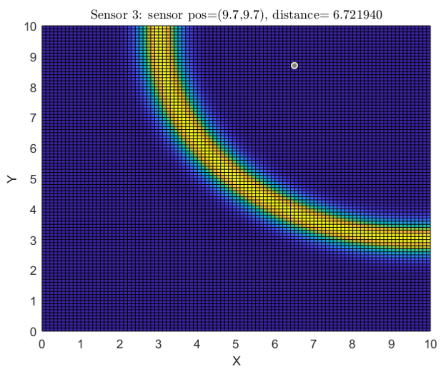
(a) IR Sensors at Position 3



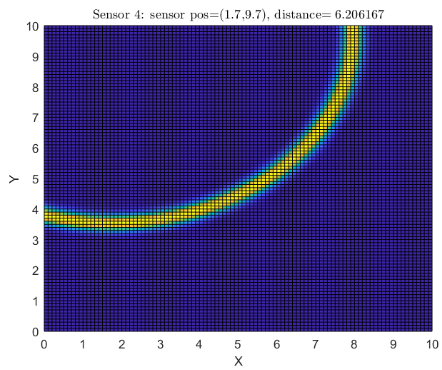
(b) Thermocouple 1 at Position 3



(c) Thermocouple 2 at Position 3

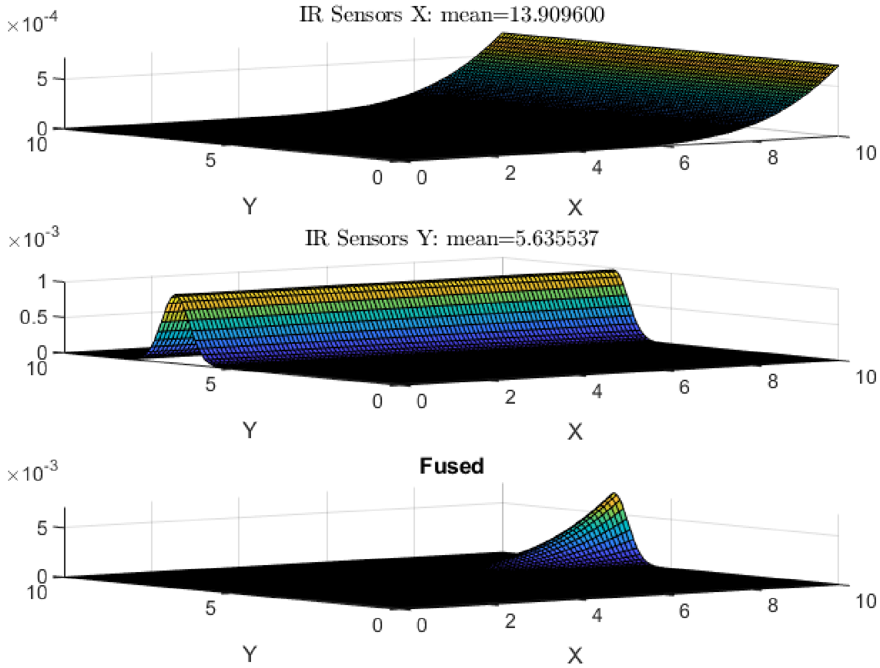


(d) Thermocouple 3 at Position 3

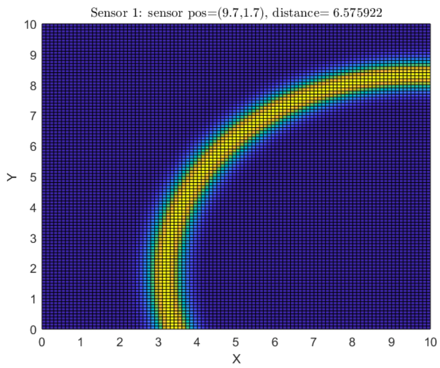


(e) Thermocouple 4 at Position 3

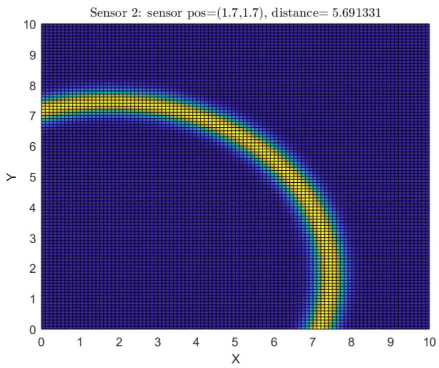
Figure 11: Position 3 Thermocouple Readings



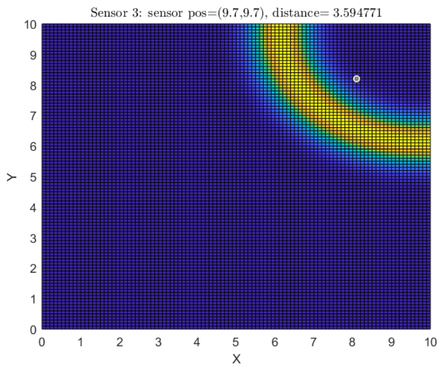
(a) IR Sensors at Position 7



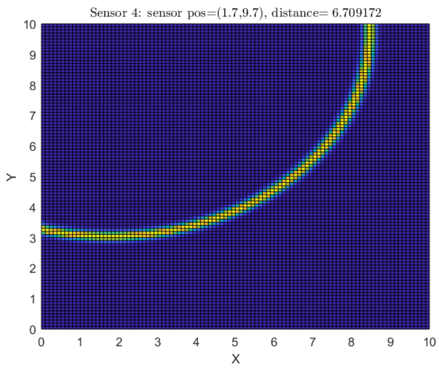
(b) Thermocouple 1 at Position 7



(c) Thermocouple 2 at Position 7

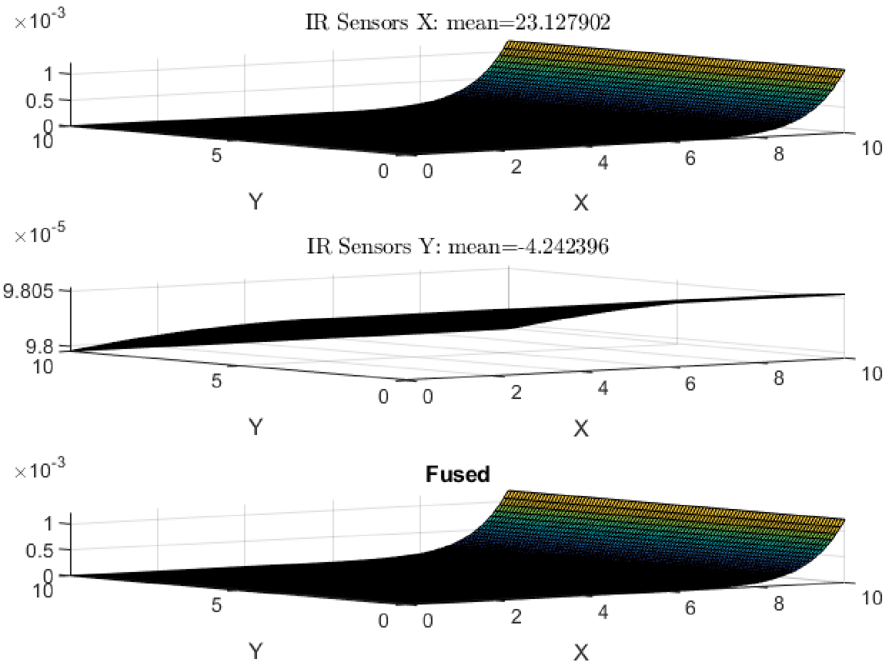


(d) Thermocouple 3 at Position 7

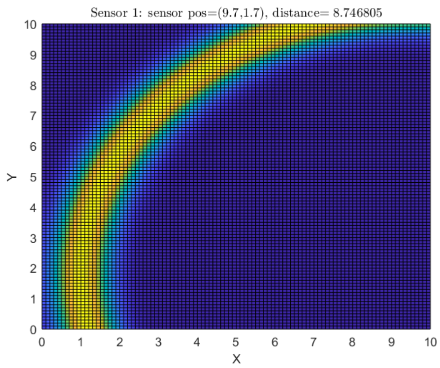


(e) Thermocouple 4 at Position 7

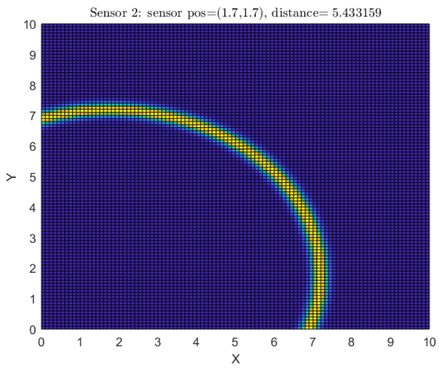
Figure 12: Position 7 Thermocouple Readings



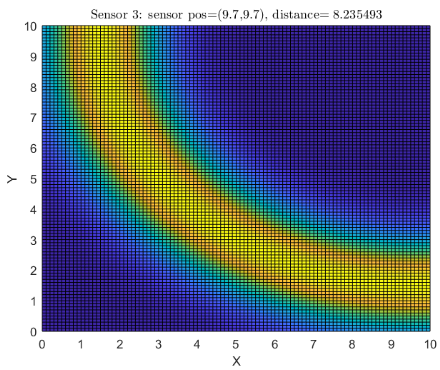
(a) IR Sensors at Position 9



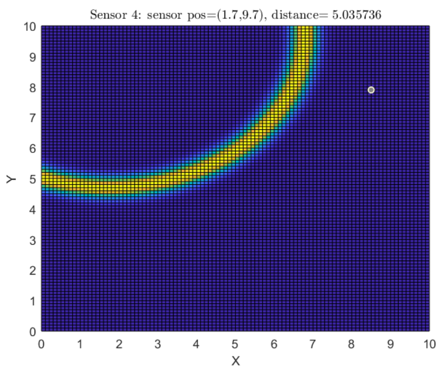
(b) Thermocouple 1 at Position 9



(c) Thermocouple 2 at Position 9



(d) Thermocouple 3 at Position 9



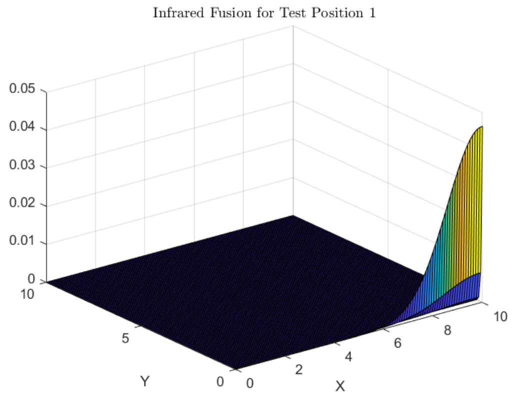
(e) Thermocouple 4 at Position 9

Figure 13: Position 9 Thermocouple Readings

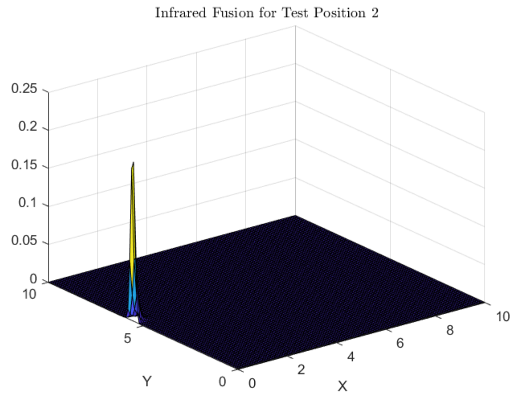
4 Model Evaluation

4.1 IR Sensor Fusion

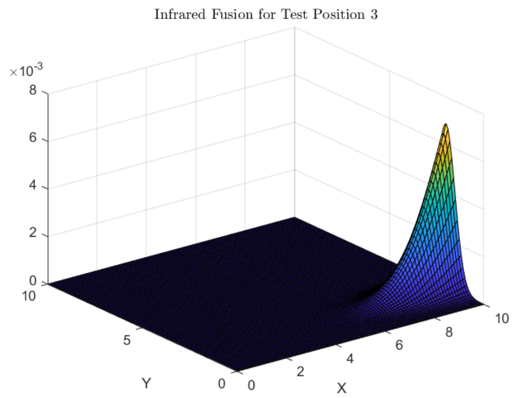
Using only the IR sensors, the positions of each of the test points was estimated. The probability distributions are shown in figure 14. In an attempt to improve the results, the IR sensor data was tested before being utilized in the model. Due to the positioning of the block in certain positions, it may not have been visible to one of the IR sensors. To account for this, the mean distance values were analyzed. In the case where both sensors registered the block at a distance within the range of the test area, Bayesian fusion was performed to fuse the data from both sensors. In a case where one sensor did not detect the block within the test area, the distribution of the other sensor was used. In the case where neither sensor detected the block within the test area, the sensor with the better reading was used. Despite this effort, the model still appears to perform rather poorly. Test point 1 in figure 14a is predicted to be in the corner of the plate at $[10, 0]$, near training point 1 (refer to table **??**). The actual location of test point 1 was $[5.5, 2.4]$. The predicted position of point 2 shows to be roughly $[0, 5]$, far from the measured $[1.9, 4.0]$, but relatively close in the Y direction. This is likely due to the long range IR sensors measuring in the X direction not seeing the block, therefore registering an invalid distance that is corrected by the filtering to be 0. The short range sensors are able to pick up the block and do a decent job of providing an accurate measurement. Test point 3 is also predicted to be in the $[10, 0]$ corner. With a measured position of $[5.5, 7.0]$, the IR sensors again fail to correctly place the block on the plate. Overall, the performance of the IR model alone is very poor, failing to accurately measure the blocks position in 7 of 8 cases.



(a) Test Point 1



(b) Test Point 2

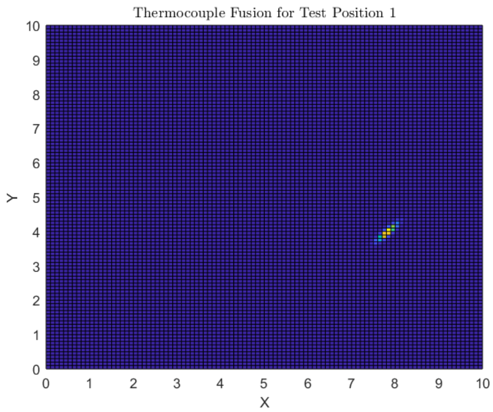


(c) Test Point 3

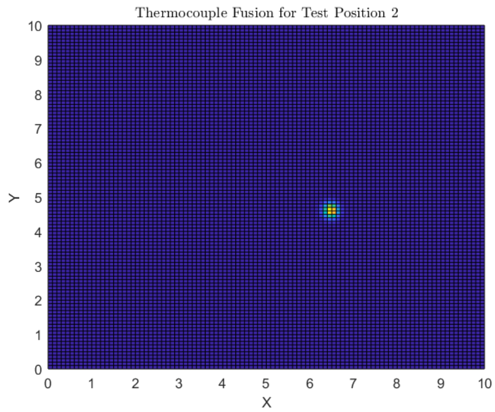
Figure 14: IR Fusion Readings

4.2 Thermocouple Sensor Fusion

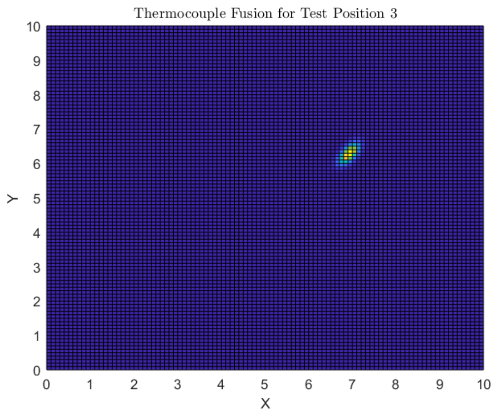
Using only the thermocouple sensors, the positions of each of the test points was estimated. The probability distributions are shown in figure 15. The distributions show promising results, producing 3 specific points of likelihood for the measured test points. Test point 1, measured at $[5.5, 2.4]$ is predicted to be at roughly $[7.9, 4.1]$. This measurement is within the margin of error of the blocks width. Accounting for the blocks dimensions of 3cmx3cm, the measured position is within 1cm of the blocks center in both the X and Y directions. Test point 2 measured at $[1.9, 4.0]$ is predicted to be at $[6.5, 4.5]$. The X measurement is wildly incorrect, predicting the wrong side of the plate, while the Y measurement is accurate to within 0.5cm. Test point 3 measured at $[5.5, 7.0]$ is predicted to be at $[7.0, 6.4]$. In this case, accounting for the blocks dimensions, puts the X prediction perfectly in line but the Y prediction 3cm off. Overall, sensor fusion from the thermocouples alone is not a reliable method of location the block and Peltier cell on the plate.



(a) Test Point 1



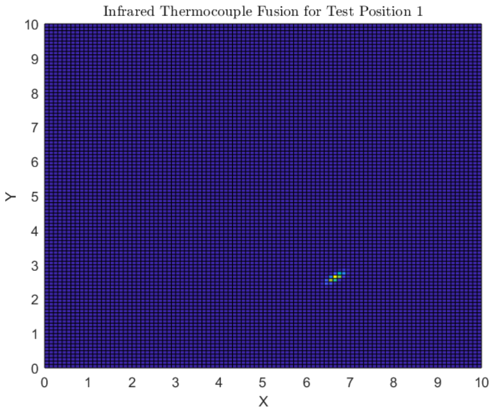
(b) Test Point 2



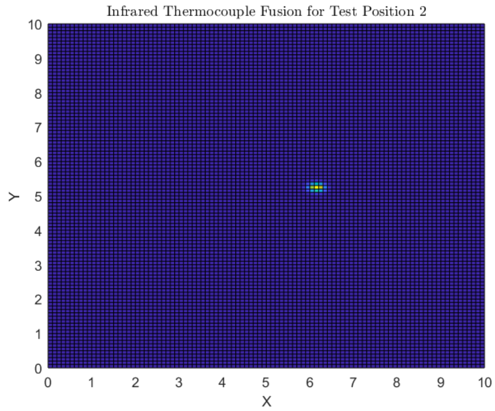
(c) Test Point 3

Figure 15: Thermocouple Fusion Readings

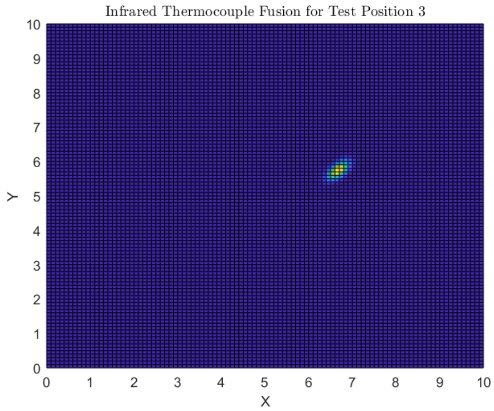
4.3 Infrared and Thermocouple Sensor Fusion



(a) Test Point 1



(b) Test Point 2



(c) Test Point 3

Figure 16: IR and Thermocouple Fusion Readings

Figure 16 above shows the combined fusion between the IR sensors and the thermocouples for the three test points. The fusion of the sensors provided a very small distribution for each of these points indicating that there was very little overlap between each of the distributions involved. Test position 1 was estimated to be at coordinates $[6.7, 2.7]$. The measured position was $[5.5, 2.4]$. This equates to an error of 1.29cm. Test position 2 was estimated to be at coordinates $[6.1, 5.4]$. Comparing this to the measured coordinates of $[1.9, 4.0]$ results in an error of 4.43cm, significantly larger. Test position 3 was estimated to be at coordinates $[6.7, 5.7]$. It was measured at a position of $[5.5, 7.0]$ resulting in an error of 1.77cm. While the fusion of these sensors provided a highly consistent result, it was an inaccurate one. Likely with superior IR data the result could have been greatly improved. The advantage of performing this type of sensor fusion is a more reliable result is achieved through the consideration of multiple sensors recording different types of data. The multiplication of valid gaussian probability distributions results in a higher certainty of measurements, as was seen with the thermocouples. The disadvantage is that there is no benefit when the measurements of the sensors are inconsistent with one another as seen with the IR sensors.

Since both sensors in the same axis often differed significantly in their measurements, the only thing that could be done was to take the better of the two sets which resulted in highly unreliable results.

5 References

[1] Arash Arami, Lyndon E. Tand, Mo Shushtari, and Ehsan Tahvilian. Lab 3 – complete sensor fusion system. University of Waterloo - ME 546 Multi-Sensor Data Fusion — Winter 2024.

[2] Austin Milne, Jude Bennett, and William Ancich. Lab 1 - sensor modeling - report. Student Report — University of Waterloo - ME 546 Multi-Sensor Data Fusion — Winter 2024.

6 Appendix

6.1 Full Code Listings

Listing 1: Sensor Calibration Script — Sensor-Calibration.py

```python
# %% [markdown]
# # Lab 3 - Machine Learning
#
# | Authors |
# |---------|
# | Austin Milne |
# | Jude Bennett |
# | William Ancich |
#
# ## IR Distance Sensor Profiling
# Four sensors were used. Two short range sensors (4cm-30cm) and two long range sensors (20cm-150cm). Each pair of sensors (short + short & long + long) were measured at 6 distances that are relevant to their range. Based on Lab 1, the sensor are profiled against the function:
#
# $$
# D = \frac{a}{V} + b \\~\\
# \begin{aligned}
#     D & = \text{Distance} \\
#     V & = \text{Voltage} \\
#     a, \, b & = \text{Constants} \\
# \end{aligned}
# $$

# %%
# Necessary Libraries
import json
import copy
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mergedeep import merge
from scipy.stats import norm
from scipy.optimize import curve_fit
from scipy.spatial.distance import cdist
from sklearn.metrics import r2_score
import pathlib
import statistics

# Read Raw Data
with open(r"data/data.json") as f:
    data = json.load(f)

# %% [markdown]
# ### Check Voltages
# Plot the voltages to visually inspect that the sensors are behaving as expected.

# %%
# Plot the voltage readings for each distance
def plot_voltages(data, name):
    distances = list(data.keys())
    distances.sort(key=lambda x: float(x))
    count=len(distances)
    columns = 3
    rows = math.ceil((count - count%3)/3)
    rows = rows+1 if count%3 else rows
    fig, axs = plt.subplots(rows, columns, sharex=True, sharey=True, figsize=(3*columns, 3*rows))
    fig.suptitle(f'{name} Voltage Readings')
    for i, distance in enumerate(distances):
```

```python
        # Create dataframe with each sensor's data
        df = pd.DataFrame({
            "Time": np.arange(start=0, stop=len(data[distance]["data"][0])*(1/20), step=(1/20)),
            "Left": data[distance]["data"][0],
            "Right": data[distance]["data"][1]
        })
        # Plot each sensor's data
        # sub = axs[(i-(i%3))/3, i%3]
        sub = axs[int((i-(i%3))/3), i%3]
        for sensor in ["Left", "Right"]:
            sub.scatter(df.index, df[sensor], label=sensor, s=0.75)
        sub.set_title(f"{distance} cm", fontsize=10)
        if (i%3 == 0): sub.set_ylabel("Voltage")
        if (i==0): sub.legend()
    plt.show()
    fig.savefig(f"out/plots/{name} Voltage Readings.png")

# Plot voltages for short and long sensors
dcs = data["Calibrations"]["Short"]
dcl = data["Calibrations"]["Long"]

plot_voltages(dcs, "Short Sensors")
plot_voltages(dcl, "Long Sensors")

# %% [markdown]
# ## IR Sensor Calibration
# Determine the relevant constants for each sensor to fit the inverse relationship between voltage and distance.

# %%
# Run regression on range of functions for each sensor
sensors = ["Left", "Right"]
sets = ["Short", "Long"]
dc = data["Calibrations"]
sensor_params = {}
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
fig.tight_layout(pad=4.0)
for i, set in enumerate(sets):
    for j, tc_id in enumerate(sensors):
        readings = pd.DataFrame({
            "Distance": [float(dist) for dist in dc[set].keys()],
            "Voltage": [np.mean(dc[set][dist]["data"][j]) for dist in dc[set]]
        })
        readings.sort_values(by="Distance", inplace=True)
        readings.reset_index(drop=True, inplace=True)

        # Inverse Regression
        inverse = lambda x, a, b : a/x + b
        inv_a, inv_b = curve_fit(inverse, readings["Distance"], readings["Voltage"])[0]
        inv_r2 = r2_score(readings["Voltage"], inverse(np.asarray(readings["Distance"]), inv_a, inv_b))
        merge(sensor_params, {
            set: {
                tc_id: {
                    "a": inv_a,
                    "b": inv_b,
                    "r2": inv_r2,
                }
            }
        })

        # Plot the sensor data and regressions lines
        pts = np.linspace(min(readings["Distance"]), max(readings["Distance"]), 1000)
        sub = axs[i, j]
        sub.scatter(readings["Distance"], readings["Voltage"], label="Measured")
        sub.plot(pts, inverse(np.asarray(pts), inv_a, inv_b), label=f"Inverse Fit")
        sub.set_title(f"{set} {tc_id} Sensor")
```

```python
        sub.set_xlabel("Distance (cm)")
        sub.set_ylabel("Output Voltage (V)")
        sub.text(0.23 + j*0.48, 0.8 - i*0.47, f"D = {inv_a:3.6} / V + {inv_b:3.6}\nR^2 = {
            inv_r2:3.6}", fontsize=10, transform=plt.gcf().transFigure)
        sub.legend()

pathlib.Path("out/plots").mkdir(parents=True, exist_ok=True)
plt.savefig(f"out/plots/IR Sensor Fits.png")
plt.show()

# Print Sensor Properties Formula in pretty table
print("Sensor Properties")
functions = []
for set in sensor_params:
    for tc_id in sensor_params[set]:
        functions.append({
            "Sensor": f"{set} {tc_id}",
            "Function": f"D = {sensor_params[set][tc_id]['a']:3.6} / V + {sensor_params[set
                ][tc_id]['b']:3.6}",
            "R^2": sensor_params[set][tc_id]['r2']
        })
functions = pd.DataFrame(functions)
print(functions.to_string(index=False))



# %% [markdown]
# ## Thermocouple Calibration and Profiling
# Four thermocouples were used. All the thermocouples were measured with the heater in 9
    different positions.

# %% [markdown]
# ### Data Parsing and Organization
# Read in and clean up the data for the thermocouples.

# %%
# Hardcode measured distances for reference points
CUBE_L = 30 # Length/Width of the usable area
displacement_to_square_center = np.array([CUBE_L/2, CUBE_L/2])

# As measured from the axes (edge of box) to the closest face of block
training_pts = np.array([
    (74, 6),
    (39, 12),
    (12, 5),
    (71, 38),
    (40, 40),
    (10, 38),
    (72, 73),
    (40, 75),
    (8, 75)
])
# Test Points
test_pts = np.array([
    (55, 24),
    (19, 40),
    (55, 70)
])

# Visualize the training and test points
center_training_pts = training_pts + displacement_to_square_center
plate_corners = np.array([
    (0,0),
    (0,115),
    (115, 0),
    (115,115)
])

```

```
# Create scatter plots
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
plt.scatter(plate_corners[:,0],       plate_corners[:,1],       label='Plate Corners')
plt.scatter(tc_pts[:,0],              tc_pts[:,1],              label='Thermocouple
    Locations')
plt.scatter(center_training_pts[:,0], center_training_pts[:,1], label='Training Points')
plt.scatter(test_pts[:,0],            test_pts[:,1],            label='Test Points')
plt.xlabel('X-axis (mm)')
plt.ylabel('Y-axis (mm)')
plt.title('Experimental Setup Layout')
plt.legend()

# Show plot
plt.grid(True)
plt.savefig("out/plots/Experimental Setup Layout.png")
plt.show()

# %%

# Channels 0-3
tc_training_dist = cdist(training_pts + displacement_to_square_center, tc_pts)
tc_test_dist = cdist(test_pts + displacement_to_square_center, tc_pts)

# Channels 4,5
ir_y_training_dist = np.stack((training_pts[:, 1] + 50 - 1/2*CUBE_L, ) * 2, axis=1)
ir_y_test_dist = np.stack((test_pts[:, 1] + 50 - 1/2*CUBE_L, ) * 2, axis=1)

# Channels 6,7
ir_x_training_dist = np.stack((training_pts[:, 0] + 300 - 1/2*CUBE_L, ) * 2, axis=1)
ir_x_test_dist = np.stack((test_pts[:, 0] + 300 - 1/2*CUBE_L, ) * 2, axis=1)

# Combine all distances
training_dist = np.concatenate([tc_training_dist, ir_y_training_dist, ir_x_training_dist],
    axis=1)
test_dist = np.concatenate([tc_test_dist, ir_y_test_dist, ir_x_test_dist], axis=1)

# Labels for the data
training_labels = [f'P{i+1}' for i in range(training_dist.shape[0])]
test_labels = [f'A{i+1}' for i in range(test_dist.shape[0])]

# Create training labels
training_labels = [f'P{i+1}' for i in range(training_dist.shape[1])]

# Get the mean temperatures for each thermo couple
training_mean_voltages = copy.deepcopy(data["TrainingData"])
for i, pt_key in enumerate(data["TrainingData"].keys()):
    # Shape is (num_channels, num_readings) or (8, 100)
    channel_readings = np.array(data["TrainingData"][pt_key]["data"])
    channel_mean_voltages = np.mean(channel_readings, axis=1)
    # We're overwriting the "time" and "data" fields with
    # a single array that contains the mean voltage readings for each channel
    training_mean_voltages[pt_key] = channel_mean_voltages

# Create a dataframe for the mean voltages
tr_mv_df = pd.DataFrame(training_mean_voltages)
tr_mv_df.index.name = "channel"
tr_mv_df.columns.name = "set"
tr_mv_df.drop(columns="control", inplace=True) # Ignore the control measurment

# Convert the mean voltages to temperatures
def voltage_to_temperature(V_out):
    return (V_out-1.25)/0.005
tr_tc_mv_df = tr_mv_df.loc[0:3, :]
tr_tc_T_df = voltage_to_temperature(tr_tc_mv_df).T

# Training distances dataframe
tr_tc_d_df = pd.DataFrame(tc_training_dist, index=tr_tc_T_df.index, columns=tr_tc_T_df.
    columns)
```

```


# %% [markdown]
# ### Check Voltages
# Plot the voltages to visually inspect that the sensors are behaving as expected.

# %%
# Plot the voltage readings of the thermocouples for each position: 1-9
tc_data = data["TrainingData"]
distances = list(tc_data.keys())
distances.sort(key=lambda x: x)
distances.remove("control")
count=len(distances)
columns = 3
rows = math.ceil((count - count%3)/3)
rows = rows+1 if count%3 else rows
fig, axs = plt.subplots(rows, columns, sharex=True, sharey=True, figsize=(3*columns, 3*rows)
    )
fig.suptitle(f'Thermocouple Voltage Readings')
for i, distance in enumerate(distances):
    # Create dataframe with each sensor's data
    df = pd.DataFrame({
        "Time": np.arange(start=0, stop=len(tc_data[distance]["data"][0])*(1/20), step
            =(1/20)),
        "0": tc_data[distance]["data"][0],
        "1": tc_data[distance]["data"][1],
        "2": tc_data[distance]["data"][2],
        "3": tc_data[distance]["data"][3],
    })
    # Plot each sensor's data
    # sub = axs[(i-(i%3))/3, i%3]
    sub = axs[int((i-(i%3))/3), i%3]
    for sensor in ["0", "1", "2", "3"]:
        sub.scatter(df.index, df[sensor], label=f'#{sensor}', s=0.75)
    sub.set_title(f"Position {distance[1]}", fontsize=10)
    if (i%3 == 0): sub.set_ylabel("Voltage")
    if (i==0): sub.legend()
fig.savefig(f"out/plots/Thermocouple Voltage Readings.png")
plt.show()

# %% [markdown]
# ### Compare Regressions
# Run a series of different regressions to better understand how the sensor behave.

# %%
sensor = dict()
fig, axs = plt.subplots(2, 2, sharex=False, sharey=True, figsize=(5*2, 5*2))
fig.suptitle(f'Thermocouple Regression Comparison')
for i, tc_id in enumerate(tr_tc_T_df.columns):
    temperatures = tr_tc_T_df[tc_id]
    distances = tr_tc_d_df[tc_id]

    x = temperatures
    y = distances

    sensor[tc_id] = dict()

    # Linear Regression
    linear = lambda x, m, b : m*x + b # Lambda for linear regression
    lin_m, lin_b = curve_fit(linear, x, y)[0] # Fit the data to the linear model
    lin_r2 = r2_score(y, linear(np.asarray(x), lin_m, lin_b)) # Get the R^2 value
    sensor[tc_id]["linear"] = [lin_m, lin_b, lin_r2] # Store the linear regression results

    # Inverse Regression
    inverse = lambda x, a, b : a/x + b # Lambda for inverse regression
    inv_a, inv_b = curve_fit(inverse, x, y)[0] # Fit the data to the inverse model
    inv_r2 = r2_score(y, inverse(np.asarray(x), inv_a, inv_b)) # Get the R^2 value
    sensor[tc_id]["inverse"] = [inv_a, inv_b, inv_r2] # Store the inverse regression results
```

```python

    # Quadratic Regression
    quadratic = lambda x, a, b, c : a*x**2 + b*x + c # Lambda for quadratic regression
    quad_a, quad_b, quad_c = curve_fit(quadratic, x, y)[0] # Fit the data to the quadratic model
    quad_r2 = r2_score(y, quadratic(np.asarray(x), quad_a, quad_b, quad_c)) # Get the R^2 value
    sensor[tc_id]["quadratic"] = [quad_a, quad_b, quad_c, quad_r2] # Store the quadratic regression results

    # Cubic Regression
    cubic = lambda x, a, b, c, d : a*x**3 + b*x**2 + c*x + d # Lambda for cubic regression
    cub_a, cub_b, cub_c, cub_d = curve_fit(cubic, x, y)[0] # Fit the data to the cubic model
    cub_r2 = r2_score(y, cubic(np.asarray(x), cub_a, cub_b, cub_c, cub_d)) # Get the R^2 value
    sensor[tc_id]["cubic"] = [cub_a, cub_b, cub_c, cub_d, cub_r2] # Store the cubic regression results

    # Create a plot of the sensor data and regressions lines
    pts = np.linspace(min(x), max(x), 1000)
    row = int(i%2)
    column = int((i-(i%2))/2)
    sub = axs[column, row]
    sub.scatter(x, y, label="Measured")
    sub.plot(pts, linear(np.asarray(pts), lin_m, lin_b), label=f"Linear")
    sub.plot(pts, inverse(np.asarray(pts), inv_a, inv_b), label=f"Inverse")
    sub.plot(pts, quadratic(np.asarray(pts), quad_a, quad_b, quad_c), label=f"Quadratic")
    sub.plot(pts, cubic(np.asarray(pts), cub_a, cub_b, cub_c, cub_d), label=f"Cubic")
    sub.set_title(f"Thermocouple #{tc_id+1}")
    sub.set_xlabel("Temperature (deg C)")
    sub.set_ylabel("Output Distance (mm)")
    if(i == 3): sub.legend()

    # Create a table of the regression results
    results = pd.DataFrame({
        "Model": ["Linear", "Inverse", "Quadratic", "Cubic"],
        "R^2": [lin_r2, inv_r2, quad_r2, cub_r2],
        "Parameters": [
            f"y = {lin_m:3.8}x + {lin_b:3.8}",
            f"y = {inv_a:3.8}/x + {inv_b:3.8}",
            f"y = {quad_a:3.8}x^2 + {quad_b:3.8}x + {quad_c:3.8}",
            f"y = {cub_a:3.8}x^3 + {cub_b:3.8}x^2 + {cub_c:3.8}x + {cub_d:3.8}"
        ]
    })
    pd.set_option('display.width', 1000)
    print(f"{tc_id} Sensor Regression Results")
    print(results.to_string(index=False))

# Save the regression results to a file
pathlib.Path("out/plots").mkdir(parents=True, exist_ok=True)
plt.savefig(f"out/plots/Thermocouple Regressions.png")
plt.show()

# %% [markdown]
# ### Linear Regression
# Run detailed Linear Regression to determine the constants for the thermocouples.

# %%
# Run regression on range of functions for each sensor
sensors = []
sensor_params = {}
fig, axs = plt.subplots(2, 2, sharex=True, figsize=(10, 8))
fig.tight_layout(pad=4.0)
fig.suptitle(f'Thermocouple Linear Regression')
for i, tc_id in enumerate(tr_tc_T_df.columns):
    readings = pd.DataFrame({
        "Distance": tr_tc_d_df[tc_id],
        "Temp": tr_tc_T_df[tc_id],
```

```python
    })
    readings.sort_values(by="Distance", inplace=True)
    readings.reset_index(drop=True, inplace=True)

    # Inverse Regression
    inverse = lambda x, a, b : a*x + b
    inv_a, inv_b = curve_fit(inverse, readings["Distance"], readings["Temp"])[0]
    inv_r2 = r2_score(readings["Temp"], inverse(np.asarray(readings["Distance"]), inv_a, inv_b))
    merge(sensor_params, {
        set: {
            tc_id: {
                "a": inv_a,
                "b": inv_b,
                "r2": inv_r2,
            }
        }
    })

    # Plot the sensor data and regressions lines
    pts = np.linspace(min(readings["Distance"]), max(readings["Distance"]), 1000)
    column = int(i%2)
    row = int((i-(i%2))/2)
    sub = axs[column, row]
    sub.scatter(readings["Distance"], readings["Temp"], label="Measured")
    sub.plot(pts, inverse(np.asarray(pts), inv_a, inv_b), label=f"Inverse Fit")
    sub.set_title(f"Thermocouple #{tc_id+1}")
    sub.set_xlabel("Distance (mm)")
    sub.set_ylabel("Temperature (C)")
    sub.text(0.2 + row*0.48, 0.6 - column*0.47, f"D = {inv_a:3.6} * T + {inv_b:3.6}\nR^2 = {inv_r2:3.6}", fontsize=10, transform=plt.gcf().transFigure)
    if (i==0): sub.legend()

# Save and show the plot
pathlib.Path("out/plots").mkdir(parents=True, exist_ok=True)
plt.savefig(f"out/plots/Thermocouple Sensor Fits.png")
plt.show()

# Print Sensor Properties Formula in pretty table
print("Sensor Properties")
functions = []
for set in sensor_params:
    for tc_id in sensor_params[set]:
        functions.append({
            "Sensor": f"{set} {tc_id}",
            "Function": f"D = {sensor_params[set][tc_id]['a']:3.6} * T + {sensor_params[set][tc_id]['b']:3.6}",
            "R^2": sensor_params[set][tc_id]['r2']
        })
functions = pd.DataFrame(functions)
print(functions.to_string(index=False))
```