12/1/2015

Sensor Output Correction Algorithm Project Report

ENGR 5520: Sensors & Actuators

Yamei Xiao & Jonathon Kreska
UNIVERSITY OF DETROIT MERCY

Preparing the Sensor Data for MATLAB

The sensor output was provided to us in an excel spreadsheet. Since our tool we used to manipulate the data was MATLAB, we first had to import the three columns into our MATLAB workspace.

Since Part 1 needed a sweep from 0mm to 16mm, at the same time as the import process, we separated the given sweep into two 0mm to 16mm voltage arrays.

This preparation step was moved to its own function because we found it easier to recreate the data each time in separate script for each part. This is preferred as opposed to having a long script for the entire project. This helped us isolate each part and prevent the confusion of using previous data.

Part 1: Creating the Sweep

To follow the instructions, we had to generate the sweep from 0mm to 16mm and back to 0mm. This was done via a simple concatenation of the arrays we created. Figure 1 shows the sweep. Since time is arbitrary in this part, we defined a step of 10µsec which gave a total sweep time of 106.6msec.

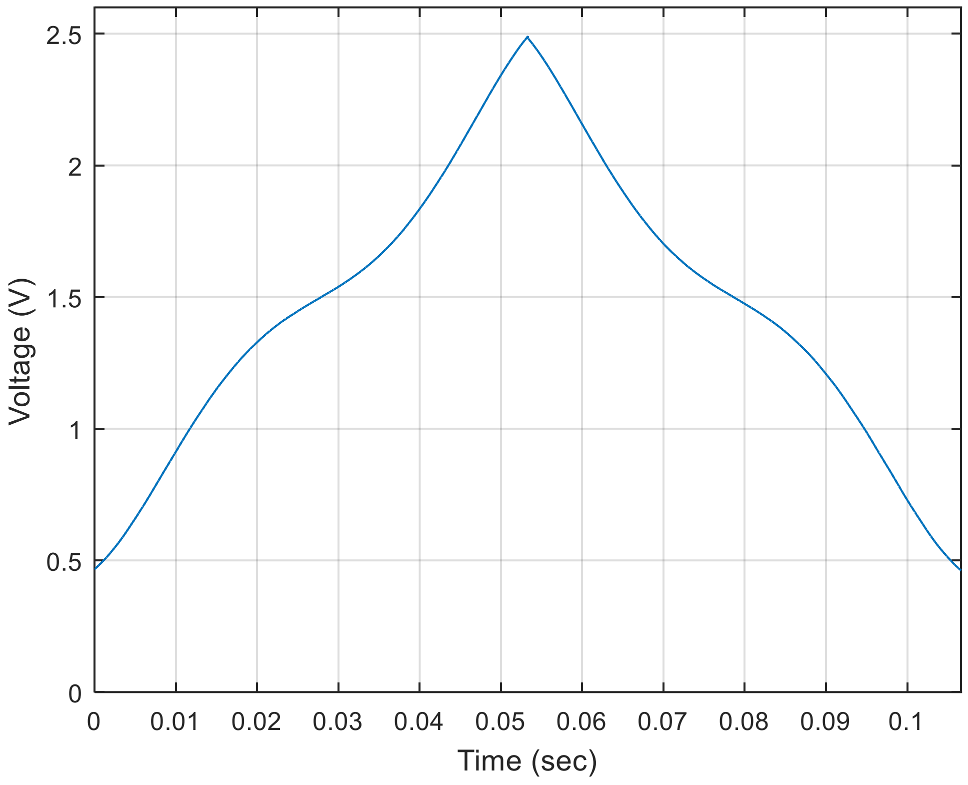

Figure 1: Sensor Output from Time Sweep

Part 2: Measure Resolution

The resolution is defined as how much change in distance gives us a change in digital output. First we designed a model for the ATD (Analog to Digital) convertor. We found the digital output for each voltage according to the ATD characteristics as seen in Figure 2. We are using a 10-bit ATD with 3.3V max. We chose this because these values are common in manufacturing and fit our scenario well.

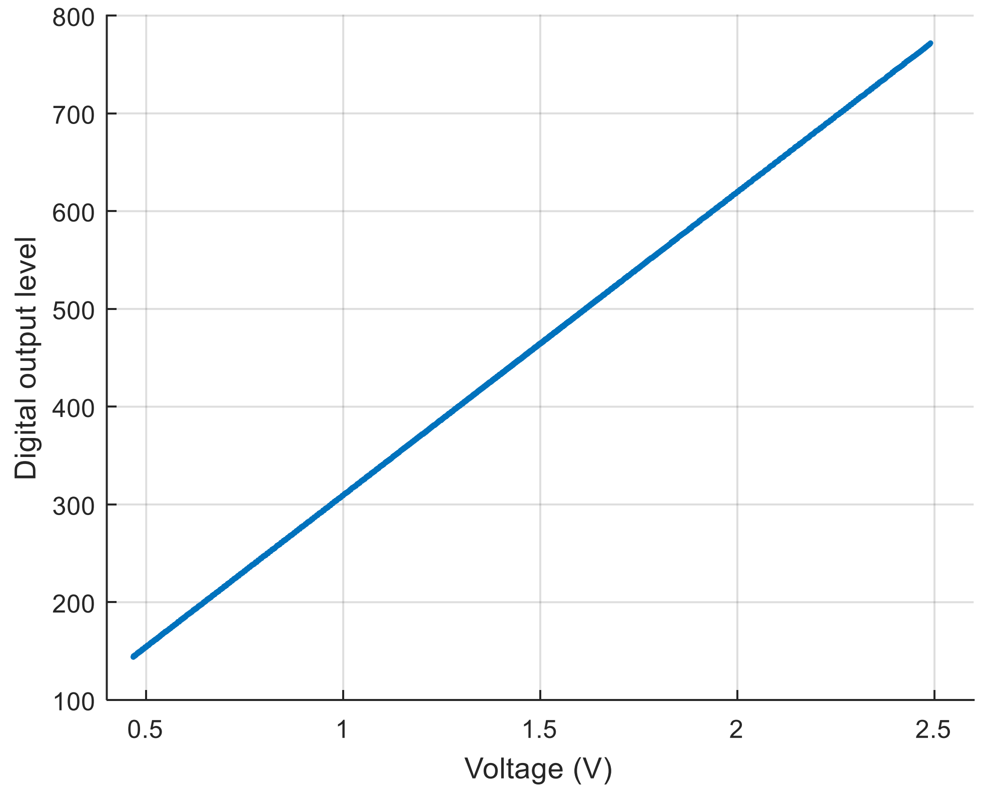

Figure 2: Digital output vs. analog voltage

By zooming-in to Figure 2, we can see the different voltages having the same digital output level. This is demonstrated in Figure 3.

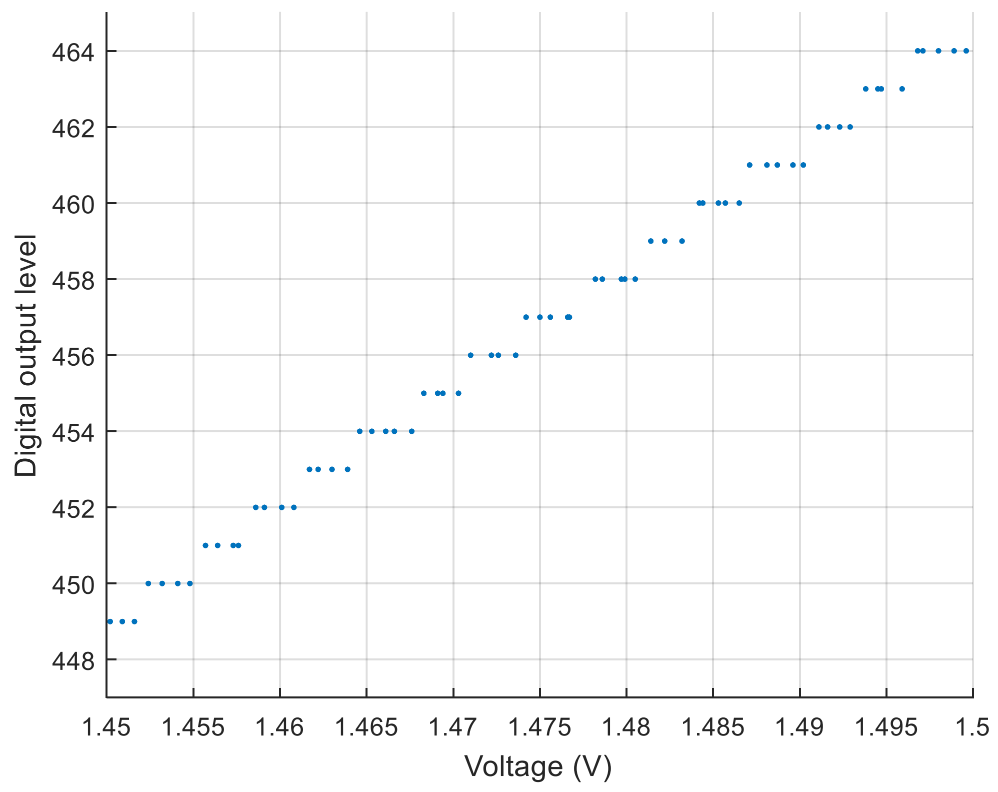

Figure 3: Zoom in of Digital output vs. analog voltage

Then we found the function between distance and digital output. Several distances as the input have the same output voltage, so these distances share the same digital output level as seen in Figure 4.

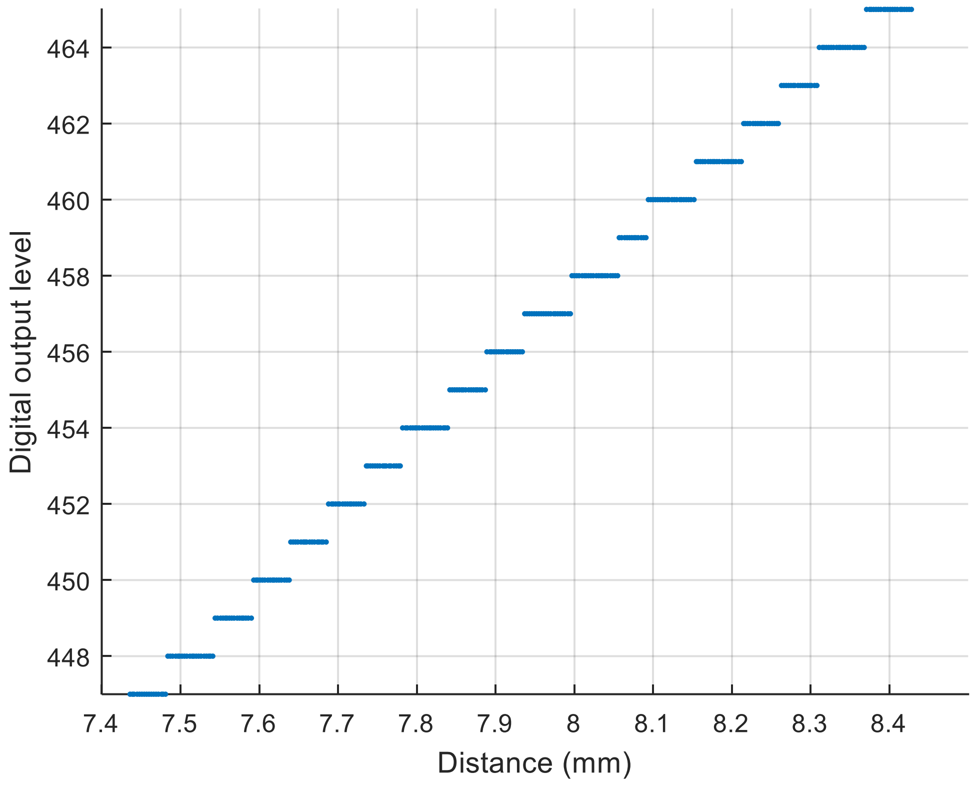

Figure 4: Digital output vs. distance input

Next we determined all the distances where the digital output changed. We also calculated the gap between distances where the digital output changed; which is the resolution. Figure 5 shows the resolution of each of the sweeps of the sensor. Resolutions above the 70 micron requirement are displayed in red, otherwise the passing values are blue. The left side of the figure is the first half of the sweep, the right side is second half of the sweep.

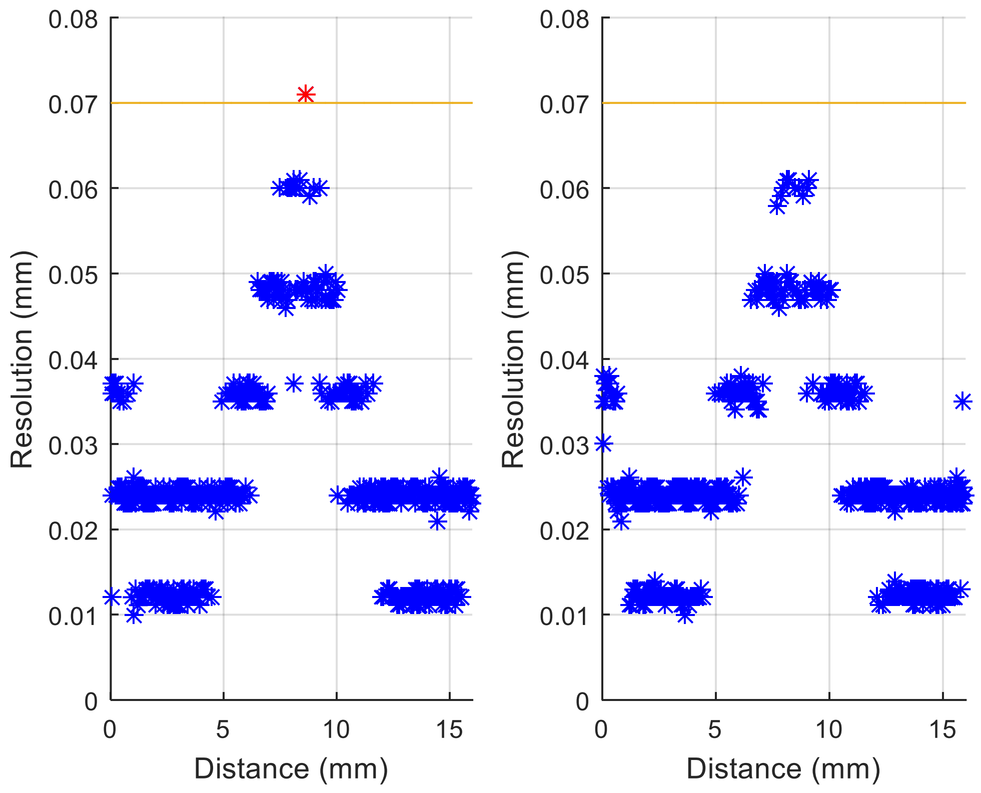

Figure 5: Resolution vs. Distance for 10-bit ATD

The resolution is related to the number of bits the ATD uses. In our case, we determined that a 10-bit resolution only had one value that failed. This is acceptable since the next step up would be an 11 or 12-bit resolution ATD, which will most likely be more expensive an uncommon.

To demonstrate what will happen with an 8-bit ATD, we have displayed the output on Figure 6. Notice that most of the values are above the 70 micron requirement.

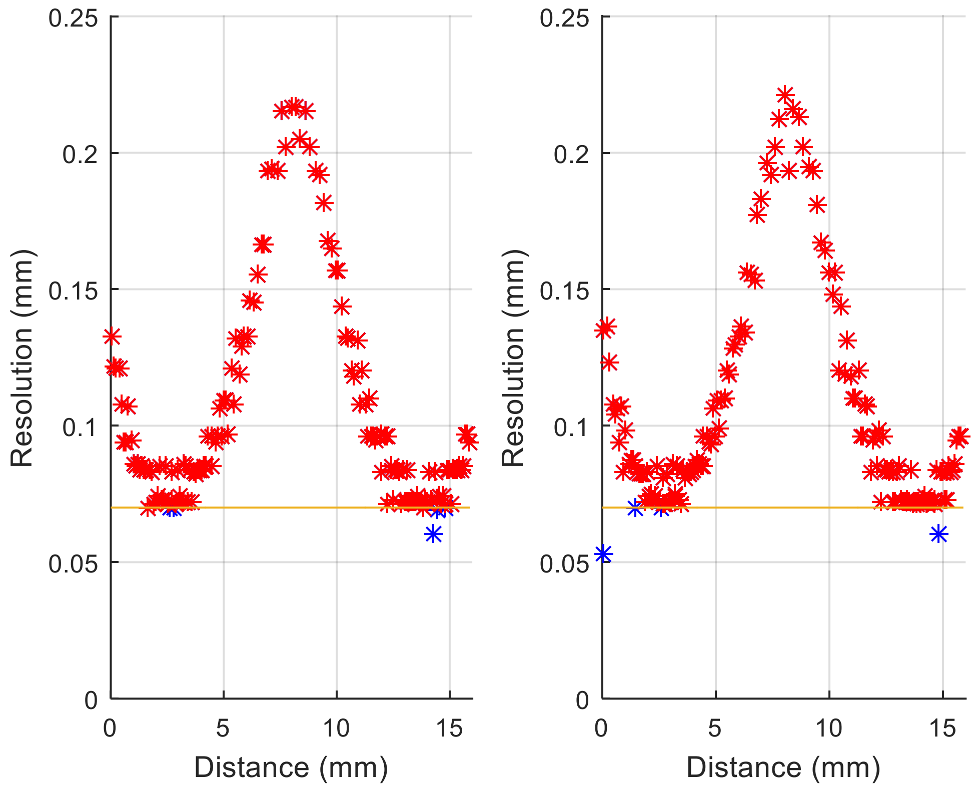

Figure 6: Resolution vs. Distance for 8-bit ATD

Part 3: Symmetric Point

Upon evaluating how to determine the symmetric point of the sensor, we found that the sensor sweep has heuristics. What we mean is that for the same distance, there are two voltage values. This makes working with the data very difficult and not intuitive. To fix this, we used a MATLAB function, "fit", to approximate a polynomial that gives a good average between the two voltage values. A 7th order polynomial was the highest we could go without warnings or errors from MATLAB. The order of the polynomial is only for the understanding the model and characteristic evaluation; it will never be integrated into anything. The polynomial is plotted in Figure 7 along with the original sensor output values.

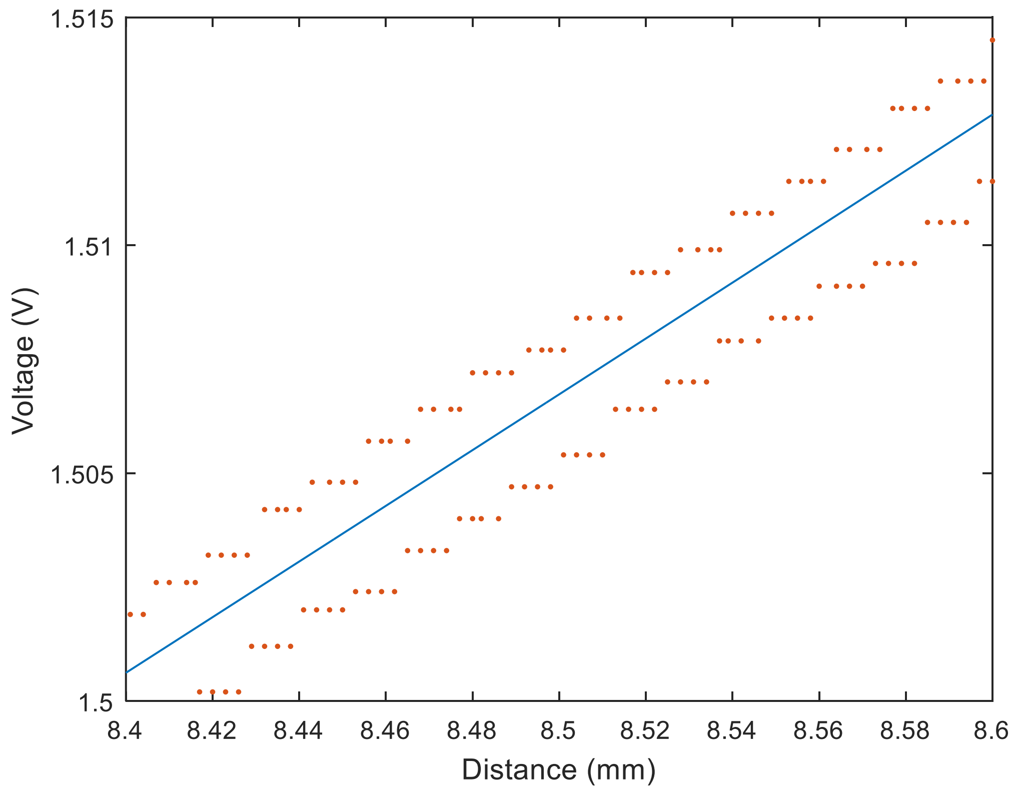

Figure 7: Sensor Model vs. Original Output Data

Now that we have a polynomial function, and its associated output that is useable, we had to determine the symmetric point of the sensor. Observing the original curve, the symmetric point should be between 8 and 8.5mm; the exact symmetric point must be determined.

Using the concept of 180 degree symmetry, we tried exact symmetry first. To determine whether the curve is symmetric about a point, we used the following method: step through the polynomial and compare the point one step to right and left of the center point. If their position has 180 degree symmetry, expand the search range. Continue to expand the search range until the points do not match. Save the central point and the range in an array. Then reset the range and move to the next point.

For exact symmetry, the symmetric range was unfortunately very small. We set out to define a tolerance where the right side of the curve can differ from the left side of the curve about the current central point. When this improved operation was performed, the symmetry ranges became larger. This called for a definition for finding a point and range where it encompasses a sizeable portion of the 0mm to 16mm range. To aid in this search, we set a fixed voltage tolerance of 0.010V.

Using the method in MATLAB, it determined that the sensor model is symmetric about the point 8.232mm from 1.033mm to 15.431mm. This range covers 89.98% of the curve, almost 90% as seen in Figure 8. It would not be reasonable to find a tolerance that encompasses 99% of the sensor model, since the model itself may has parts that are not symmetric. This is why we performed some trial and error until we arrived at a reasonable symmetric point and range.

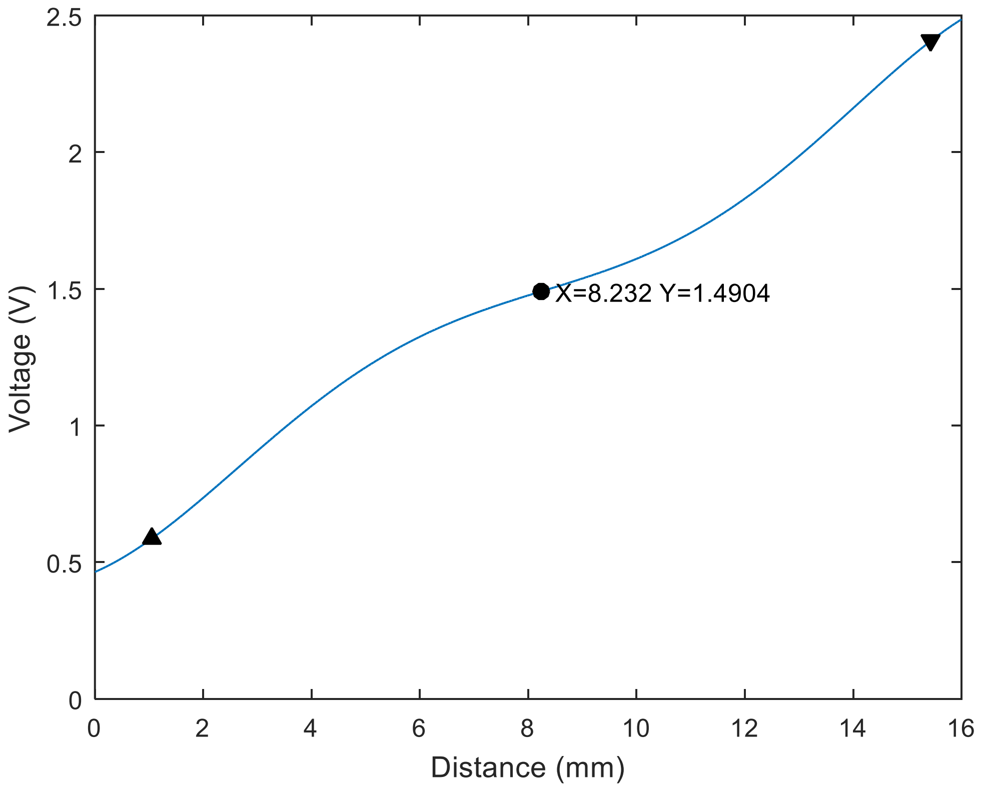

Figure 8: Symmetric Point and Range of the Sensor Model

Part 4: Error of Sensor

The error of the sensor is represented by the following equation:

$$Error = |V_{actual} - V_{intended}|$$

Figure 9 shows the relationship between distance and the error of the sensor.

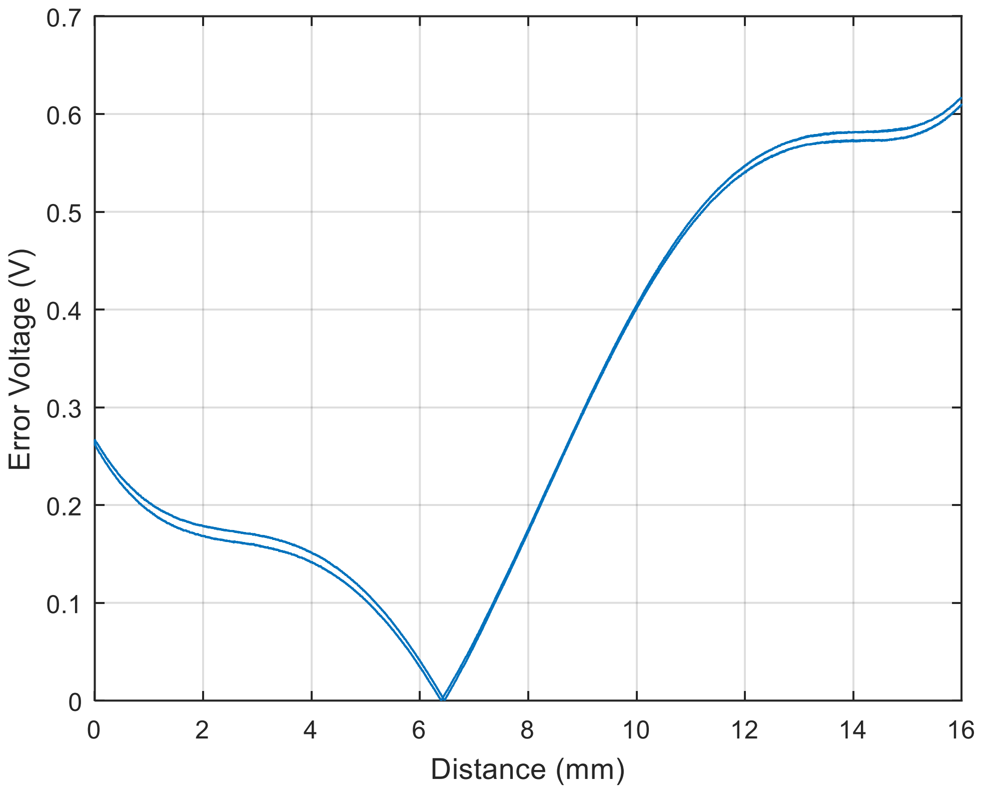

Figure 9: Distance vs. Error

Part 5: Distribution of Resolution and Error

Resolution

Figure 10 shows the distribution of resolution.

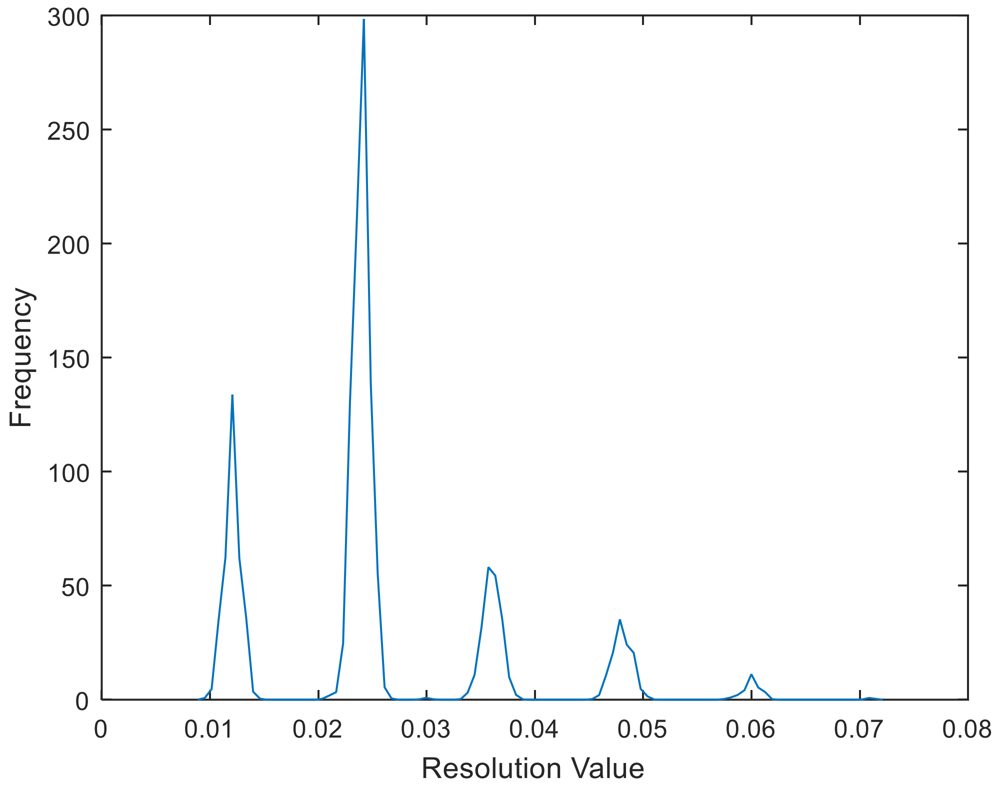

Figure 10: Distribution of Resolution

Taking the calculated resolution values, we obtained the following parameters:

a. Mean = 0.025505

b. Median = 0.024000

c. Mode = 0.024000

d. Std_Dev = 0.010814

Error

Figure 11 shows the distribution of error.

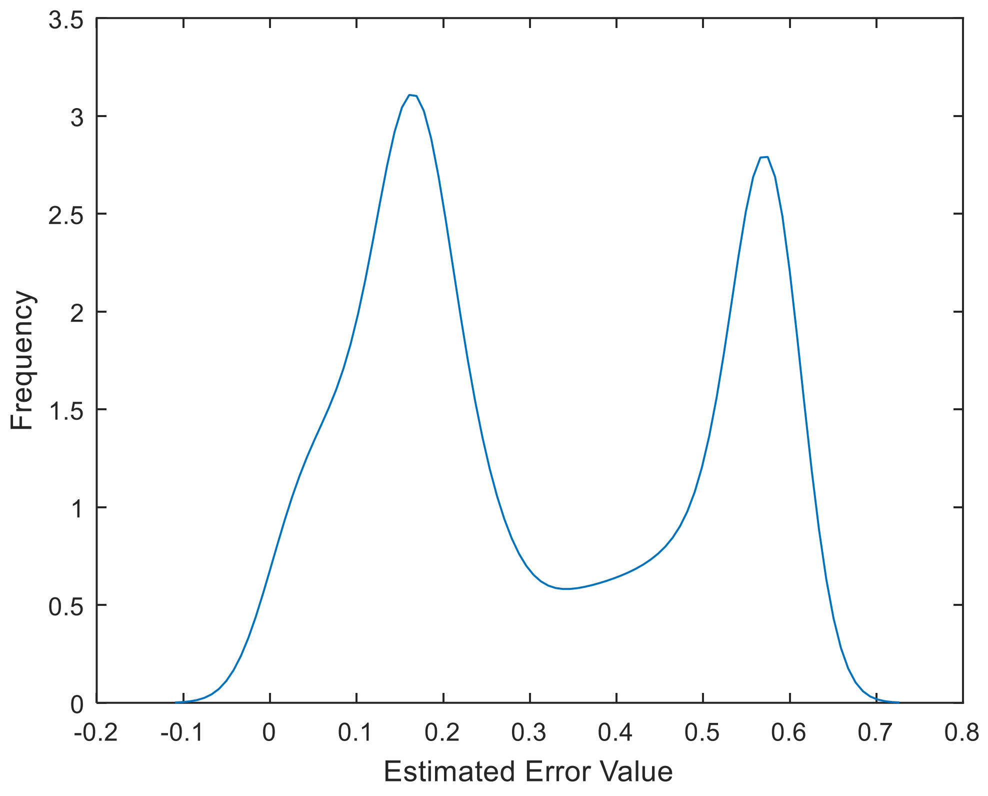

Figure 11: Error Distribution

Taking the calculated error values, we obtained the following parameters:

a. Mean = 0.308519

b. Median = 0.227797

c. Mode = 0.572256

d. Std_Dev = 0.199165

Sigma-level for 10mV Error

To determine the sigma-level, we need to calculate the PPM, which is found using the following equation:

$$PPM = \frac{10^6 \times \# \, of \; Defects}{Population}$$

Our PPM is 986871.717929. Using an online PPM to sigma table, we found our sigma-level to be 0.

Sigma-level for 70 microns Threshold for Resolution

Our PPM is 797.448166. Using an online PPM to sigma table, we found our sigma-level to be 4.7.

Part 6/7: Algorithm to Shift Symmetric Point and Adjust to Desired Characteristic

We decided the best choice was do Part 7 then Part 6 to have the best results. The method and explanation is described below.

First we shifted the intended sensor data linear model to intersect with the symmetric point of the sensor model as seen in Figure 12.

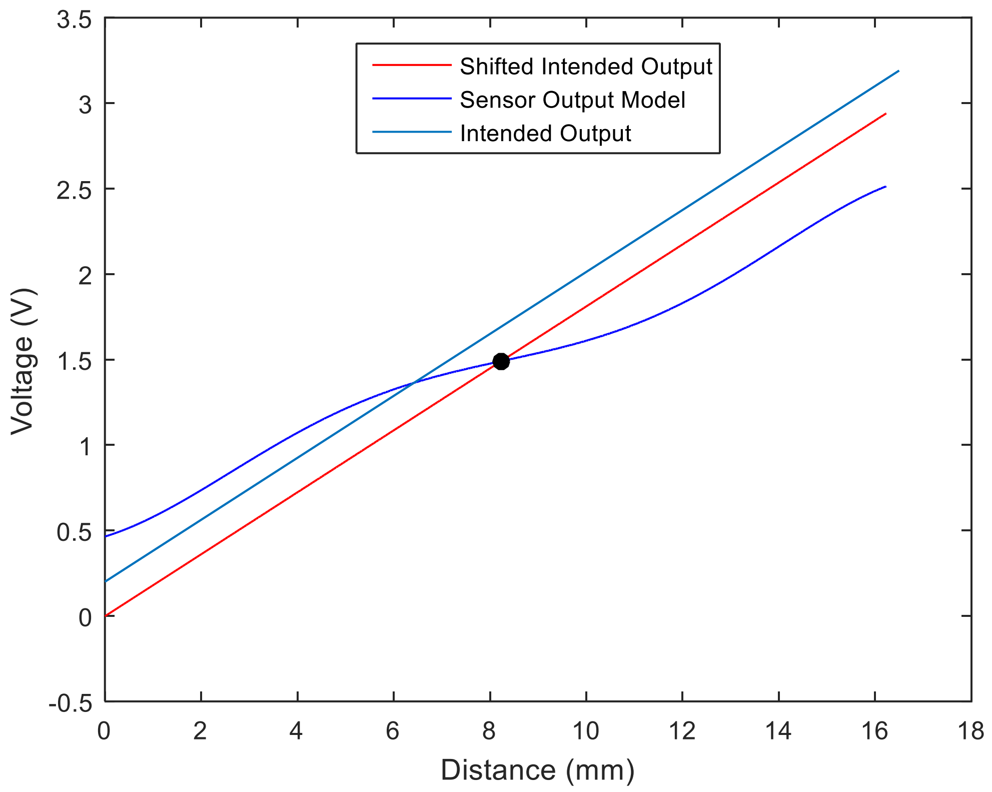

Figure 12: Intended Sensor Output Shifted to Symmetric Point

Next we used the “polyfit” function in MATLAB to determine a polynomial equation that would change the sensor output values to the shifted intended output. A 7th order polynomial gave a very good results and no warnings or errors from MATLAB. The fitted curve is shown in Figure 13.

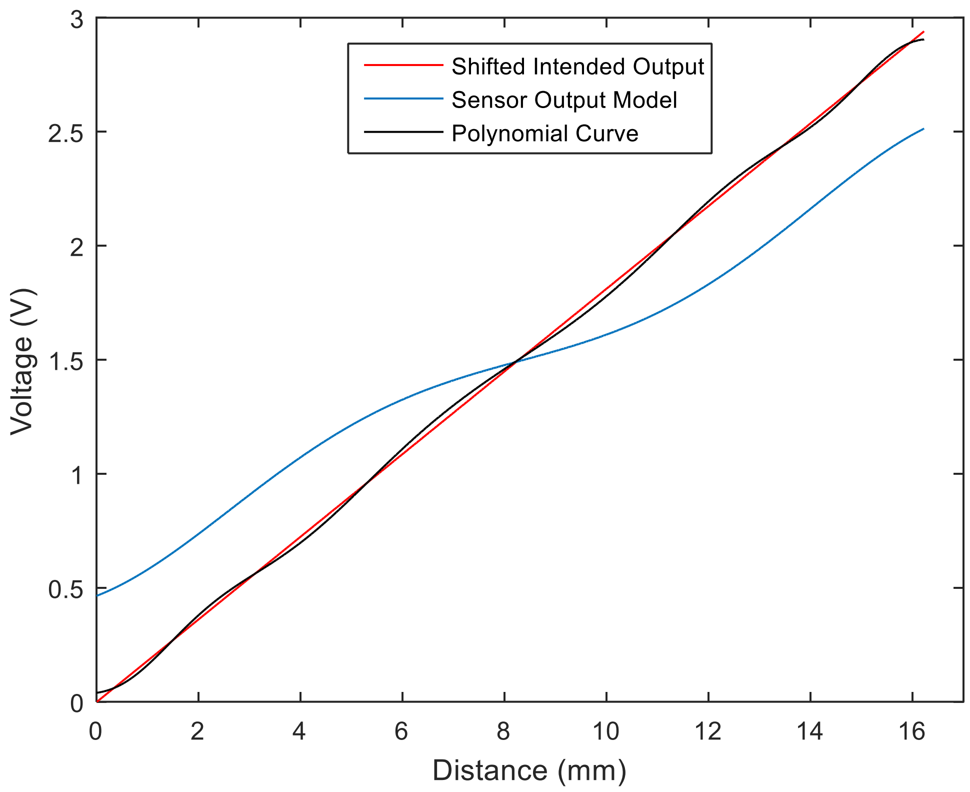

Figure 13: Polynomial Output Curve compared to Shifted Intended Output

Now we shifted the Symmetric Point of the Polynomial Output Curve to the Symmetric Point of the original Intended Sensor Output. We predicted the values larger than 16mm on the sensor model. Since we used the original sensor data to develop the model, the predicted values should still have similar low error compared to the original sensor data and the 0mm to 16mm data.

The model of the sensor was shifted to the left and up, then the negative values were removed to have a sweep between 0mm and 16mm. This results of this shift is shown in Figure 14. Performing the data correction operation in this order does not matter, but it helped us understand and develop the method to rectify the output of the sensor.

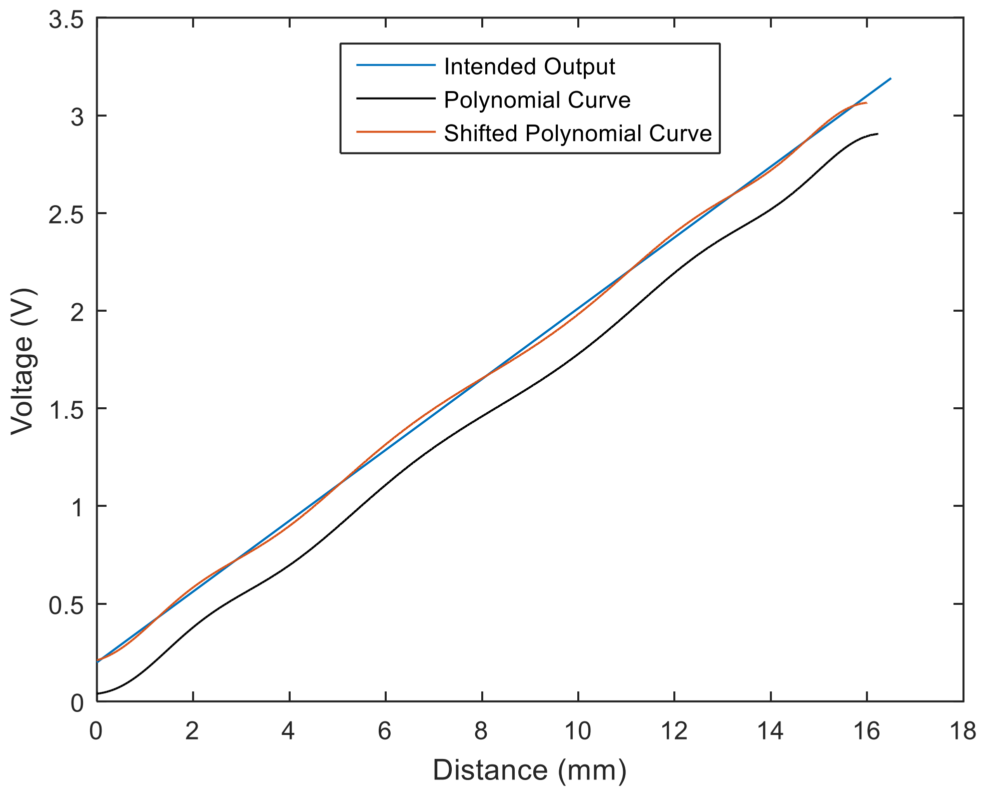

Figure 14: Polynomial Output shifted to Intended Output Symmetric Point

Part 8: Overall Algorithm Method

Figure 14 demonstrates that our algorithm shifts the Sensor Output Symmetric point to the Intended Symmetric point as well as correcting the error in the sensor output. Shifting the symmetric point will help with the heuristics of the sensor via an X and Y shift and the data correction via a 7th order polynomial gives an output very close to the desired output.

The following is the exact algorithm to modify the sensor output to the desired characteristic:

Given Sensor Output is a,

$$b = -2.1398a^7 + 22.3241a^6 - 94.9028a^5 + 211.2954a^4 - 263.9733a^3 + 184.2593a^2 - 65.3190a + 9.0813$$

Now that the values are adjusted to a linear function, we need to shift the values to the intended symmetric point.

$$c = b + 0.1496$$

$$d = x - 0.232$$

Where c is the Intended Voltage, d is the Intended Corresponding Distance, and x is the Distance on the Datasheet.

To shift the symmetric point, we are using the data between 0.232mm and 16.232mm and using it as the data between 0mm and 16mm.

Part 9: Evaluation of Amended Sensor Characteristics

First we created a Time Sweep for the Amended Sensor Characteristic as shown in Figure 15.

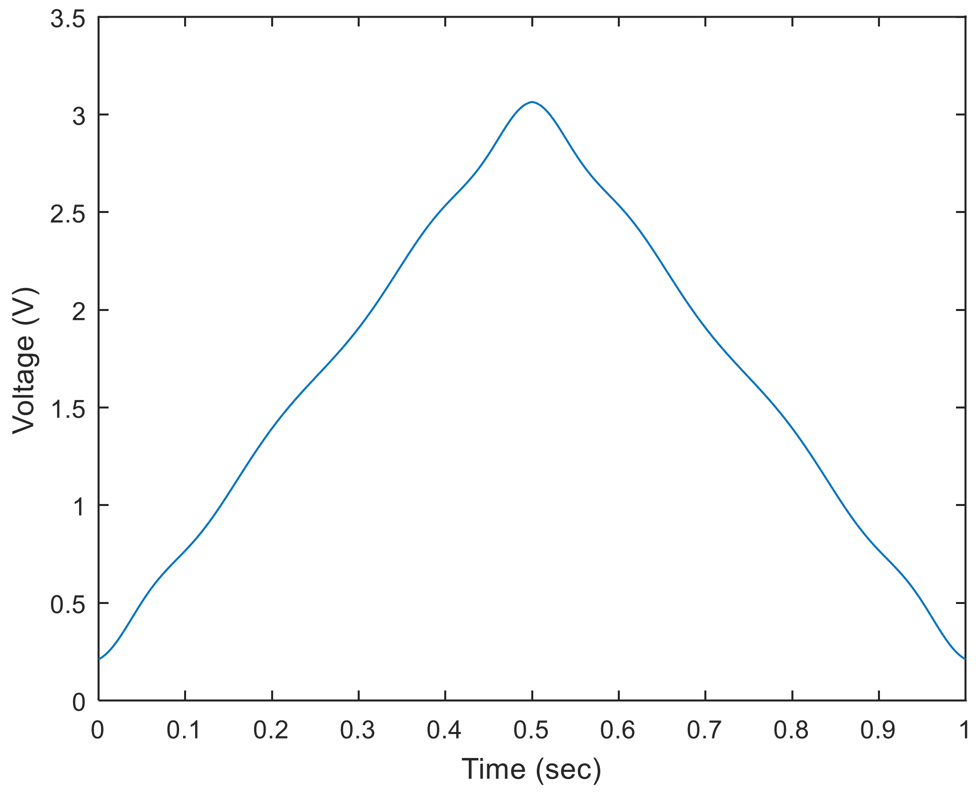

Figure 15: Sensor Output from Time Sweep

Next is to plot the resolution as seen in Figure 16. Notice that the edges have a higher resolution than the rest, but most of the data has a lower resolution than the original data. All data is under the 70 micron requirement.

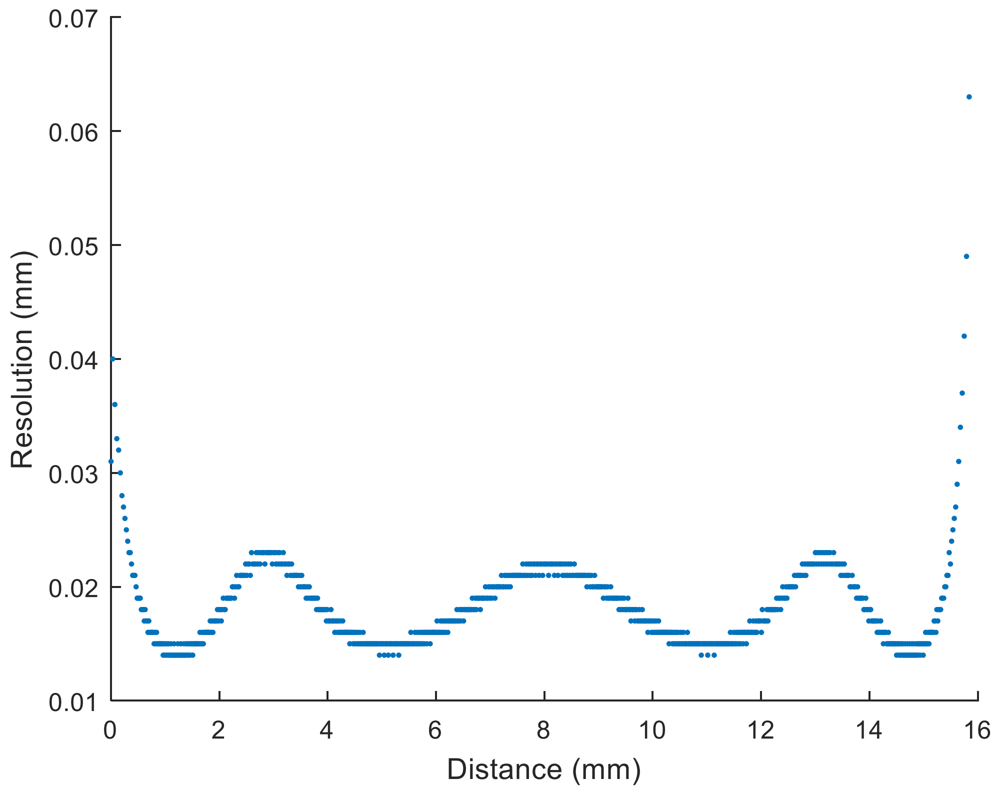

Figure 16: Resolution vs. Distance

The Symmetric point is the same as the one found in Part 3. However, it has been moved to 8mm because of our algorithm.

Resolution

Figure 17 shows the distribution of resolution of the modified sensor values.

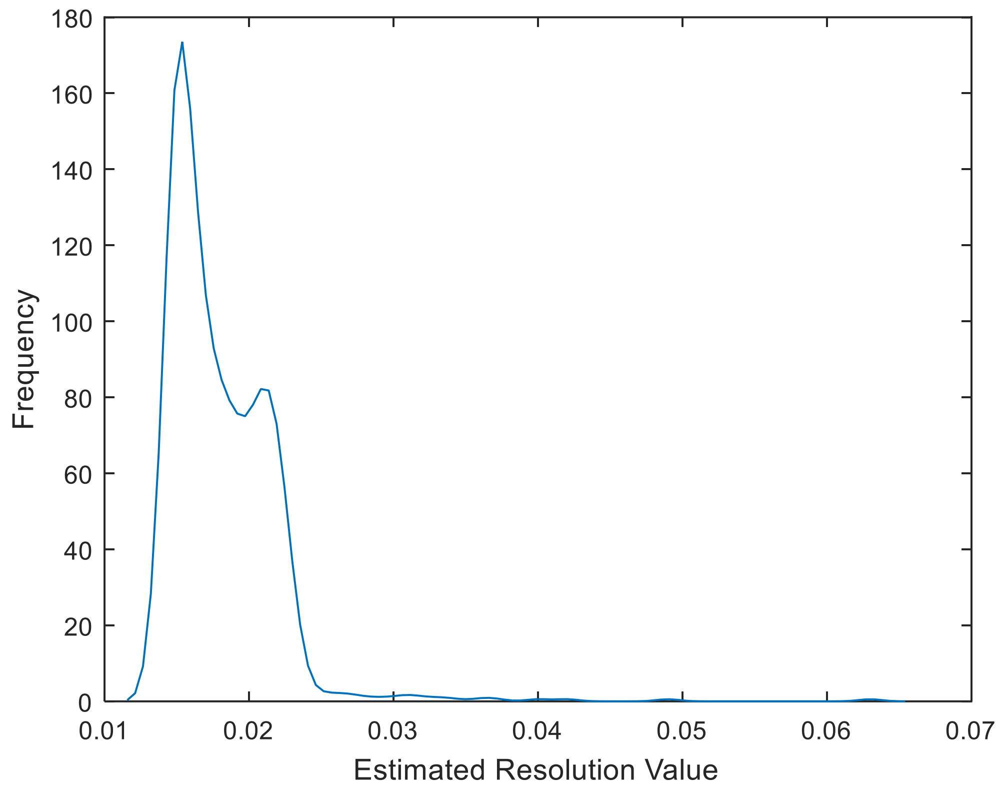

Figure 17: Modified resolution vs. distance

Taking the calculated resolution values, we obtained the following parameters:

a. Mean = 0.017991

b. Median = 0.017000

c. Mode = 0.016000

d. Std_Dev = 0.003809

f. PPM = 0

Our PPM is 0, which gives us greater than 6-sigma!

Error

Figure 18 shows the error of the modified sensor values.

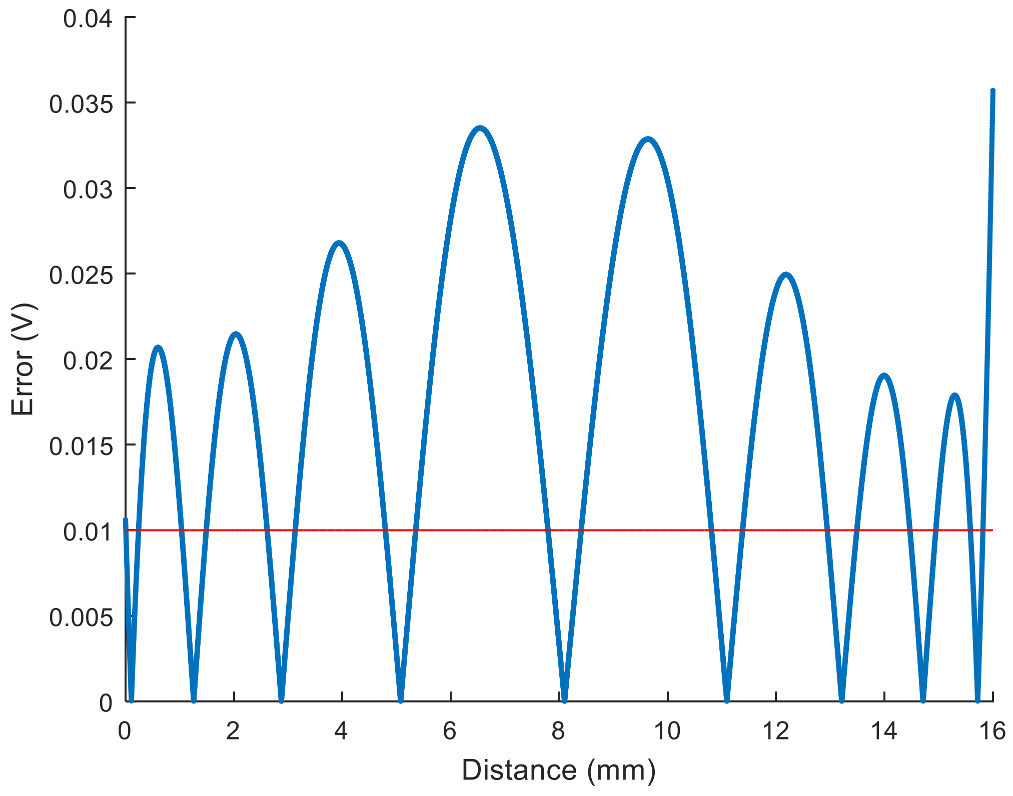

Figure 18: Modified error vs. distance

Figure 19 shows the distribution of the error of the modified sensor values.

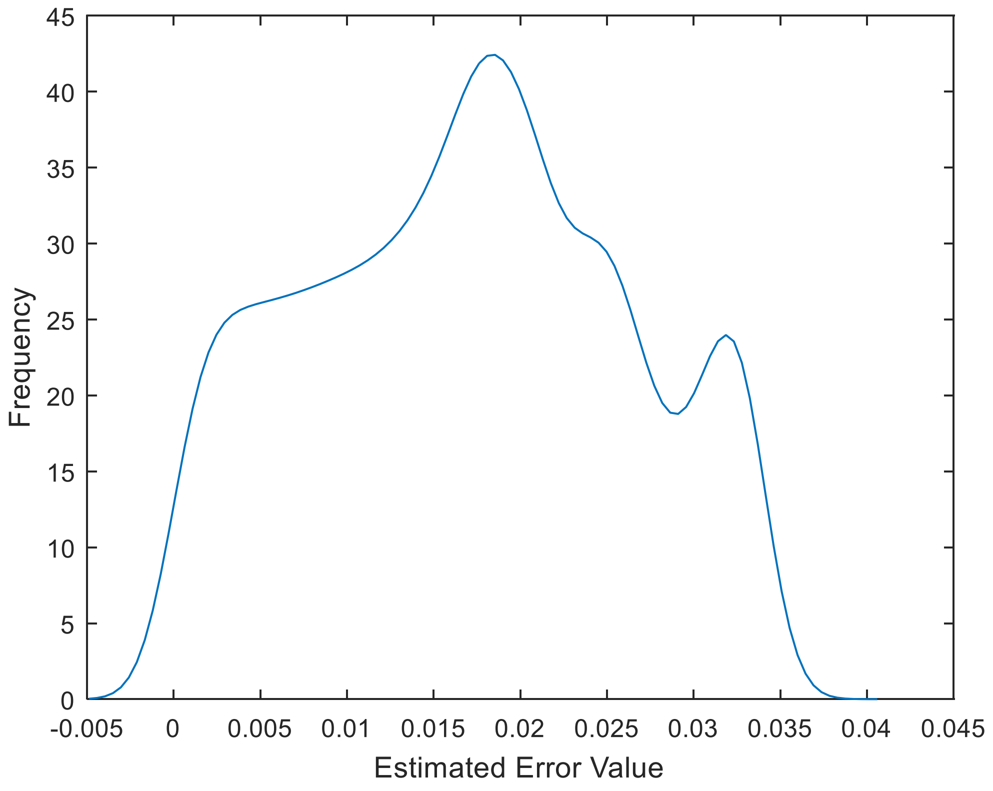

Figure 19: Distribution of Modified Error

Taking the calculated error values, we obtained the following parameters:

a. Mean = 0.016927

b. Median = 0.017330

c. Mode = 0.000004

d. Std_Dev = 0.009184

f. PPM = 737203.924755

Our PPM is 737203.924755, a decrease of about 250,000 PPM. Using an online PPM to sigma table, we found our sigma-level to be 0.9; a great increase from a sigma of 0!

Part 10: Evaluating the Algorithm

The algorithm is simple when using MATLAB. It gives us less resolution and error. We also moved the symmetric point to help with the heuristics of the sensor. However, the signal level for 10mV error is too difficult to get under without adding another polynomial to modify the data, greatly increasing processing time. From the original signal, we have made a significant improvement of almost one sigma-level and have many values that are passing. Resolution now has zero defects.

Code

The MATLAB Code used is provided as a zip attachment along with this report. The code seems scrambled, but the Figure numbers match up with this report.