University of Waterloo

Faculty of

Engineering

Design and Calibration of a Potentiometer

MTE 201

Written By

Abhijith Ramalingam - 20542021

Ayodeji Ige -

Spurrya Jaggi - 20529023

Pruthviraj Atodaria - 20522237

Prepared For

Professor Peter Michael Teertstra

1.0 Design Summary

The purpose of this project was to build, calibrate and analyze statistically a measurement system for either temperature pressure or displacement. This report focuses on the implementation of a measurement system for displacement.

1.1 Measurement System and Theory of Operation

The device comprised of several components including an Infrared sensor, an Arduino Uno, and a laptop. A mechanical setup consisting of a sliding wall, stopper block, base and sensor wall as seen in **figure 1** was used to accommodate the task. The sensor used was a sharp 2d120x infrared proximity sensor which transmits infrared waves from an IR light emitting diode. The transmitted wave is reflected if there's an object within the sensor's range and detected by a light dependent resistor and outputs a sensor value (integer value mapped from 0 - 5v to 0 - 1024 units). The Arduino and computer acted as the signal modification systems which took the raw values of the infrared sensor and converted it to a displacement measurements. The computer was also used as the indicator to display the distance.

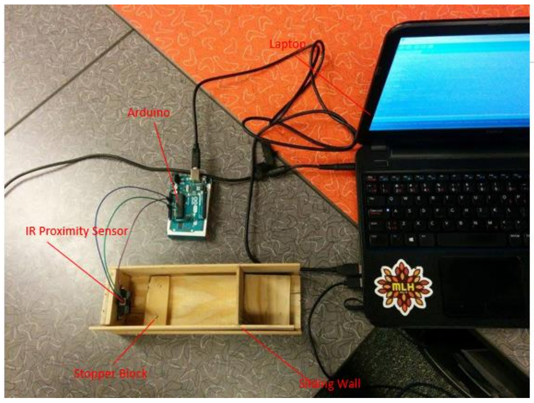


Figure 1 - Labelled Image of Measurement System

In order to measure the dimension of an object, the object was placed on the platform of the device and the slider adjusted so that the object fitted between the slider and stopper block. The infrared sensor measured a value corresponding to the distance between the sensor, which was then read by the Arduino and processed by a script written in Python Programming Language. The program used the converted the sensor value to distance measurement based on function generated during calibration and was displayed on the screen.

In order for the device to function properly, a few assumptions were made. One assumption was that the device must be used in light conditions similar to those when the device was calibrated since the readings may vary depending on whether the lighting conditions had enough IR radiation to cause interference. The device was calibrated in fluorescent light which means that if the device is used in sunlight or incandescent light, it would not return accurate values. Another assumption made was that the output of the infrared sensor is consistent and therefore gives a constant range of value for a particular distance. This is because the program running on the laptop required a specific input value from the sensor to calculate distance accurately. The primary challenge faced was coming up with a decently accurate relationship between sensor values and distance. This was mostly due to the fact that our initial approach which was to fit a curve and obtain the function of the curve using Microsoft Excel was inefficient as a result of an inconsistency between the function Microsoft Excel gives and the behavior of the function. After discovering the issue, a more reliable approach using Python Programming Language was taken and this sped up the calibration process. A secondary challenge faced was with the construction of the mechanical system as it took time to come up with a creative and effective way of implementing the IR sensor to take dimensions.

1.2 Governing equation

Since the infrared sensor used in this device does not have a governing equation, the device does not have a governing equation either.

2.0 Calibration Procedure and Results

2.1 Method, Apparatus and Procedure

A ruler was placed on the surface of the potentiometer to measure the distance between the slider and the stopping block as seen in Figure 2. The slider was moved along the ruler from 0.4cm to 12.0cm with increments of 0.2cm. The corresponding modal value of the sensor was then taken and stored. This process was repeated two more times to obtain a total of three sets of readings. The average of these readings were then taken to reduce random error and then used to build the calibration curve.

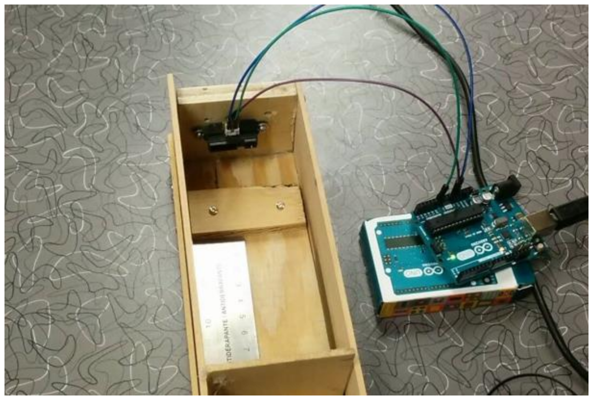


Figure 2 - Ruler placed on the potentiometer to obtain calibration data

2.2 Calibration data and calibration curve

The dataset obtained as seen in Table 1 was used to make a calibration curve.

Voltage Mapped From Arduino									
Voltage	Actual Value (cm)	Voltage	Actual Value(cm)	Voltage	Actual Value (cm)	Voltage	Actual Value(cm)	Voltage	Actual Value (cm)
471	0.4	342	2.8	269	5.2	230	7.6	208	10
464	0.6	336	3	265	5.4	229	7.8	204	10.2
452	0.8	328	3.2	261	5.6	226	8	204	10.4
437	1	320	3.4	257	5.8	225	8.2	200	10.6
426	1.2	312	3.6	253	6	221	8.4	200	10.8
415	1.4	309	3.8	253	6.2	221	8.6	196	11
403	1.6	301	4	249	6.4	217	8.8	196	11.2
391	1.8	297	4.2	246	6.6	217	9	192	11.4
380	2	289	4.4	242	6.8	216	9.2	192	11.6
372	2.2	286	4.6	237	7	212	9.4	188	11.8
361	2.4	282	4.8	238	7.2	212	9.6	188	12
353	2.6	273	5	234	7.4	208	9.8		

Table 1 - Calibration data used to generate the calibration curve.

Using the script file written in Python[1] , a polynomial regression was performed on the dataset to find the best fitting curve as seen in <Insert Figure Num Here>. Multiple degrees of Polynomials were used to fit the data, from a single degree (linear) curve up to six degrees. Ultimately, the curve generated for a polynomial of degree four was found to be most accurate. An exponential curve was also tested on the data set with some success towards higher distance values, however it did not fit the data well for lower distance values. The function generated for the cure was used to make estimates of the distance value based on the sensor readings. The function which was used during the final demonstration was

$$y = (2.137 \times 10^{-9})x^4 - (3.605 \times 10^{-6})x^3 + (2.285 \times 10^{-3})x^2 - 0.6631x^1 + 7.724$$

$$x - Raw\ values\ from\ IR\ Proximity\ Sensor$$

$$y - Predicted\ Distance\ Value$$

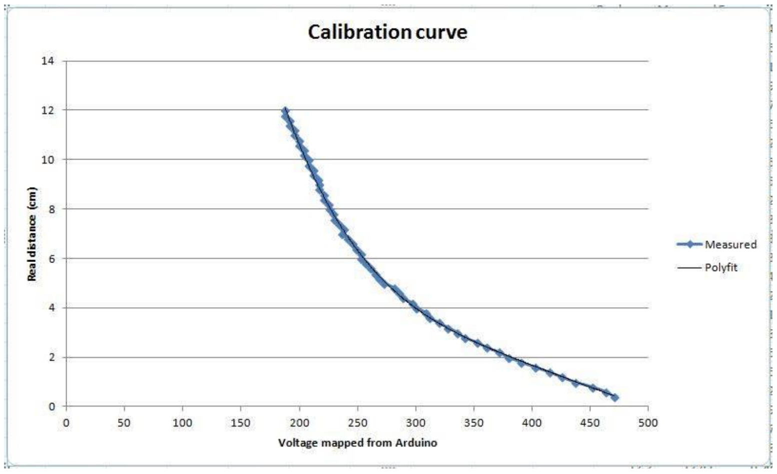
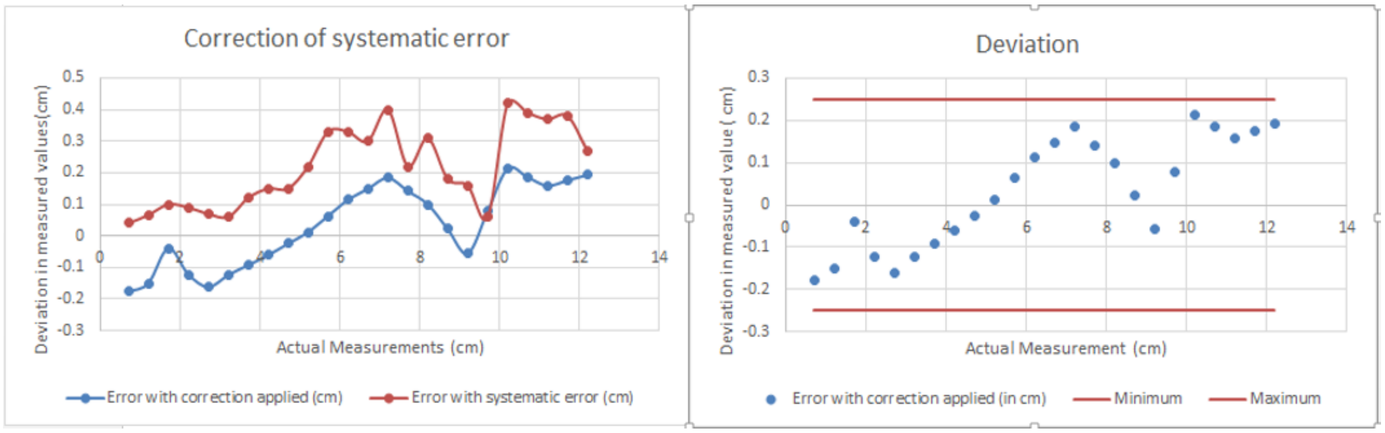


Figure 3 - Calibration curve used to predict distance(cm) using readings from the Arduino.

3.0 Results of the Uncertainty Analysis

3.1 Deviation Plot

After obtaining the calibration curve, the predicted values were tested against the real values using a method similar to that in the above section by placing a ruler on the potentiometer surface and moving the slider by 5 mm increments. A table consisting of the difference in true values and the expected value (error) and the true value was obtained. It was observed that the predicted values were consistently positive. The average error was found to be **+2.162 mm.** Hence a correction constant equal to -**2.162 mm** was applied to all the points which led to improved results as seen in Figure XX.



a) Figure 4 - a) Comparison between the deviation plots before and after the correction was applied , b) Figure 5 - Error (cm) vs Actual value (cm)

The deviation table was then recalculated in a similar way once the systematic error was removed. This was used to generate a Deviation Plot seen in Figure 5.

3.2 Estimate of maximum uncertainty

The deviation plot (seen above in Figure 5) was used to determine that the maximum and minimum errors are **2.1 mm** and **-1.6 mm** respectively. An uncertainty of **± 2.5 mm** was estimated to account any additional error.

4.0 Conclusions and Recommendations for Future Work

Overall this project was a good practical application of concepts taught in the cause and promoted a more appreciation of statistical analysis. Although the system was fairly accurate, it can still be improved in various ways, for example, obtaining more data points to generate the calibration curve with would lead to a more accurate calibration

As suggested by the Teaching Assistants during the demonstration, the sensor should be calibrated under different light conditions like underneath floodlights, bright sunshine or fluorescent light to understand the influence of external light sources on the IR sensor. Since the laboratory conditions were similar to the condition where the device was calibrated, the device was able to return accurate values of measurement.

It is also possible to use the KNN (K- Nearest Neighbors)[2] algorithm which stores all the data points in a database and then returns the closest match to a given value. While this approach would improve accuracy and reduce the complexity of curve fitting, it would take more time to predict a distance value and would require a lot of space to store all the data points.

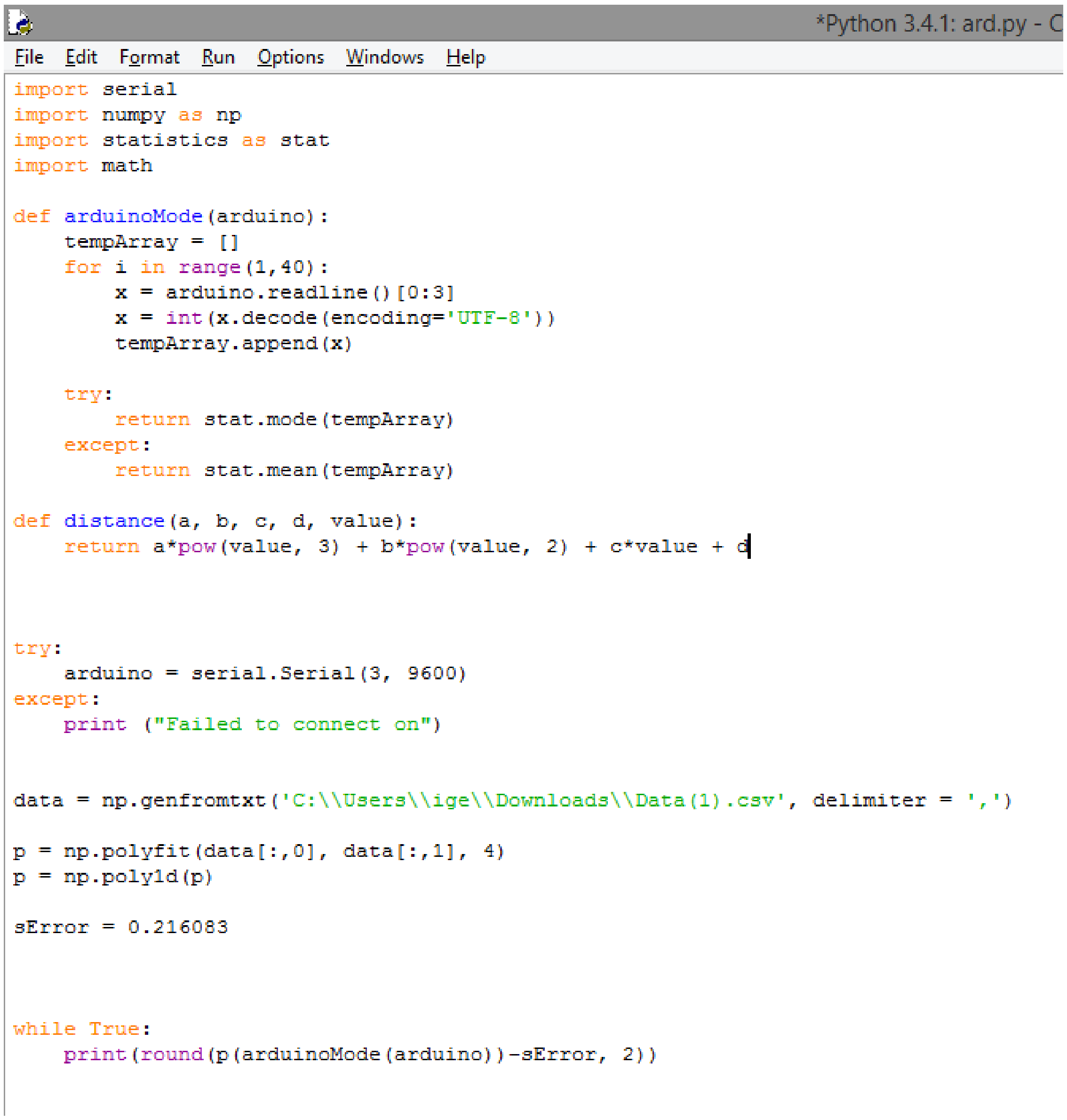
Several improvements can also be made to the mechanical system to

5.0 References

[1] Docs.scipy.org, 'numpy.polyfit — NumPy v1.9 Manual', 2015. [Online]. Available: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html. [Accessed: 20- Mar- 2015].

[2] Scikit-learn.org, '1.4. Nearest Neighbors — scikit-learn 0.15.2 documentation', 2015. [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html. [Accessed: 20- Mar- 2015].

6.0 Appendix


*Python 3.4.1: ard.py - C
File Edit Format Run Options Windows Help

```python
import serial
import numpy as np
import statistics as stat
import math

def arduinoMode(arduino):
    tempArray = []
    for i in range(1,40):
        x = arduino.readline()[0:3]
        x = int(x.decode(encoding='UTF-8'))
        tempArray.append(x)

    try:
        return stat.mode(tempArray)
    except:
        return stat.mean(tempArray)

def distance(a, b, c, d, value):
    return a*pow(value, 3) + b*pow(value, 2) + c*value + d



try:
    arduino = serial.Serial(3, 9600)
except:
    print ("Failed to connect on")


data = np.genfromtxt('C:\\Users\\ige\\Downloads\\Data(1).csv', delimiter = ',')

p = np.polyfit(data[:,0], data[:,1], 4)
p = np.poly1d(p)

sError = 0.216083



while True:
    print(round(p(arduinoMode(arduino))-sError, 2))
```