Smart Sensor Interface: Optimizing and increasing the accuracy of self-calibrating algorithm

Thuta Aung, Robert Dick, Brian Purnomo, Siva Aduri, Kyle May

University of Michigan

Ann Arbor, US

zackchen@umich.edu

Abstract

A smart sensor interface with generalized self-calibration algorithm with DAC as a feedback path has been proposed to support variety of manual sensors for wireless sensor nodes. Three major algorithms: simple linear regression, multiple piecewise linear regression, polynomial regression was tested and analyzed for optimal accuracy, runtime and memory expended for self-calibration process. Additionally, external sources of noises were investigated and considered of while considering the accuracy of the self-calibration algorithms. Combining the novel hardware architecture and standardized multiple piecewise linear regression model, the team was able to achieve error 2 times the 1LSB of a 16bit ADC.

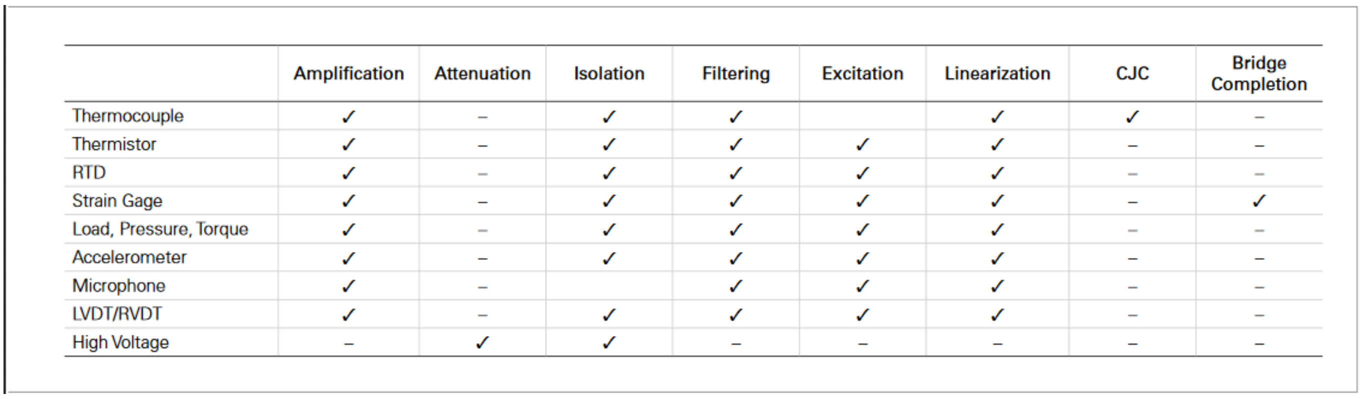
1. Introduction

Increased popularity of internet of things and sensor networks has led to an increased amount diverse sensor being integrated into wireless networks. In order to accommodate different kinds of manual sensors for collecting raw data from the external stimulus, a smart sensor interface that is wireless, cheap and accurate has been proposed.

The fundamental idea of the signal path conditioning lies within the fact that given any analog sensor input into the signal path conditioning, the accuracy of the readings is heavily influenced and distorted by various factors including temperature inconsistencies, imperfection of electrical components, and noise [1].

1.1 Applications

Various manual sensors require different kinds of accommodation within the signal path conditioning. To achieve the best for the measurements, it is crucial that one understands the needs of each type of sensors. The table below shows different unique properties that each sensor application would have, but by considering audio sensing, TDS meter and pH sensor it should cover most of the general case.



	Amplification	Attenuation	Isolation	Filtering	Excitation	Linearization	CJC	Bridge Completion
Thermocouple	✓	–	✓	✓		✓	✓	–
Thermistor	✓	–	✓	✓	✓	✓	–	–
RTD	✓	–	✓	✓	✓	✓	–	–
Strain Gage	✓	–	✓	✓	✓	✓	–	✓
Load, Pressure, Torque	✓	–	✓	✓	✓	✓	–	–
Accelerometer	✓	–	✓	✓	✓	✓	–	–
Microphone	✓	–		✓	✓	✓	–	–
LVDT/RVDT	✓	–	✓	✓	✓	✓	–	–
High Voltage	–	✓	✓	–	–	–	–	–

Figure 1: Summary of different needs from signal path conditioning for different kinds of sensors [2]

Audio Sensing: The acoustic waves measured by human ear ranges from 1Hz to upper limit of 20kHz [3]. In order to ensure that the filtered audio signal is not aliased or under samples, we need to sample the signal at the Nyquist frequency or higher. This means that the signal path would need to have high data rates for audio sensing applications.

pH Sensor: A pH electrode measures the potential difference between the two electrodes to determine the number of hydrogen ions present within the solution which is also known as the pH. On of the electrodes made in glass has a very high impedance typically in the range of 10Mohms to 1000Mohms. Hence, to accommodate for such a sensor, the measuring device needs to be either equally or higher impedance than the glass electrode [4].

1.2 High-Level Diagram

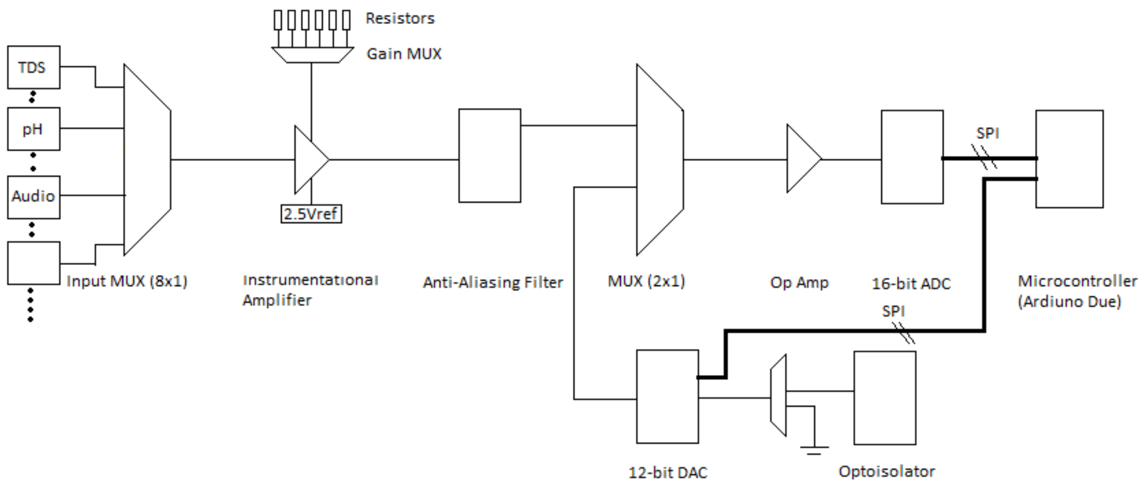


Figure 2: High level diagram of signal path conditioning with input signals (sensors) from the left to the right.

Based on the high-level diagram of a signal path conditioning circuit in the figure above, a DAC feedback loop has been added to the original signal path. The smart interface intends to exploit the advantages of a DAC feedback loop in order to implement a self-calibrating algorithm that improves the accuracy and reliability of the Smart Sensor Interface's conditioned data. In order to resolve this, there will be two stages of calibration.

Short Circuit path calibration: DAC is used to help map the expected representation of m-DAC bits to n-ADC bits by bypassing the preconditioning signal path and directly feeding the DAC output towards the ADC output. Through realizing the relationship between the DAC and the ADC a linear model can be created to accurately map the representation of m-DAC bits to n-ADC bits.

Full path calibration: The aim here is to realize the shifts and noise that could potentially be introduced by the signal path. This is realized by sweeping the m-DAC bits through the entire signal conditioning path to characterize how the expected data bits would have drifted or changed. The only difference in software aspect of the calibration is the use of linear model from the short circuit path to characterize the noise and offsets introduced within the full signal path calibration. There has been previous work on using a separate DAC to calibrate out the inconsistencies and the noise introduced by the signal conditioning path [5], but some of the known works have only explored one of the two techniques:

1) Generating a lookup table to record the voltage to bit representation relationship which requires a large amount of memory [6]

2) Fitting the sensor data into best fit polynomial at the expense of bigger runtime [7].

The proposed idea for the project is to explore the latter option while dynamically readjusting the best fit algorithm and incorporating the first option with the second to compensate for each of the method's weakness.

The solution is comprised of reducing the datasets required for calibration based on the desired accuracy, varying the threshold of RMS to tradeoff between accuracy and speed, and measuring the weight of different external factors that effects the drifts in the offset of the signal path and characterizing the noise accordingly.

The rest of the paper is organized as follows. The closely related work to our design and architecture for the smart sensor interface and some regression models for self-calibration algorithms has been presented in section 2. Our high-level description of our contributions: multiple piecewise linear regression algorithm and usage of either DAC as a feedback path for autocalibration has been described within section 3. The details different stages and components within our signal path conditioning and algorithm has been included within section 4. Following up with a discussion in section 5, conclusion in section 6 and future research direction on different areas of optimizing the self-calibration algorithm in section 7.

2. Related Works

A paper done by Paul T.Kolen for Microcontroller-Based sensor arrays uses a DAC as a feedback path for self-calibration algorithm. However, he uses PGA instead of an instrumentational amplifier; using PGA is more accurate but not as cost effective for commercial purposes such as making sensor nodes. Additionally, his work uses a lookup table of linear regression tailored towards a range of temperature measurements. Hence, was lacking a standardized calibration scheme for variety of sensors.

There is also other calibration algorithm that uses dynamic compensation techniques [8][9] or assigning confidence numbers to each sensor to employ different algorithm schemes. However, all of them does not have a DAC as a feedback path for self-calibration algorithm. In fact, all their work is tailored towards a specific sensor application such as gas and moisture.

To achieve a cheap, and reliable device that could support variety of manual sensors, we came up with a hardware architecture that utilizes DAC and instrumentation amplifier to calibration and dynamic input range adjustment, and employed multiple piecewise linear regression to calibrate our majority of the noise and offsets that could be introduced by external factors and imperfect electronic components within the signal path.

3. System Architecture

Within this section, main parts of the signal path conditioning and the self-calibrating algorithm behind the device has been described.

After considering audio sensor input signals is to accommodate for voltage ranges from -2.5V to 2.5V. The signal path conditioning is comprised of a standard signal path and a DAC feedback path. The standard signal path includes an instrumentational amplifier, anti-aliasing filter, a multiplexer, a buffer op amp, an ADC and a microcontroller. Whereas, the feedback path is made up of a DAC, and an optoisolator as shown in Figure 2.

4.1 Signal Path Conditioning

Instrumentational Amplifier (AD8421ARZ): Purpose of having an instrumentation amplifier is to be able to dynamically adjust gains of the input signals so that the input signals can exploit the full-scale range of the ADC. By using a multiplexer with 8 different values of 1% accurate resistors, it is possible to achieve 2^n gains where n = 1 to 7. Resistor's values for specific gains can be found by using R_G= 9.9kohms/(Gain-1).

As the ADC used only operates at a full scale range from 0V to 5V, we would need a 2.5V offset as our input signal ranges from -2.5V to 2.5V. [10]

Anti-Aliasing Filter: The anti-aliasing filter serves as a function to remove the unwanted frequencies in the higher bands to prevent aliasing when reconstructed. The specific AAF used was LTC-1563, a butterworth low pass filter.

While prototyping, it was observed that the AAF introduces unwanted gains and offsets to the input signal. This was error was then calibrated out using the regression models employed within the self-calibration processes described in the next following section.

Analog to Digital Converter: The ADC used for the Smart Sensor Interface device is 16-bits, MAX11100. The main reason to use this was due to its simple SPI interface and 16-bit resolution. However, its FSR is from 0V to +5V given that the voltage reference is 5V. Ideally, the desired range for the FSR of the ADC would be from -2.5V to +2.5V, but due to this limitation, shifting of 2.5V needs to be applied at different stages of calibration.

Digital to Analog Converter: The DAC used for the Smart Sensor Interface device is 12-bits, AD5721RBRUZ from Analog Devices. The reason AD5721RBRUZ was chosen was because of its price and functionality; the price of the 16-bit DAC was more costly than the resolution it was offering with 16-bits. Additionally, 1 LSB of a 12-bit DAC is 0.00122070312V which is significantly small enough for calibration purposes given that the device's desired analog sensor readings range from -2.5V to 2.5V.

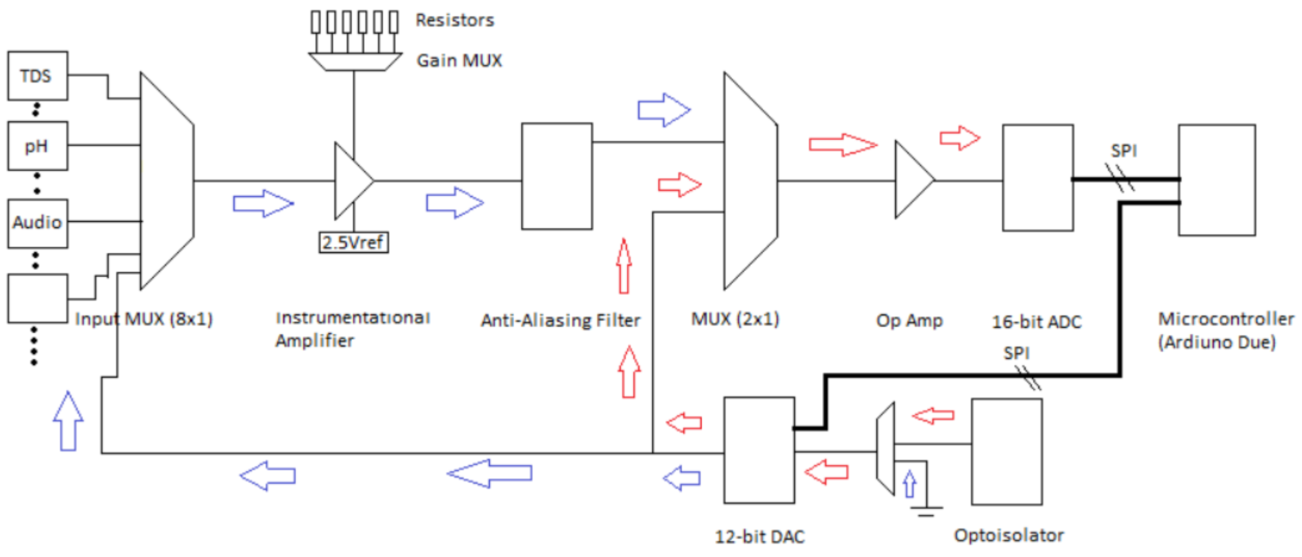
Due to the mismatch of 2.5V between the FSR of ADC and the device's desired range of operation, the DAC whose sole purpose is to calibrate must accommodate for both -2.5V to 2.5V normal operation (full path calibration) and short circuit calibration from 0V to 5V operation. This shifting was achieved through shifting the ground of the DAC using an optoisolator, thus shifting the output range of the DAC from -2.5V to 2.5V to 0V to 5V.

Optoisolator: An optoisolator was required to solely compensate for the ADC's operating range from 0V to 5V. The optoisolator used was SO7741FDBQR and it is attached to 2:1 multiplexer which chooses either normal ground for full path calibration or 2.5V shifted ground from optoisolator for short circuit path calibration.

Additionally, the logical levels of the SPI signals were also shifted by 2.5V accordingly depending on the phase of the calibration.

4.2 Self-Calibrating Algorithm

The major idea in self-calibrating algorithm is to remove and mitigate the offsets and errors introduced by the imperfect components used in the forward signal path. For instance, the gains and offsets introduced by the anti-aliasing filter mentioned in the previous section. This is achieved through short circuit calibration and full path calibration.



Short Circuit path calibration: Short circuit path calibration is to find out the true mappings of the ADC bits in terms of the DAC without going through the full signal path. During this phase, as the ground of the DAC is shifted by 2.5V upwards so that the effective output range of the DAC is from 0V to 5V which would then match the ADC's FSR.

The mappings of the DAC bits to ADC bits are mapped by incrementing 1LSB each for 12 bits of the DAC and then finally applying the simple linear regression model onto the 16-bit ADC readings.

For short circuit path, as the relationship between 12-bit DAC and 16-bit ADC resembles that of a staircase and if infinitely small, can be considered a linear relationship, simple linear regression model will always be used to model the mappings.

Full Circuit Path Calibration: As the full circuit path is comprised of additional components such as the input sensor, the AAF, and the IAs, the full path calibration is going to need a model different from simple linear regression to calibrate out the nonlinearities introduced by the additional components. Hence, for the full circuit path calibration, simple linear regression, multiple piecewise linear regression and polynomial regression models has been analyzed for the team's specific application.

Simple Linear Regression

The simple linear regression involves around y = ax+b where a = Cov(x,y)/Var(x,y) and b is mean(y)-a*mean(x). As this was relatively simple, it was coded in C and ran on a full path data set with gain of 1 and the result is shown as below.

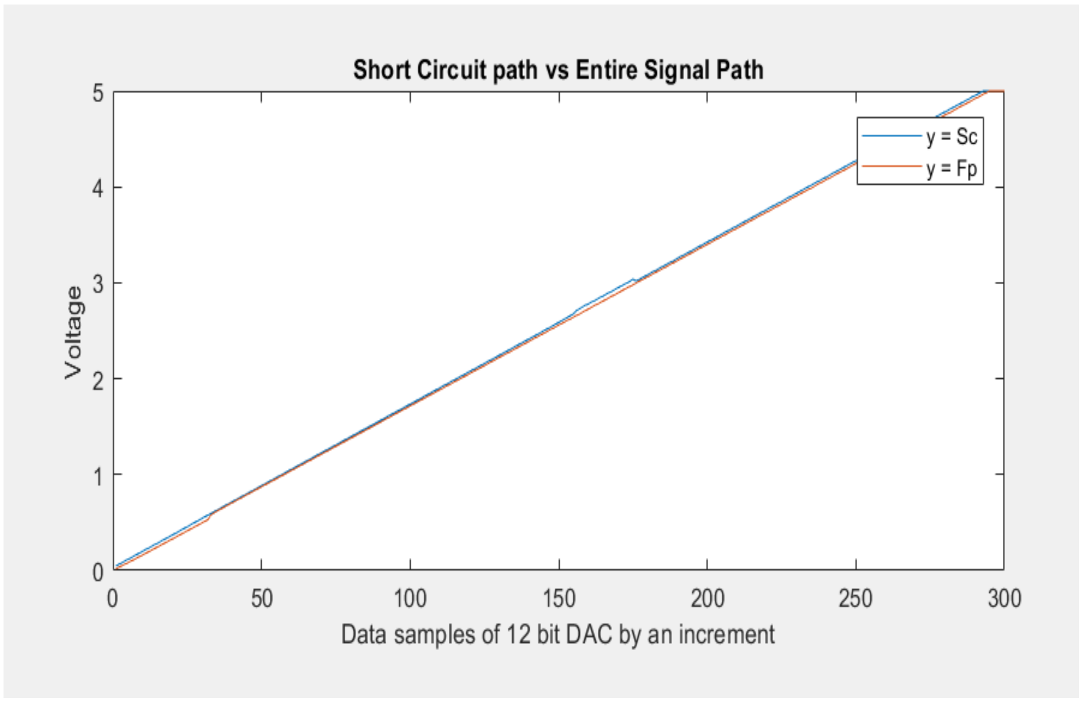


Figure 3:Short Circuit vs Full Path using linear regression

In order to evaluate how close, the full path DAC bits accurately represent that of the true ADC bits reading, the difference for all the sample points were taken and is plotted in the figure below.

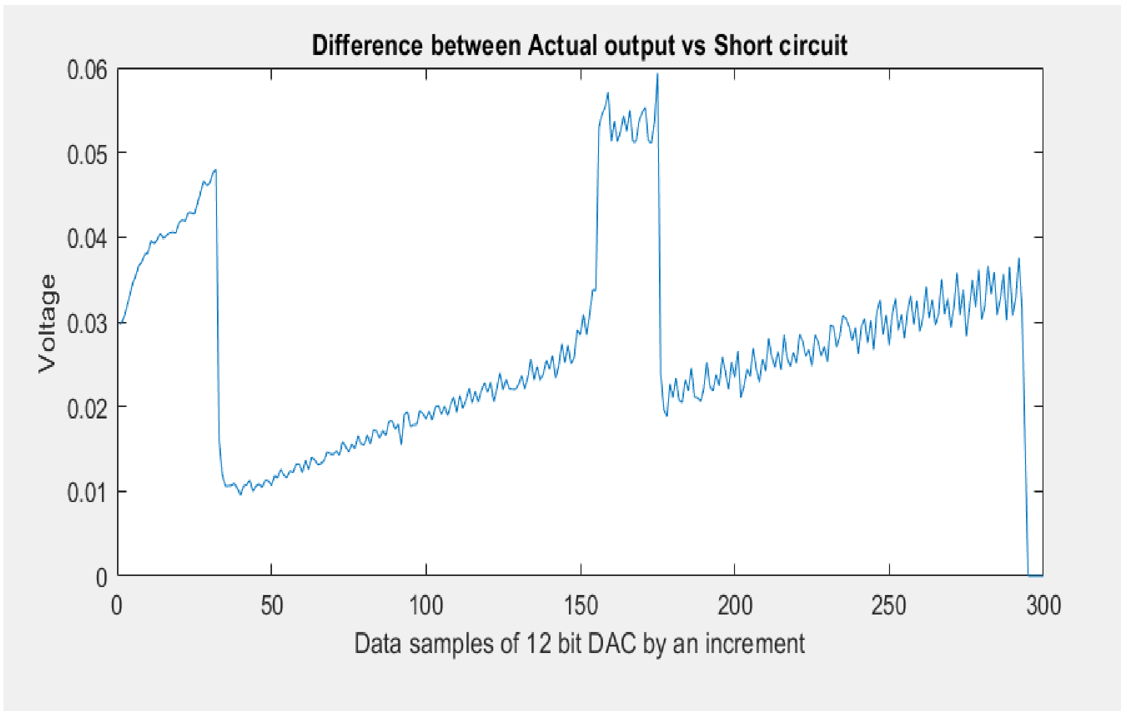


Figure 4:Difference between short circuit and full path calibration using simple linear regression

Based on the previous figure, there are two problems that could be identified. First is the evident of non-linear noise within the 12-bits of DAC range. The sources of the errors would be discussed later in the section about discussion later, but the figure evidently showed that linear regression is not the best suited regression model for the full path calibration. Secondly, the difference resulting is at around 0.02V which is in the order of 10^-2 V. When 0.02V is compared to that of 1LSB of 16bit ADC, the error is 4 times larger than 1LSB of the ADC.

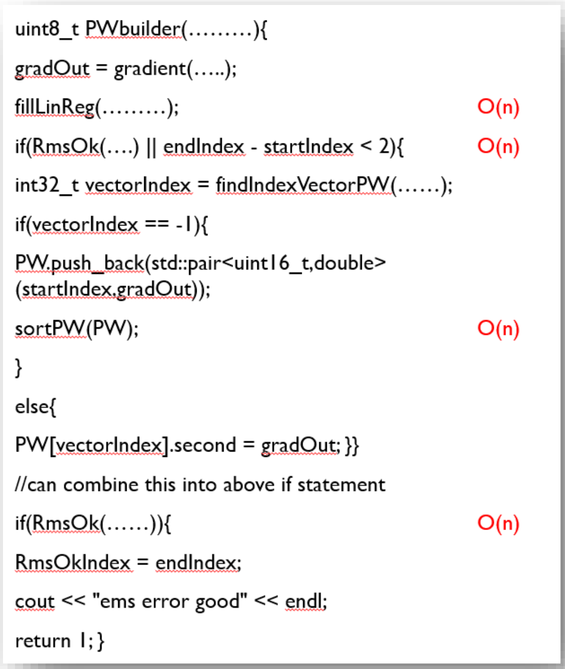
Evaluating the time and the memory complexity of the algorithm, it was found that the average time complexity of the simple linear regression is O(n) with its average memory complexity being O(1). T

Multiple Piecewise Linear Regression

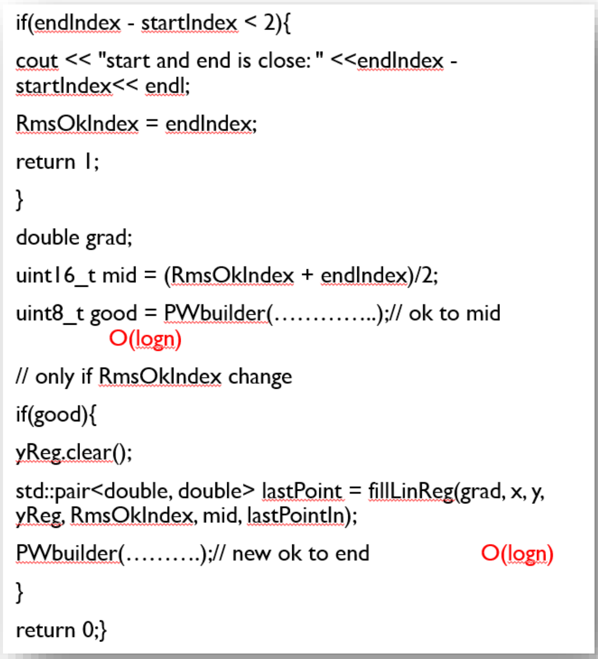
The concept of multiple piecewise linear regression similar to that of a binary search. The only difference is using RMS error threshold as the boundary to decide if more piecewise linear regression is required for the remaining data points.

The general algorithm starts first by finding the mid data point of the n samples. Then find the linear regression between the mid-point and the two different ends. Next, check if the RMS error for the left is linear regression curve is within specified range. If it is, the constants of the linear regression for the left data points are stored in a global look up table. Else, we repeat the algorithm by finding the mid-point of the n/2 data points of the left-hand side.

A simplified version of the algorithm is described in the figures below:



```
uint8_t PWbuilder(.........){
gradOut = gradient(.....);
fillLinReg(.........);                                   O(n)
if(RmsOk(....) || endIndex - startIndex < 2){            O(n)
int32_t vectorIndex = findIndexVectorPW(......);
if(vectorIndex == -1){
PW.push_back(std::pair<uint16_t,double>
(startIndex,gradOut));
sortPW(PW);                                              O(n)
}
else{
PW[vectorIndex].second = gradOut; }}
//can combine this into above if statement
if(RmsOk(......)){                                       O(n)
RmsOkIndex = endIndex;
cout << "ems error good" << endl;
return 1; }
```



```
if(endIndex - startIndex < 2){
cout << "start and end is close: " <<endIndex -
startIndex<< endl;
RmsOkIndex = endIndex;
return 1;
}
double grad;
uint16_t mid = (RmsOkIndex + endIndex)/2;
uint8_t good = PWbuilder(.............);// ok to mid
            O(logn)
// only if RmsOkIndex change
if(good){
yReg.clear();
std::pair<double, double> lastPoint = fillLinReg(grad, x, y,
yReg, RmsOkIndex, mid, lastPointIn);
PWbuilder(..........);// new ok to end              O(logn)
}
return 0;}
```

From the figure above, the PWbuilder is the name of the function, and returns either a logical value of either 1 or 0 if midpoint could be found. RmsOk is the function that finds the RMS error of all the data points compared to the linear regression curve found from the function gradient().

The MPL regression with RMS threshold of 0.05 has been performed on the same data set as the simple linear regression as above and is shown in the figure below. Note that the non-blue circles are the additional piecewise linearly regressive segments that was found due to RMS threshold exceeding 0.05 if combined with the closest blue linear curve.

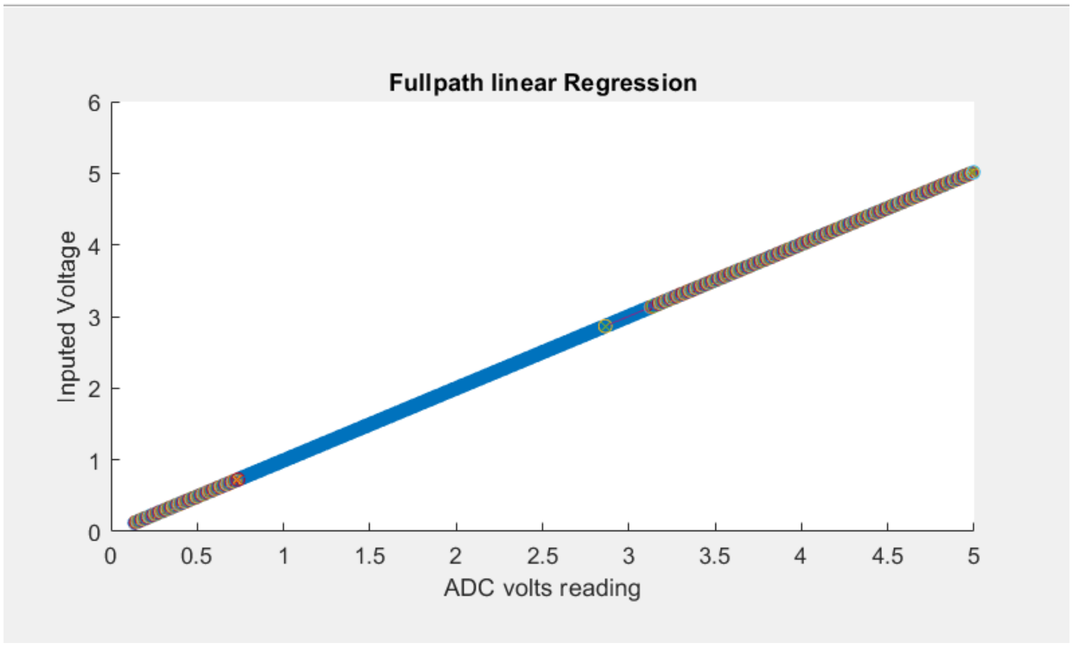


Figure 5:Full Path Calibration using Multiple Piecewise Linear Regression with RMS error of 0.05

If the difference were to be found between the full path and the short circuit path, the error lies within 0.0002V which lies in the order of 10^-4 V for error. The difference between the two paths is shown in the figure below.

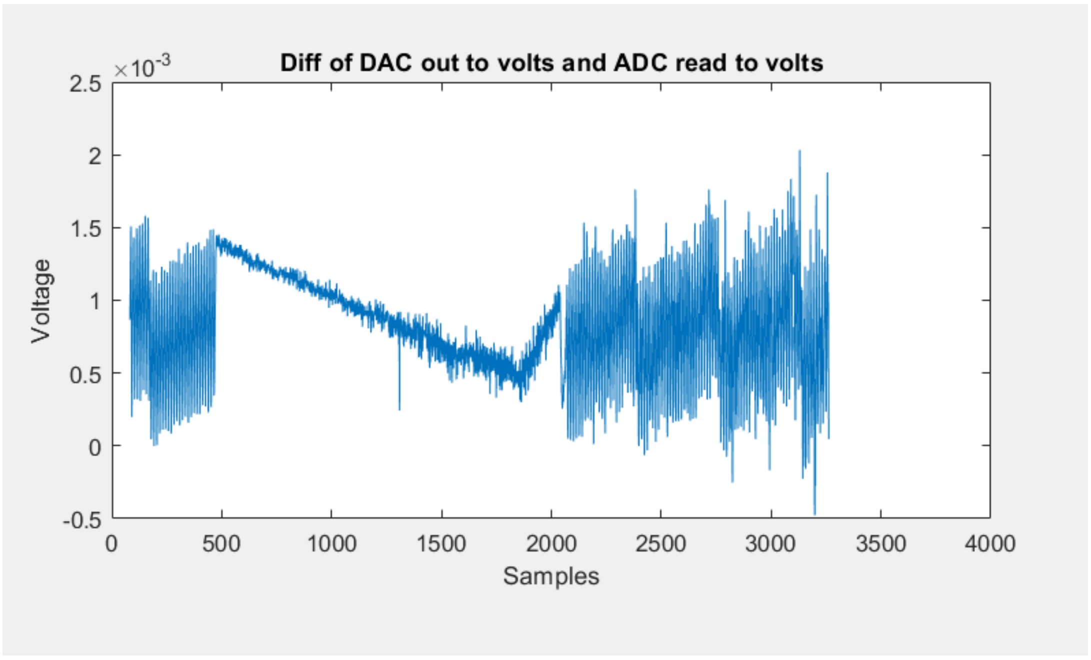


Figure 6: Difference between short circuit path from Figure 3 and full path calibration

Time complexity: As this RmsOk() function iterates through all the data points that falls within the range of the linearly regressive curve, it has runtime of O(n). Similarly, other functions within the main function PWbuilder like sortPW() and fillLinReg() also takes on an average time complexity of O(n). As these functions are within the recursive binary search algorithm (PWbuilder()), the worst case runtime complexity is found to be O(nlogn). At the best case, the runtime complexity is found to be O(n).

Memory Complexity: The memory occupied by running MPL is found to be O(1) as the MPL itself is tail recursive.

Polynomial Regression

The polynomial regression models the curve in order of magnitudes greater than or equal to n = 2 where y = a + $bx^1 + cx^2 + \ldots nx^n$. As many of the python libraries already have existing polynomial regression methods, the same dataset for previous regression models were used with the polyfit() function of the numpy library from the default python library. With the order of magnitude n = 2 for the polynomial regression, the full path calibration is portrayed in the figure below. The equation was found to be:
y = -4.61724725e-05 +(1.00669251) *x -(1.54181348e-02) *x.^2;

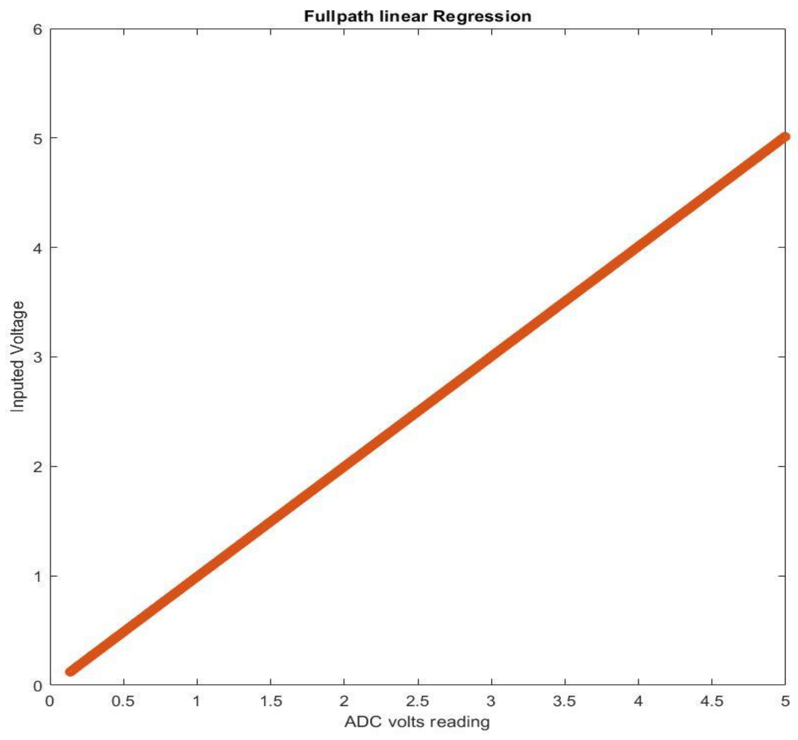


Figure 7:Full Path Calibration using Polynomial Regression of power 2

The difference between short circuit path and full circuit path is found and is plotted in the figure below.

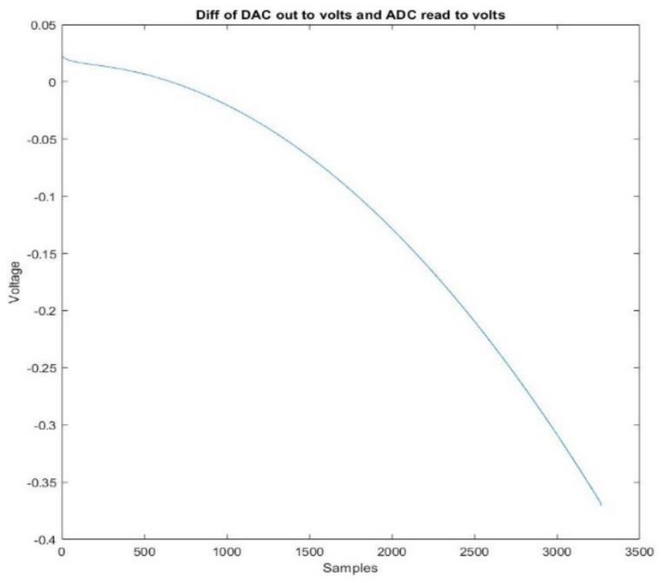


Figure 8: Difference between short circuit path from Figure 3 and full path calibration

Based on the figure above, it is evident that the error increases as the number of DAC bits increases. Moreover, the error is within the range of 0.4V which is in the order of 10^-1 V. This is worse than that of the linear regression model as it is 5 times larger than 1LSB of the 16-bit ADC.

Time Complexity: Using the python libraries such as sklearn DBSCAN, scipy, HDBSCAN, it is found that the average runtime complexity is O(n^2). [11]

Memory Complexity: The memory complexity is found to be O(n*(n-1)/2) which is approximately equal to O(n^2).

5. Discussion

While it is evident that neither simple linear regression nor the polynomial regression is suitable for the full path calibration due to their inaccuracies being 4-5 times greater than that of 1 LSB of 16 bit ADC, there are also other advantages that are offered by MPL that could not be achieved from the both the simple linear regression and polynomial regression model. These include the flexibility of changing the RMS threshold of the MPL algorithm to adjust the tradeoff between the speed and the accuracy of the algorithm. For applications like that of audio sensing, one might want to use a lower RMS error threshold as the data rate is higher than most of the sensors. Hence, having a faster algorithm would take higher priority than having a near perfect accurate algorithm.

On the other hand, although the MPL regression model solves most of the non-linearities introduced by the imperfect components of the system, and calibrates out the offsets, frequency dependent noises are still of a concern within our self-calibration algorithm. As these tend to require more computationally heavy like Fourier transforms, it might perhaps be even a possibility to leave the error as it is considering the amount of energy required to sacrifice to gain a small bit amount of accuracy in data.

Another concern are the sources of the noise that could be evidently seen in that of the difference short circuit and full path calibration of the simple linear regression graph.

The noise observed within the full signal path can be broken down into several possible factors:

1) Frequency dependent gains and offsets introduced (largely by AAF) and other electrical components
2) External factors such as the thermal
3) Internal factors such as burst noise, flicker noise
4) Protoboard
    i) EMI from serial communication signals
    ii) Breadboard

We believe that the biggest contribution to the noise we are observing is coming from the protoboards. We were able to safely reduce the EMI interference from the SPI serial communication between the Arduino Due to our ADC and DAC using copper shield.

However, it is also possible that without the imperfections of the breadboard, the external and frequency dependent gains still negatively impact the accuracy of our signal path. Internal noise such as burst and flicker noise on the other hand exists in most analog circuit due to imperfection of electrical components such as op amps and diodes, but it is less punishing compared to the former. Thus, will be explored on later if needed.

Some work has been done for thermal dependency, and it turned out that most of the major component's tolerable errors for extreme ranges of temperatures were lower than that of 1LSB. Hence, the team looked into the effect of temperature onto passive components such as the resistors used for gains within the instrumentational amplifiers. It was then found out that using 5% accurate resistors will surely cause inaccurate gains as temperature fluctuates by every 20^0F. However, to resolve this, one can use 1% accurate resistors for selecting accurate gains for the instrumentational amplifier.

6. Conclusion

Through the paper, a generalized system that supports a variety of sensors has been established on both hardware and software aspects. Using DAC as a feedback path for self-calibration and using MPL regression model allows us to accommodate various kinds of sensors within our interface with difference of errors between short circuit and full path calibration with around 2 times greater than 1LSB of a 16-bit ADC.

On the other hand, usage of Instrumentational amplifier instead of PGAs also allowed us to reduce the costs for commercial purposes and reduce the overall complexity of the system.

7. Future Research Direction

1. Testing out the circuitry on the actual PCB and an application

2. Characterizing other sources of noises by simulating extreme environmental conditions like humidity as the sensor nodes tend to operate in harsh conditions

3. Refining multiple piecewise linear regression to reduce the RMS error to smaller than or equal to that of 1LSB of a 16bit ADC.

4. Finding a better suited ADC that operates from -2.5V to 2.5V so that additional components such as the optoisolator can be removed and thereby reducing the overall complexity of the system

8. References

[1] P. T. Kolen, "Self-calibration/compensation technique for microcontroller-based sensor arrays," in *IEEE Transactions on Instrumentation and Measurement*, vol. 43, no. 4, pp. 620-623, Aug. 1994. doi: 10.1109/19.310177

[2] Ashlock, D. and Warren, A. (n.d.). *The Engineer's Guide to Signal Conditioning*.

[3] Anon, (2019). [online] Available at: https://www.electronics-tutorials.ws/io/io_8.html [Accessed 24 Apr. 2019].

[4] AN-1852 Designing With pH Electrodes. (2019). Texas Instruments, p.2.

[5] P. T. Kolen, "Self-calibration/compensation technique for microcontroller-based sensor arrays," in

IEEE Transactions on Instrumentation and Measurement, vol. 43, no. 4, pp. 620-623, Aug. 1994.

[6] P. T. Kolen, "Self-calibration/compensation technique for microcontroller-based sensor arrays," in

IEEE Transactions on Instrumentation and Measurement, vol. 43, no. 4, pp. 620-623, Aug. 1994.

[7] J. M. Dias Pereira, O. Postolache and P. M. B. Silva Girao, "A Digitally Programmable A/D Converter

for Smart Sensors Applications," in IEEE Transactions on Instrumentation and Measurement, vol. 56, no.

1, pp. 158-163, Feb. 2007.

[8] Jun Cao, Jiawei Zhang, Yixing Liu, "A Novel Dynamic Compensated Interface for Lumber Moisture Content Sensor", *Intelligent Control and Automation 2006. WCICA 2006. The Sixth World Congress on*, vol. 2, pp. 5267-5271, 2006.

[9] A. Voulgaris, Th. Laopoulos, "An intelligent microcontroller-based configuration for sensor validation and error compensation", *Instrumentation and Measurement Technology Conference 2003. IMTC '03. Proceedings of the 20th IEEE*, vol. 2, pp. 973-976 vol.2, 2003.

[10] https://www.analog.com/media/en/technical-documentation/data-sheets/AD8421.pdf

[11] “Benchmarking Performance and Scaling of Python Clustering Algorithms.” *Benchmarking Performance and Scaling of Python Clustering Algorithms - Hdbscan 0.8.1 Documentation*, hdbscan.readthedocs.io/en/latest/performance_and_scalability.html