Design Report:

Structural Health Monitoring System

Edge Impulse Deployed to Edge Hardware for Real-World Impact

Shelby Racca
Christopher Ferdin
Christopher Houser

11-30-2025

Executive Summary

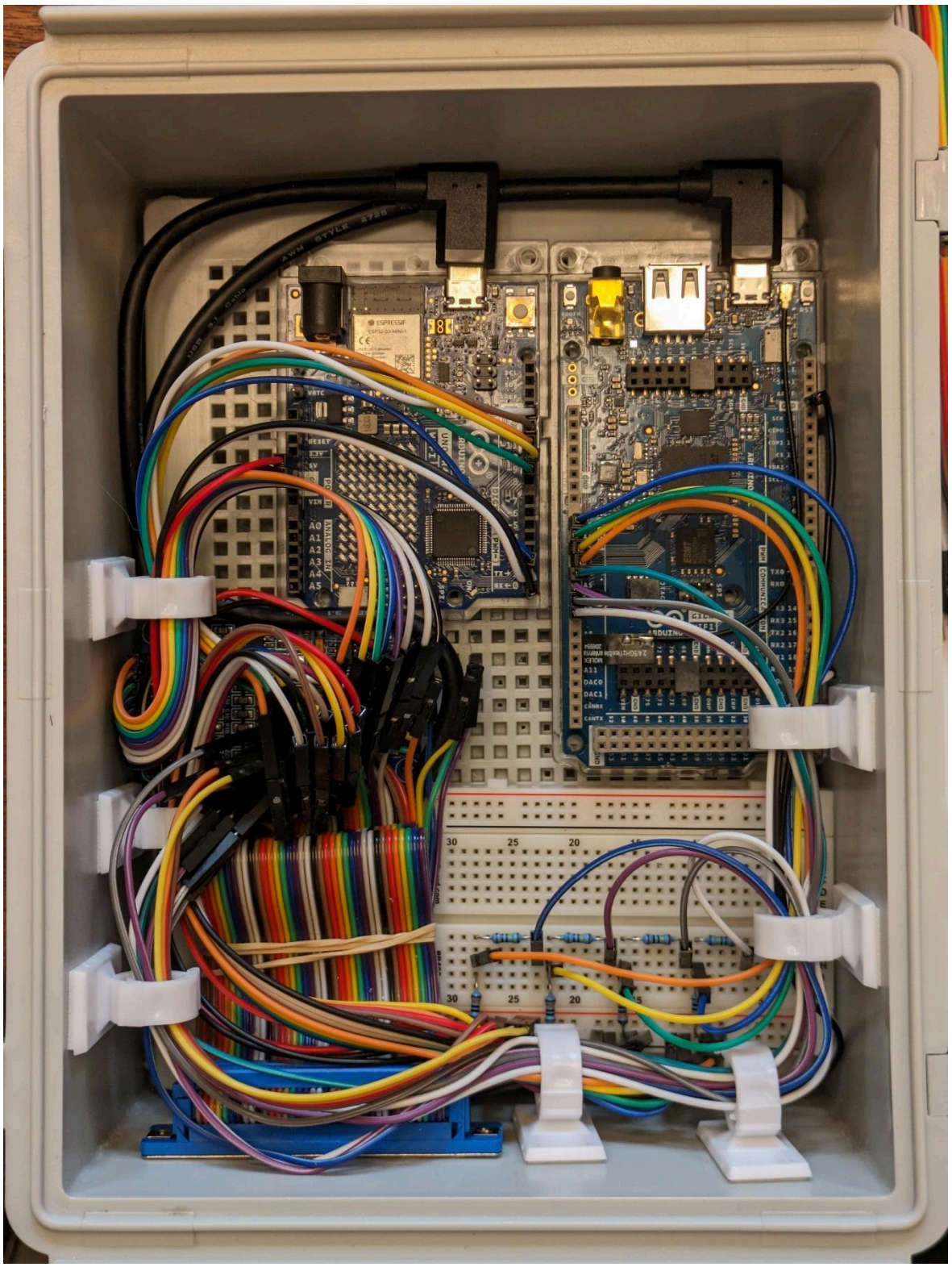
This study aims to explore structural health monitoring (SHM) systems by leveraging signal processing techniques with low-cost hardware. The primary objective is to develop a cost-effective solution that integrates with a mobile app, providing real-time feedback on structural integrity for residential, commercial, or industrial infrastructure to facilitate proactive maintenance practices and improve infrastructure safety.

Our design uses a microcontroller equipped with analog-to-digital converters, dual processors, and Wi-Fi support, enabling real-time data acquisition and preprocessing through a dedicated real-time operating system. Leveraging the Arduino platform and wireless communication protocols, the hardware allows future scalability by maintaining modularity.

This SHM system incorporates signal processing techniques focusing on the extraction of Fourier coefficients, power spectral density, short-time Fourier transform, wavelet transform, and spectrogram values to distinguish stable and active signals. The study uses simulated environmental disruptions for the purposes of acquiring data and determining the existence of structural abnormalities.

Areas of further research include incorporating a machine learning model to be trained iteratively with experimental data. The iterative nature of data collection ensures continual refinement of the machine learning model during development through the incorporation of additional experimental data and exploring new methodologies. This systematic approach ensures the improvement of the system and continued adaptability.

Prototype Circuit Hub



1. Project Description

In the pursuit of advancing structural health monitoring systems, our project focuses on the development of a cost-effective solution with a specific emphasis on signal processing. The aim is to create a compact, efficient device that can connect to a mobile app and provide real-time feedback on structural integrity. Our effort is rooted in contributing valuable research to the realm of edge intelligent devices for structural health monitoring.

At the core of our design is the carefully selected microcontroller, a pivotal component offering analog-to-digital converters, dual processors, and Wi-Fi support on a unified board. This integration facilitates real-time data acquisition and preprocessing through a dedicated real-time operating system. Leveraging the Arduino platform and TCP/IP stack libraries, the microcontroller ensures direct connectivity with a mobile app over Wi-Fi. Importantly, our system remains modular, prepared for future evolution in the absence of a fixed back-end structure.

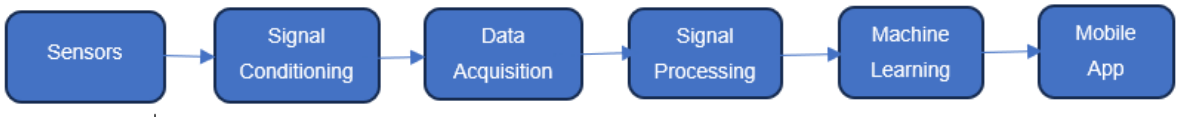
The project's local data storage relies on a secure SD card, enabling two-way communication using HTTP based long polling while maintaining a user-friendly interface through intuitive graphical elements.

Our signal processing approach involves a machine learning model trained and refined iteratively with experimental data. The goal is to employ CUDA (Compute Unified Device Architecture) while training the model to experiment with diverse models and data inputs, with a specific focus on extracting Fourier coefficients, power spectral density, and spectrogram values. These features should demonstrate damage and its location on the structure using a machine learning model.

The project's iterative nature ensures continual refinement of the machine learning model, incorporating additional experimental data, and exploring new

methodologies. This forward-looking approach ensures the adaptability and relevance of our system in the dynamic field of structural health monitoring.

2. Final Design Specifications and Ethical Considerations



Sensors in full-scale structures may be disjointed in such a way that smart sensors could be preferred. For a small-scale model like the one used in the project, the sensors are directly connected to a voltage divider and the voltage divider to the Arduino Giga. The microcontroller possesses a Cortex-M4 processor running at up to 240 MHz and a Cortex-M7 processor running at 240 MHz. There is 1 MB of RAM onboard the processor and an additional 8 MB of RAM available on the board as SDRAM. This shared memory space is used to allocate and transfer data between the processors. This Arduino system also contains 14 12-bit analog-to-digital converters which are necessary for reading voltage levels at sufficient resolution.

Because of the nature of polling intervals and the validity of sensor data based on these intervals, a real-time operating system is preferred. FreeRTOS is the real-time operating system selected in this project. The Arduino Giga R1 Wi-Fi contains two microprocessors which can simultaneously allow the use of a real-time operating system in conjunction with a multitasking operating system running on the same board. This device also provides Wi-Fi connectivity and Bluetooth connectivity, which are required for wireless connection in modern mobile devices. When the processor time available is sufficient, signal processing is handled by the processor running the real-time operating system.

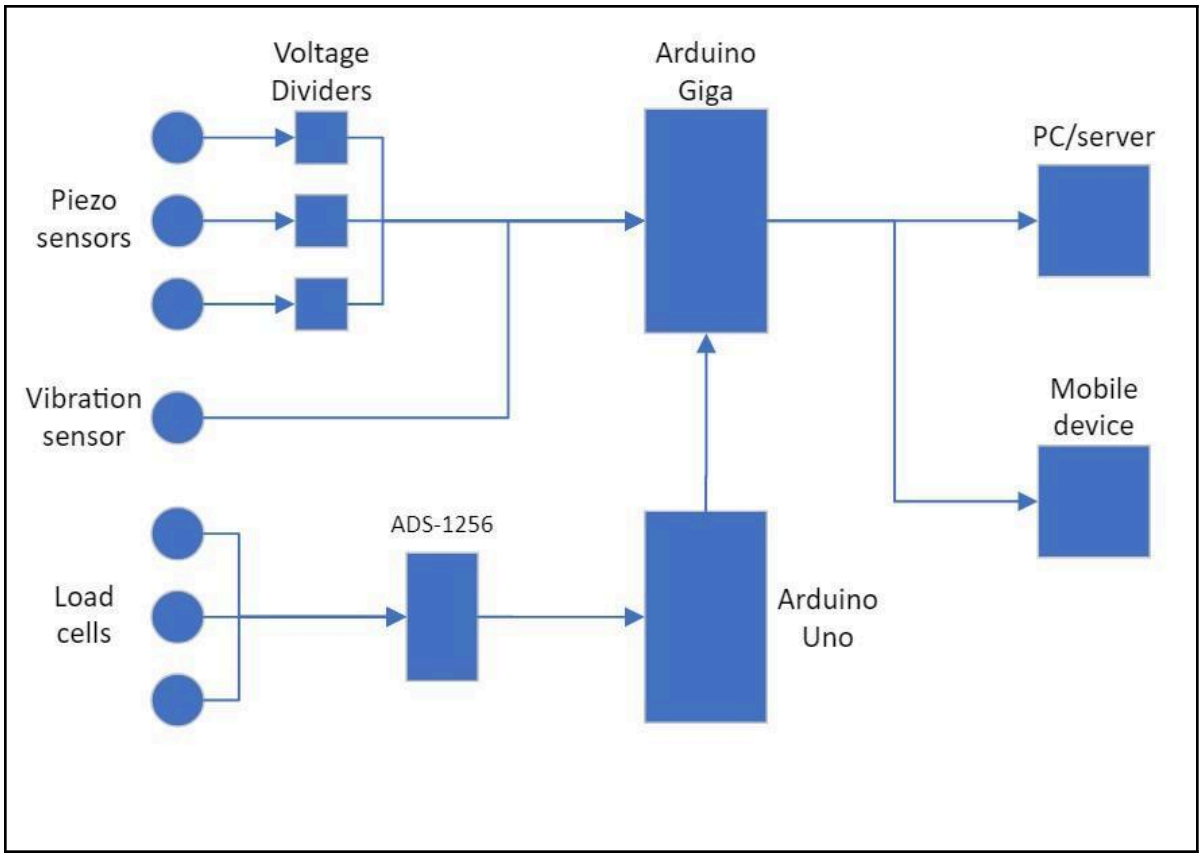
The processed data is accessed by the second microprocessor, a more powerful processor which runs the machine learning algorithm and networking functions. This algorithm is trained using the scale model with equipped sensors to determine criteria for sending a signal to the mobile app, a future development.

For the mobile app stretch goal, Android is used due to its available libraries, affordability, and ability to develop and load software directly. Wi-Fi is the protocol which is used as Internet capability is available with the use of a VPN.

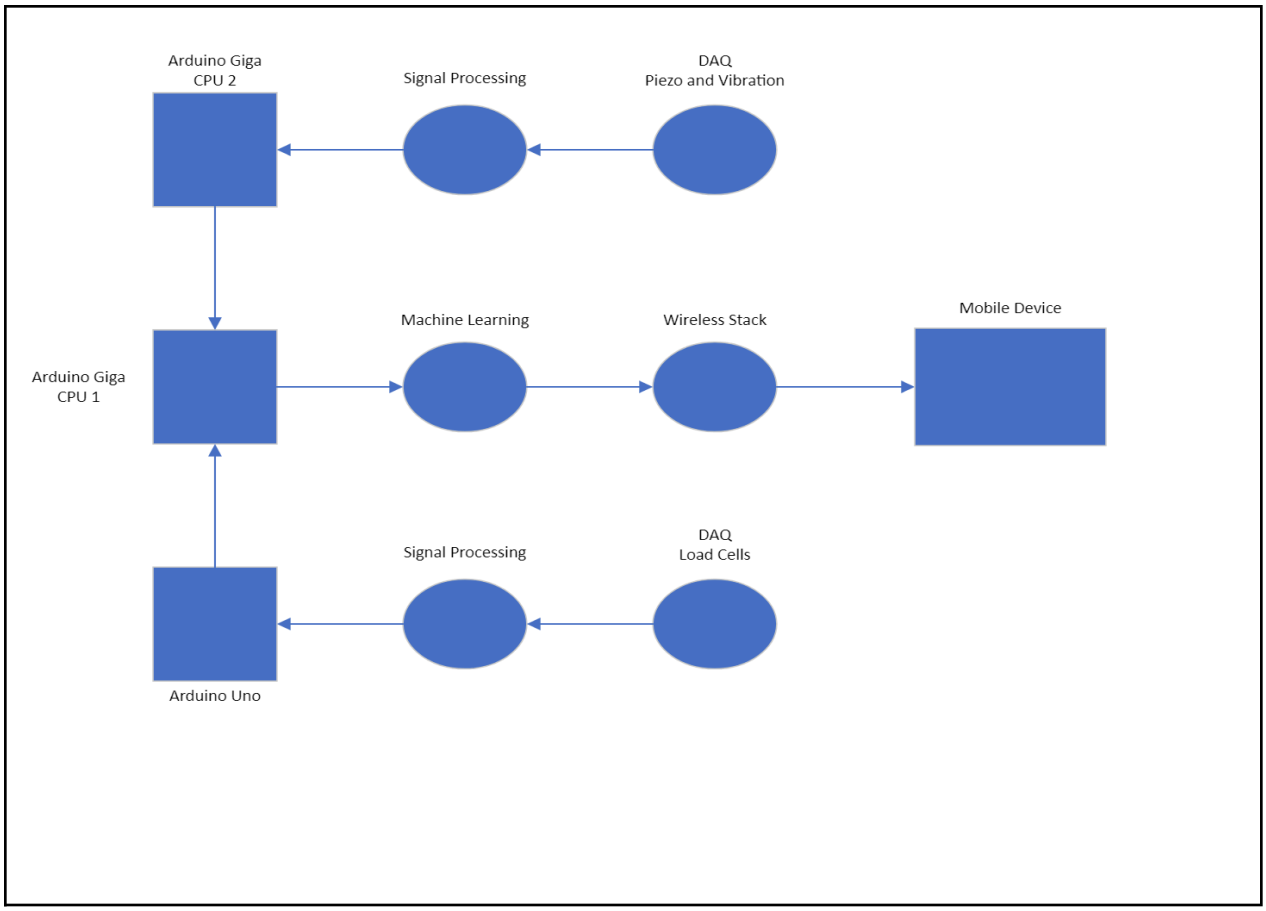
3. Design Solution

3.1 Product Architecture

Hardware Architecture



Software Architecture



3.2 Hardware Subsystems

1) Piezoelectric Sensors with voltage divider
2) Load Cell Sensors with ADS-1256 to Arduino

3.2.1 Voltage Divider and Piezoelectric Sensors

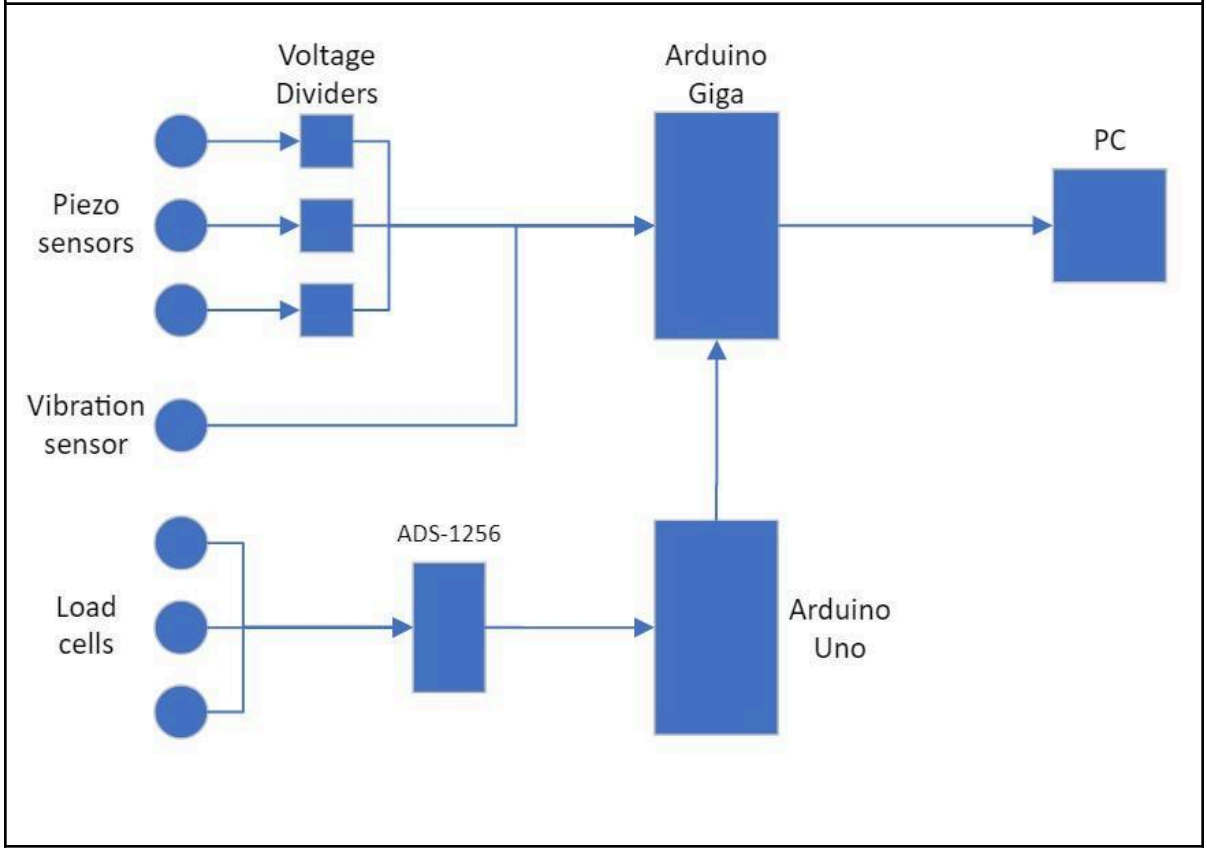
These sensors detect vibrations and stresses within the structure. They are strategically placed to capture data that is indicative of structural integrity.

3.2.2 ADS-1256 and Digital Load Cell Weight Sensors

These sensors measure the load and weight distribution across the structure, providing critical data for assessing potential points of failure.

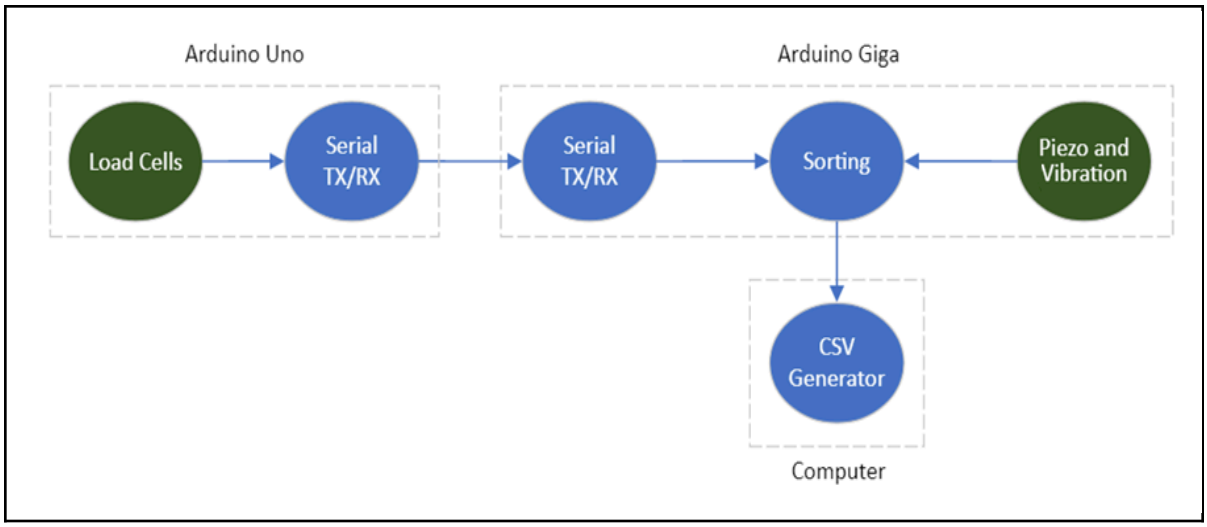
3.2.3 Arduino as Data Acquisition System Framework

DAQ Hardware Framework



3.3 Software Architecture

DAQ Software Framework



3.4 Software Modules

3.4.1 Data Acquisition Module

Arduino libraries for Communication between Arduino Giga CPUs
o RPC.begin() o RPC.print() o RPC.read()
Arduino libraries for Communication between Arduinos
o Serial.write() o Serial.read()

3.4.2 Data Processing Module

Implements digital signal processing, including filtering and data cleansing.

3.5 Hardware Off-The-Shelf Items

Item	Description

400mm Threaded Rods and Studs Assortment	Assortment of stainless-steel threaded rods and studs for mechanical connecting.
Digital Load Cell Weight Sensor (1KG)	1-Kg weight sensor
Digital Load Cell Weight Sensor (5KG)	5-Kg weight sensor
ADS1256 24 bit 8 Channel ADC AD Module	24-bit ADC module with SPI interface, supporting up to 30ksps data output rates.
LM358 Operational Amplifier Module	Set of LM358 operational amplifiers, suitable for a wide range of power supplies.
Flexible Piezoelectric Film Sensor High Sensitive Polymer PVDF	Piezoelectric film sensor
Metric Bolt Assortment M2 M3 M4, 21 Sizes 4MM to 20MM	Bolt assortment kit including screws, nuts, and washers.
55PCS M2 M2.5 M3 Threaded Insert Assortment Kit	Assortment of threaded inserts for creating durable threaded holes.
Banana Plugs	Gold plated banana plugs, open screw type connector
Arduino Giga R1	dual-core STM32H747XI microprocessor with a Cortex-M7 and a Cortex-M4 core, 76 digital I/O pins, and 12 analog input pins.
Arduino Uno	The Arduino Uno microcontroller board
NI USB 6211	A multifunction DAQ device that offers analog I/O, digital input, digital output, and two 32-bit counters
PLA filament	3D printing material
TPU Filament	3D printing material
16GB SDHC Flash Memory Card	Memory card for Arduino DAQ processing

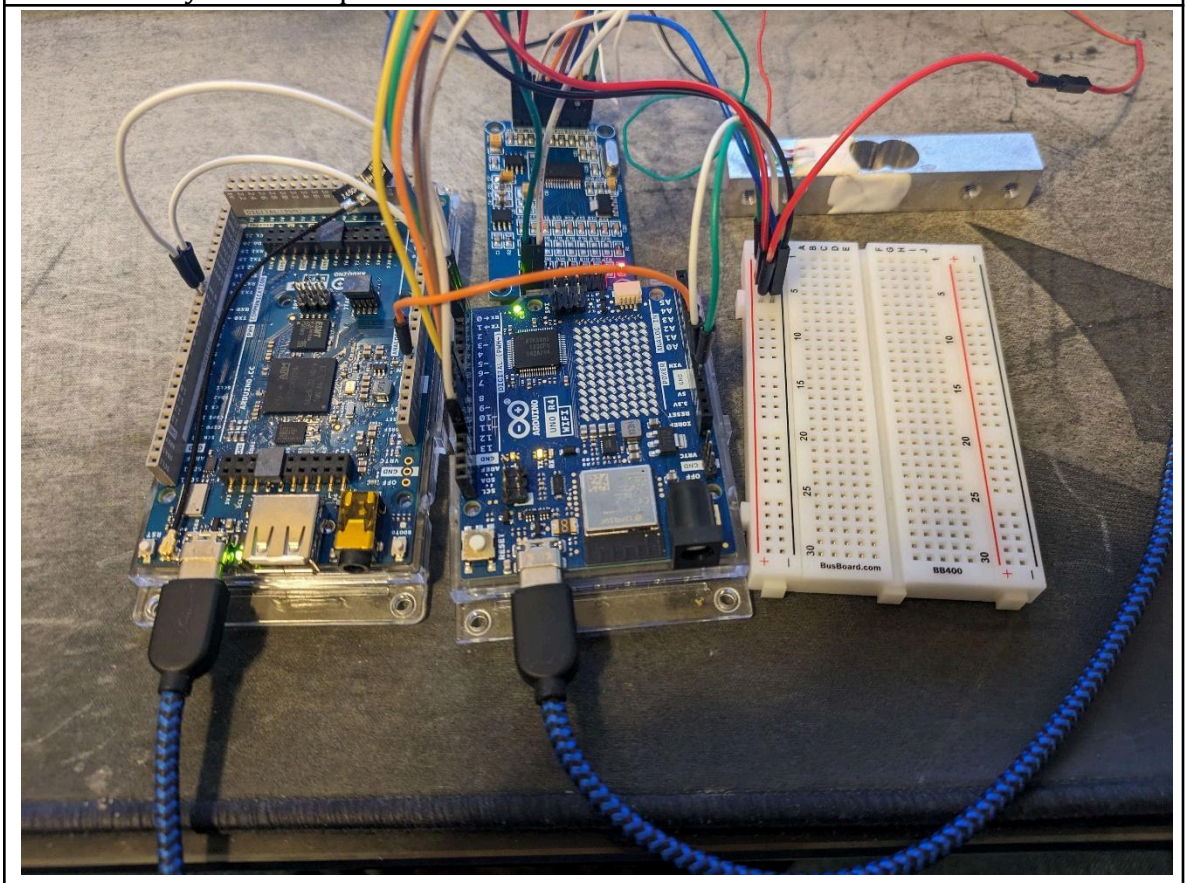
3.6 Software Off-The-Shelf Items

Software	**Description**
MATLAB	Graphs

DAQ Express	Data from National Instruments DAQ
Arduino IDE	Interact with Arduino Giga and Uno
Arduino Cloud	Arduino cloud service
Arduino IoT Remote	Arduino IoT Remote phone application
Edge Impulse	Building, deploying, and scaling embedded ML applications

3.7 Schematics/Wiring Diagrams/Mechanical Drawings

Load Cell Physical Set Up



3.8 Custom Software

Matlab for signal processing

Edge Impulse for Machine Learning solution

4. Prototype Design and Fabrication

As this is primarily intended as a research project, the following approach is not strictly intended to be successful. Choices in design were made based on available materials and information found in research papers. Sensors must initially be chosen based on relevance to what is being measured.

An initial structural model was first developed using Tinker CAD. Multiple materials were considered but a mix of TPU and PLA was chosen to provide a greater degree of vibration through the structure and to allow for damage to the structure to magnify the results. A 3-tiered structure was developed which provided ample degrees of shear and twist when mechanical force was applied. Rubber bands were used to simulate crossbeams and packing foam was used to simulate soil.

For this testing, flexible piezoelectric sensors, a vibration sensor, and load cell sensors were chosen. The flexible piezoelectric sensors are capable of being directly connected to the ADC of the microcontroller. In this case, an Arduino Giga is chosen. However, it is helpful to attenuate the signals prior to connection using a voltage divider. This project uses a mix of two flexible piezoelectric sensors. One set of four is provided through PolyK and measures 10mm x 25mm and four are from a generic brand found through Amazon which measure 10mm x 15mm. A single vibration sensor from TE Connectivity Measurement Specialties was chosen. Lastly, A set of four generic brand bar-style load cells were picked which were rated for 5 kg.

The ratio of resistors must first be chosen before connecting them to the ADC of the Arduino. This can be done by connecting the piezoelectric to the Arduino through the voltage divider and determining if the signal is clipped during experimentation. Successive voltage dividers should use equivalent resistor values. Must note the range of what the ADC can measure and select the correct voltage divider resistor

values. In this case, a maximum of 20V is considered. This can be discovered through testing the piezoelectric sensors in a realistic scenario and this step can be performed at a later point in time. The vibration sensor can be directly connected to the ADC. The load cells, however, must be connected to the ADS-1256 module. As publicly available libraries do not exist for this device which will run directly on an Arduino Giga, an Arduino Uno was chosen. The connection choices were the same as provided through Matt Bilsky's libraries and the available code was readapted for this purpose.

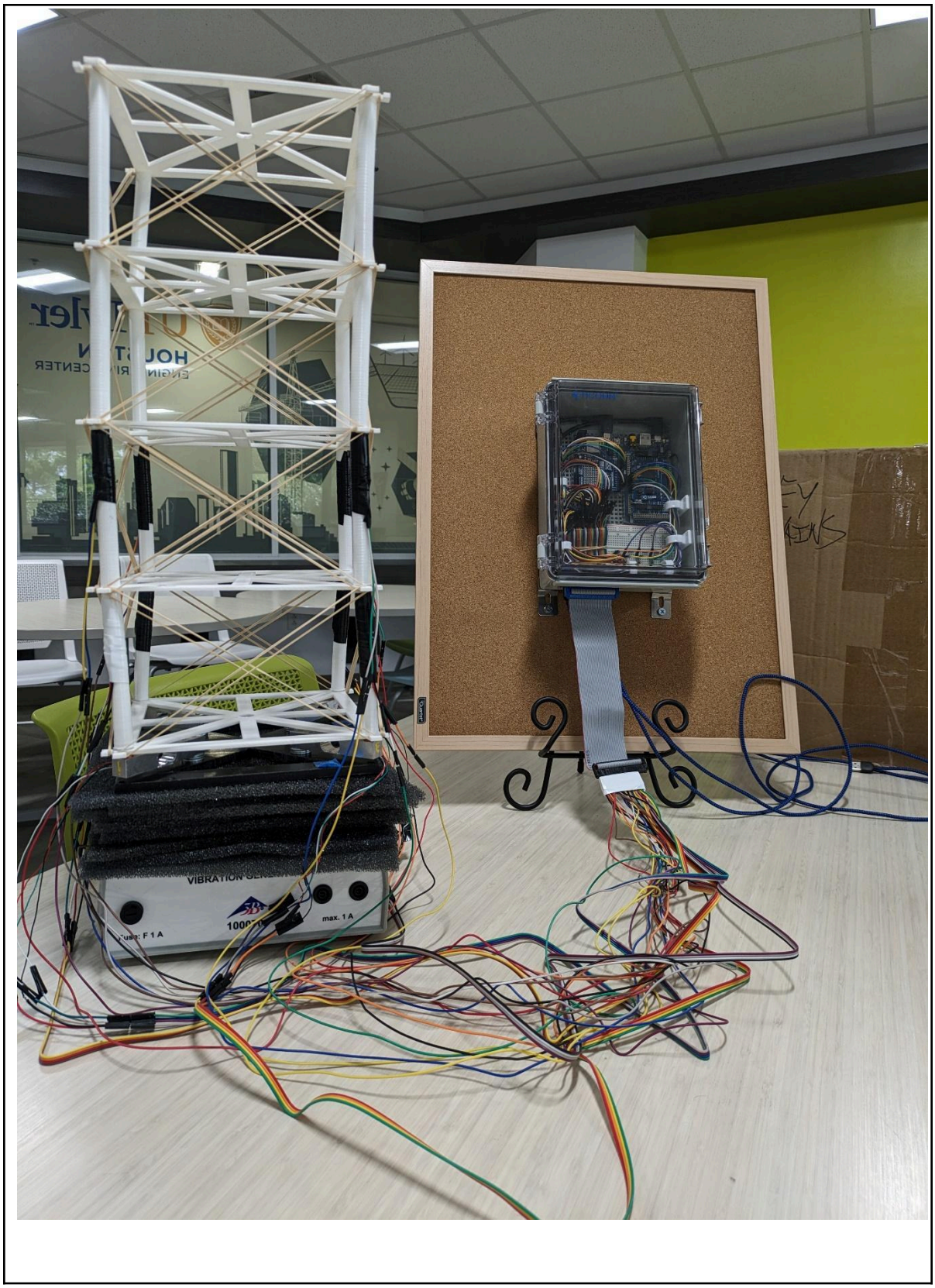
The various sensors should be affixed to the structure's critical points. These will be based on the geometry and load characteristics of the model. For this case, the first and second sets of columns had piezoelectric sensors affixed near their bases, the vibration sensor was affixed to the center of the lowest platform, and the load cells were affixed beneath the corners of the baseplate which was used to affix the structure to the vibration generator. Code must be written to collect data through both systems. The Arduino IDE provides convenient access to and importation of libraries and allows for C++ to be used. Data can be transferred wirelessly from the Arduino Uno to the Arduino Giga in chunks, which can then be transferred to a PC then stored in a CSV format for further processing.

For the testing procedure, connect an ammeter between the function generator and vibration generator then maximize the amplitude of the function generator and step through frequencies until the structure's resonance frequency has been discovered. This will provide a maximum physical amplitude for the vibrations. For convenience, round to the nearest multiple of 5 in Hertz. Record the data in 60 second segments. Alternate between different frequencies in 5 Hertz increments, then move back to the resonant frequency and alternate between different amplitudes. It is suggested to use simple fractions for the amplitude outputs such as ¼, ½, ¾, and the maximum amplitude. Once the set of data is recorded, damage a column and note the quantity and location of damage then repeat the recording process. Damage should be performed on columns at each floor of the structure and then replaced with an intact column before further damage is applied. Once the column damage has been fully

recorded, do the same with the cross beams of the structure. This can be conveniently performed by removing individual rubber bands to simulate quantities of damage to the cross beams.

With the recorded data and CSV files, MATLAB should be used to perform signal processing techniques. Processing techniques should include FFT, STFT, PSD, Wavelet transform, and spectrogram results. The relevant data should be output to their own CSV files with respect to the type of damage which has occurred. Machine learning models created using PyTorch, NumPy, Scikit-learn, TensorFlow, and Pandas are recommended. The model should be able to detect the location and quantity of damage to the structure.

Prototype Set Up


VIBRATION
max. 1 A

Bill of Materials (BOM)					
Item	**Description**	**Quantity**	**Units**	**Unit Cost**	**Cost**
400mm Threaded Rods and Studs Assortment	Assortment of stainless-steel threaded rods and studs for mechanical connecting.	1	2	$1.00	$2.00
Digital Load Cell Weight Sensor (1KG)	1-Kg weight sensor	4	4	$7.59	$30.36
Digital Load Cell Weight Sensor (5KG)	5-Kg weight sensor	4	4	$7.59	$30.36
ADS1256 24-bit 8 Channel ADC AD Module	24-bit ADC module with SPI interface, supporting up to 30ksps data output rates.	1	1	$14.93	$14.93
LM358 Operational Amplifier Module	Set of LM358 operational amplifiers, suitable for a wide range of power supplies.	5	1	$8.09	$8.09
Flexible Piezoelectric Film Sensor High Sensitive Polymer PVDF	Piezoelectric film sensor	4	4	$9.86	$39.44
Metric Bolt Assortment M2 M3 M4, 21 Sizes 4MM to 20MM	Bolt assortment kit including screws, nuts, and washers.	1	1	$23.97	$23.97
55PCS M2 M2.5 M3 Threaded Insert Assortment Kit	Assortment of threaded inserts for creating durable threaded holes.	1	1	$15.52	$15.52
Banana Plugs	Gold plated banana plugs, open screw type connector	8	1	$6.46	$6.46
Arduino Giga R1	dual-core STM32H747XI microprocessor with a Cortex-M7 and a Cortex-M4 core, 76 digital I/O pins, and 12 analog input pins.	2	2	$73.00	$146.00
Arduino Uno	The Arduino Uno microcontroller board	1	1	$28.00	$28.00
NI USB 6211	A multifunction DAQ device that offers analog I/O, digital input, digital output, and two 32-bit counters	1	1	$650.00	$650.00
PLA filament	3D printing material	3	3	$24.99	$74.97
TPU Filament	3D printing material	3	3	$24.99	$74.97
16GB SDHC Flash Memory Card	Memory card for Arduino DAQ processing	4	4	$4.00	$16.00
			Total	**$899.99**	**$1,161.07**
Record of Expenditures for Prototype Development – Designing, Creating, Testing					

Man Hours	Description
150	Designing, Creating, Testing
50	Reproduce protype given current knowledge

5. Testing and Validation

Testing is performed for various stages of the software pipeline.

Data Acquisition:

Testing of the data acquisition system was successful. The modules which acquire data from the analog-to-digital converters perform properly and acquire accurate data to be used in the signal processing pipeline.

Signal Processing:

Signal processing was successfully performed on a computer. The embedded software is still in the development phase; therefore, this testing is incomplete.

Machine Learning:

The machine learning model currently has 98.6% accuracy and is ready to be exported to the embedded system. This is a satisfactory level of accuracy.

Networking System:

The system for relaying data to an end user has been tested via an HTTP protocol. This is not the final iteration of this subsystem and has room for future improvement.

6. Broader Impacts of the Project

Our broader aspirations with this project were to deliver a system which could acquire data from a structure being shaken at various levels of damage, and to interpret that data accurately for an end user. Although the application and machine learning have not been finalized, the signal processing and acquisition from our 3D printed model have been successful. Our team has demonstrated repeatable patterns in the data when processed in different ways, corresponding to the damage of the structure. These findings lead us to be confident in our ability to develop an algorithm able to detect these findings.

Structural Health Monitoring takes place in the nexus of IoT and Wireless Sensor Networks. With the collection of vast amounts of data and the implementation of AI to groom the feedback, this project sits comfortably between several novel and

budding fields of the last couple decades. We are using a state-of-the-art dual processor controller in the design, and we had considered the concept of digital twinning in order to simulate a structure being shaken.

7. Discussions and Conclusions

Structural Health Monitoring (SHM) Solution Using Arduino, Sensors, and Edge Impulse. The work completed in designing and testing the structural health monitoring (SHM) solution using the Arduinos, sensors, and Edge Impulse provides an innovative approach that aligns well with the project's objectives. The designed solution successfully integrates low-cost hardware, advanced signal processing techniques, and machine learning to achieve real-time feedback on structural integrity, enabling proactive maintenance practices and fast emergency response times during disasters.

Assessment of the Design Solution:

The designed solution effectively incorporates a 3D printed modular and multi-material structure, various sensors including flexible piezoelectric sensors, a vibration sensor, and load cell sensors, along with microcontrollers for real-time data acquisition and preprocessing. The inclusion of signal processing techniques such as Fast Fourier Transform (FFT) and Short-time Fourier Transform (STFT) effectively distinguishes stable and active signals, fulfilling the requirement for identifying structural abnormalities. The system's scalability, incorporation of machine learning algorithms, and real-time feedback through a future mobile app address the need for proactive structural health monitoring and demonstrate the success of the final design.

Analysis of Success and Limitations:

While the design solution successfully meets the project's objectives, it could benefit from a more focused approach tailored to specific structural points of concern. We

are looking to refine the monitoring system to include specific factors such as vibrations, location and type of damage, and material quality. Designing the system around the structure it will be implemented in will allow for more granular approaches and address project specific needs.

Recommendations for Future Development:

Moving forward, several recommendations for future work and development related to the SHM design include:

1. Incorporating a more robust machine learning model for continual refinement and adaptability.
2. Exploring new methodologies and refining the system through additional experimental data.
3. Implementing a wireless sensor network for real-time data transmission and communication between sensors.
4. Ensuring secure data transmission to safeguard the system against potential cyber threats.
5. Integration of mixed reality elements to enhance user experience and visualization of monitoring capabilities.
6. Expanding the use of piezoelectric sensors for more accurate and sensitive data collection.
7. Enhancing the integration with the mobile app for real-time feedback and remote monitoring.
8. Integrating the system with cloud services and enabling two-way communication for scalability and responsiveness.
9. Implementing real-time signal recognition for quick detection and response to structural abnormalities.

The Arduino Giga R1 Wi-Fi best fits these needs while remaining affordable. However, further research is needed to determine the number of sensors to be utilized.

In summary, our SHM system has laid a solid groundwork for the proactive maintenance of infrastructure safety. By embracing the recommended future enhancements, our solution has potential to be a more resilient and efficient solution catering to the evolving demands of infrastructure safety and maintenance. We aim to create a comprehensive SHM solution that not only ensures infrastructure integrity but also enables proactive intervention to the longevity and safety of our built environment.

8. References

1. Jeff Masters and Bob Henson, “The U.S. had 18 different billion-dollar weather disasters in 2022”, Yale Climate Connections, January 2023

2. “A Comprehensive Guide to Fine-Tuning Large Language Models.”

3. J. P. Amezquita-Sanchez and H. Adeli, “Signal Processing Techniques for Vibration-Based Health Monitoring of Smart Structures,” *Archives of Computational Methods in Engineering*, vol. 23, no. 1, pp. 1–15, Mar. 2016, doi: 10.1007/s11831-014-9135-7.

4. M. A. Arefeen, B. Debnath, and S. Chakradhar, “Lean Context: Cost-Efficient Domain-Specific Question Answering Using LLMs,” Sep. 2023, [Online]. Available: http://arxiv.org/abs/2309.00841

5. T. Baldazzi, L. Bellomarini, S. Ceri, A. Colombo, A. Gentili, and E. Sallinger, “Fine-tuning Large Enterprise Language Models via Ontological Reasoning,” Jun. 2023, [Online]. Available: http://arxiv.org/abs/2306.10723

6. Y. Bao and H. Li, “Machine learning paradigm for structural health monitoring,” *Structural Health Monitoring*, vol. 20, no. 4, pp. 1353–1372, Jul. 2021, doi: 10.1177/1475921720972416.

7. R. Behnia, M. R. Ebrahimi, J. Pacheco, and B. Padmanabhan, “EW-Tune: A Framework for Privately Fine-Tuning Large Language Models with Differential Privacy,” in *IEEE International Conference on Data Mining Workshops, ICDMW*, IEEE Computer Society, 2022, pp. 560–566. doi: 10.1109/ICDMW58026.2022.00078.

8. C. Chen, B. Liu, S. Wan, P. Qiao, and Q. Pei, “An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1840–1852, Mar. 2021, doi: 10.1109/TITS.2020.3025687.

9. C. Chen, B. Liu, S. Wan, P. Qiao, and Q. Pei, “An Edge Traffic Flow Detection Scheme Based on Deep Learning in an Intelligent Transportation System,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1840–1852, Mar. 2021, doi: 10.1109/TITS.2020.3025687.

10. P. K. Donta and S. Dustdar, “Towards Intelligent Data Protocols for the Edge,” in *Proceedings - IEEE International Conference on Edge Computing*, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 372–380. doi: 10.1109/EDGE60047.2023.00060.

11. D. Goyal and B. S. Pabla, “The Vibration Monitoring Methods and Signal Processing Techniques for Structural Health Monitoring: A Review,” *Archives of Computational Methods in Engineering*, vol. 23, no. 4, pp. 585–594, Dec. 2016, doi: 10.1007/s11831-015-9145-0.

12. S. Hassani and U. Dackermann, “A Systematic Review of Advanced Sensor Technologies for Non-Destructive Testing and Structural Health Monitoring,” *Sensors*, vol. 23, no. 4. MDPI, Feb. 01, 2023. doi: 10.3390/s23042204.

13. A. Hernández-Blanco, B. Herrera-Flores, D. Tomás, and B. Navarro-Colorado, “A Systematic Review of Deep Learning Approaches to Educational Data Mining,” *Complexity*, vol. 2019. Hindawi Limited, 2019. doi: 10.1155/2019/1306039.

14. S. Hong, J. S. Kim, and Y. Joon Jung, “Lightweight Object Recognizer for Edge Devices,” in *International Conference on ICT Convergence*, IEEE Computer Society, 2022, pp. 1729–1731. doi: 10.1109/ICTC55196.2022.9952447.

15. H. Hu, M. Tang, L. Li, H. Hu, and S. Qiao, “Signal processing techniques for structural health monitoring of super high-rise buildings,” in *IOP Conference Series: Earth and Environmental Science*, Institute of Physics Publishing, Nov. 2019. doi: 10.1088/1755-1315/330/2/022015.

16. G. Kakareko, S. Jung, and O. A. Vanli, “Hurricane risk analysis of the residential structures located in Florida,” *Sustainable and Resilient Infrastructure*, vol. 5, no. 6, pp. 395–409, Nov. 2020, doi: 10.1080/23789689.2019.1632599.

17. H. Kumar, A. R. Jadhav, S. Gvk, J. Bapat, and D. Das, “Intelligent Edge Detection of Attacks on IP-based IoT deployments,” in *Proceedings - 2021 19th OITS International Conference on Information Technology, OCIT 2021*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 132–137. doi: 10.1109/OCIT53463.2021.00036.

18. A. Moreno-Gomez, C. A. Perez-Ramirez, A. Dominguez-Gonzalez, M. Valtierra-Rodriguez, O. Chavez-Alegria, and J. P. Amezquita-Sanchez, “Sensors Used in Structural Health Monitoring,” *Archives of Computational Methods in Engineering*, vol. 25, no. 4, pp. 901–918, Nov. 2018, doi: 10.1007/s11831-017-9217-4.

19. A. Mufti and K. Helmi, “A case for structural health monitoring (SHM) and civionics enhances the evaluation of the load carrying capacity of aging

bridges," *Innovative Infrastructure Solutions*, vol. 4, no. 1, Dec. 2019, doi: 10.1007/s41062-018-0187-7.

20. H. Qarib and H. Adeli, "A comparative study of signal processing methods for structural health monitoring," *Journal of Vibroengineering*, vol. 18, no. 4, pp. 2186–2204, 2016, doi: 10.21595/jve.2016.17218.

21. W. Qu, X. Ding, K. Yang, Y. Bao, and W. Chen, "IDEC: Intelligent Distributed Edge Computing System Architecture Enabling Deep Learning across Heterogeneous IoT Devices," in *2020 IEEE 6th International Conference on Computer and Communications, ICCC 2020*, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 926–933. doi: 10.1109/ICCC51575.2020.9345083.

22. A. Raha *et al.*, "Special Session: Approximate TinyML Systems: Full System Approximations for Extreme Energy-Efficiency in Intelligent Edge Devices," in *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 13–16. doi: 10.1109/ICCD53106.2021.00015.

23. A. Sofi, J. Jane Regita, B. Rane, and H. H. Lau, "Structural health monitoring using wireless smart sensor network – An overview," *Mechanical Systems and Signal Processing*, vol. 163, Jan. 2022, doi: 10.1016/j.ymssp.2021.108113.

24. S. Sony, S. Laventure, and A. Sadhu, "A literature review of next-generation smart sensing technology in structural health monitoring," *Structural Control and Health Monitoring*, vol. 26, no. 3. John Wiley and Sons Ltd, Mar. 01, 2019. doi: 10.1002/stc.2321.

25. U. K. Sreekumar, R. Devaraj, Q. Li, and K. Liu, "Real-time traffic pattern collection and analysis model for intelligent traffic intersection," in *Proceedings - 2018 IEEE International Conference on Edge Computing, EDGE 2018 - Part of the 2018 IEEE World Congress on Services*, Institute of Electrical and

Electronics Engineers Inc., Sep. 2018, pp. 140–143. doi: 10.1109/EDGE.2018.00028.

26. C. Surianarayanan, P. Raj, and S. K. Niranjan, “The Significance of Edge AI towards Real-time and Intelligent Enterprises,” in Proceedings of the International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics, ICIITCEE 2023, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 1–6. doi: 10.1109/IITCEE57236.2023.10090926.

27. Tallinna Tehnikaülikool. Department of Electronics. and IEEE Circuits and Systems Society., BEC 2006: 2006 International Baltic Electronics Conference: proceedings of the 10th biennial Baltic Electronics Conference: Tallinn University of Technology, October 2-4, 2006, Tallinn, Estonia. IEEE, 2006.

28. B. Xu, K. Mou, Institute of Electrical and Electronics Engineers. Beijing Section, and Institute of Electrical and Electronics Engineers, Proceedings of 2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA 2020): November 6-8, 2020, Chongqing, China.

29. X. W. Ye, T. Jin, and C. B. Yun, “A review on deep learning-based structural health monitoring of civil infrastructures,” *Smart Structures and Systems*, vol. 24, no. 5. Techno-Press, pp. 567–585, Nov. 01, 2019. doi: 10.12989/sss.2019.24.5.567.

Appendix A – Arduino Giga code

```
using namespace std;

int piezo1;
int piezo2;
int piezo3;
int piezo4;
int piezo5;
int piezo6;
int piezo7;
int piezo8;

int value;
char buf[30];
char charload1[6];
char charload2[6];
char charload3[6];
char charload4[6];
long tempconv;
long load1;
long load2;
long load3;
long load4;
char *output;
long timer;
const long interval = 4000;
unsigned long previousmicros = 0;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(921600);
  Serial1.begin(921600);
  analogReadResolution(16);
  timer = 0;
}

void loop() {
  unsigned long currentmicros = micros();
  if(currentmicros - previousmicros >= interval){

  Serial1.write('g');
  int rlen = Serial1.readBytes(buf, 30);
```

```
for(int i = 0; i < 6; i++){
  charload1[i] = buf[i+2];
  charload2[i] = buf[i+9];
  charload3[i] = buf[i+16];
  charload4[i] = buf[i+23];
}

load1 = 0;
load2 = 0;
load3 = 0;
load4 = 0;

tempconv = charload1[0] - '0';
load1 += tempconv * 100000;
tempconv = charload1[1] - '0';
load1 += tempconv * 10000;
tempconv = charload1[2] - '0';
load1 += tempconv * 1000;
tempconv = charload1[3] - '0';
load1 += tempconv * 100;
tempconv = charload1[4] - '0';
load1 += tempconv * 10;
tempconv = charload1[5] - '0';
load1 += tempconv;

tempconv = charload2[0] - '0';
load2 += tempconv * 100000;
tempconv = charload2[1] - '0';
load2 += tempconv * 10000;
tempconv = charload2[2] - '0';
load2 += tempconv * 1000;
tempconv = charload2[3] - '0';
load2 += tempconv * 100;
tempconv = charload2[4] - '0';
load2 += tempconv * 10;
tempconv = charload2[5] - '0';
load2 += tempconv;

tempconv = charload3[0] - '0';
load3 += tempconv * 100000;
tempconv = charload3[1] - '0';
load3 += tempconv * 10000;
tempconv = charload3[2] - '0';
load3 += tempconv * 1000;
tempconv = charload3[3] - '0';
load3 += tempconv * 100;
tempconv = charload3[4] - '0';
load3 += tempconv * 10;
tempconv = charload3[5] - '0';
load3 += tempconv;

tempconv = charload4[0] - '0';
load4 += tempconv * 100000;
tempconv = charload4[1] - '0';
load4 += tempconv * 10000;
tempconv = charload4[2] - '0';
load4 += tempconv * 1000;
```

```
  tempconv = charload4[3] - '0';
  load4 += tempconv * 100;
  tempconv = charload4[4] - '0';
  load4 += tempconv * 10;
  tempconv = charload4[5] - '0';
  load4 += tempconv;


  piezo1 = analogRead(A0);
  piezo1 = analogRead(A0);
  piezo2 = analogRead(A1);
  piezo2 = analogRead(A1);
  piezo3 = analogRead(A2);
  piezo3 = analogRead(A2);
  piezo4 = analogRead(A3);
  piezo4 = analogRead(A3);
  piezo5 = analogRead(A4);
  piezo5 = analogRead(A4);
  piezo6 = analogRead(A5);
  piezo6 = analogRead(A5);
  piezo7 = analogRead(A6);
  piezo7 = analogRead(A6);
  piezo8 = analogRead(A7);
  piezo8 = analogRead(A7);

  Serial.print(load1);
  Serial.print(',');
  Serial.print(load2);
  Serial.print(',');
  Serial.print(load3);
  Serial.print(',');
  Serial.print(load4);
  Serial.print(',');
  Serial.print(piezo1);
  Serial.print(',');
  Serial.print(piezo2);
    Serial.print(',');
  Serial.print(piezo3);
    Serial.print(',');
  Serial.print(piezo4);
    Serial.print(',');
  Serial.print(piezo5);
    Serial.print(',');
  Serial.print(piezo6);
    Serial.print(',');
  Serial.print(piezo7);
    Serial.print(',');
  Serial.println(piezo8);


  previousmicros = currentmicros;
  }
}
```

Appendix B – Arduino Uno ADS main code

```
#include <SPI.h>
#include <digitalWriteFast.h>
#include <iostream>
using namespace std;

//(other stuff for getting the ADS1526 to work is in the next tab
#define ADS_RST_PIN    9 //ADS1256 reset pin
#define ADS_RDY_PIN    2 //ADS1256 data ready
#define ADS_CS_PIN    10 //ADS1256 chip select

/*
    CLK  - pin 13
    DIN  - pin 11 (MOSI)
    DOUT - pin 12 (MISO)
*/


//put the ADC constants here
String sendarr[3];
double resolution = 8388608.; //2^23-1

//this needs to match the setting in the ADC init function in the library
tab
double Gain = 64.; //be sure to have a period

double vRef = 5.0; //reference voltage

//we'll calculate this in setup
double bitToVolt = 0.;

char bufsend[30] = {};

void setup() {
  delay(1000);
  Serial.begin(921600);
  Serial1.begin(921600);
  Serial.println("booting");
  //initialize the ADS
  pinMode(ADS_CS_PIN, OUTPUT);

  pinMode(ADS_RDY_PIN, INPUT);
  pinMode(ADS_RST_PIN, OUTPUT);

  SPI.begin();

  initADS();
  Serial.println("done init");

  //determine the conversion factor
    //do some calculations for the constants
  bitToVolt = resolution*Gain/vRef;
  bufsend[0] = ' ';
  bufsend[1] = '<';
  bufsend[8] = ',';
  bufsend[15] = ',';
  bufsend[22] = ',';
  bufsend[29] = '>';
```

```
}

int32_t val1;
int32_t val2;
int32_t val3;
int32_t val4;
int32_t val5;
int32_t val6;
int32_t val7;
int32_t val8;
int flag = 0;
//char flagtest[1];
char flagtest[1] = {'s'};
char buf1[6];
char buf2[6];
char buf3[6];
char buf4[6];

String strval1;
String strval2;
String strval3;

void loop() {
  // put your main code here, to run repeatedly:
  Serial1.readBytes(flagtest, 1);
  if(flagtest[0] == 'g'){
  double value1 = 0;
  double value2 = 0;
  double value3 = 0;
  double value4 = 0;
  for (int i = 0; i <1; i++){
    read_all_channels();
    value1 += val1;
    value2 += val3;
    value3 += val5;
    value4 += val7;
  }

  value1 /=1.;
  value2 /=1.;
  value3 /=1.;
  value4 /=1.;

  value1 += 400000;
  value2 += 400000;
  value3 += 400000;
  value4 += 400000;

  dtostrf(value1, 1, 0, buf1);
  dtostrf(value2, 1, 0, buf2);
  dtostrf(value3, 1, 0, buf3);
  dtostrf(value4, 1, 0, buf4);
  for(int i = 0; i < 6; i++) {
    bufsend[i+2] = buf1[i];
    bufsend[i+9] = buf4[i];
    bufsend[i+16] = buf3[i];
    bufsend[i+23] = buf2[i];
```

```
  }
  Serial1.write(bufsend, 30);
  flagtest[0] = 's';
  }
}
```

Appendix C – Arduino Uno ADS definitions code

```
//using the definitions in this library:
https://github.com/Flydroid/ADS12xx-Library

#define SPI_SPEED 2500000


/* For information to the register and settings see manual page (p..) */

/* ADS1248 Register (see p42 for Register Map) */

#define    STATUS    0x00 //Status Control Register 0
#define   MUX     0x01 //Multiplexer Control Register 0
#define   MUX2    0x23 //Multiplexer Control Register 1 (self added)
#define   ADCON     0x02 //A/D Control Register 0
#define   DRATE   0x03 //A/D Data Rate Control Register 0
#define   IO        0X04 //GPIO Control Register 0
#define   OFC0    0x05 //Offset Calibration Coefficient Register 1
#define   OFC1    0x06 //Offset Calibration Coefficient Register 2
#define   OFC2    0x07 //Offset Calibration Coefficient Register 2
#define   FSC0    0x08 //Full scale Callibration Coefficient Register 0
#define   FSC1    0x09 //Full scale Callibration Coefficient Register 1
#define   FSC2    0x0A //Full scale Callibration Coefficient REgister 2

/*STATUS - Status Control Register 0 ( see p30)*/
/* BIT7 - BIT6 -  BIT5   -  BIT4   -  BIT3   -  BIT2   -  BIT1   -  BIT0 */
/* ID   - ID   -  ID     -  ID     -  ORDER  -  ACAL   -  BUFEN  -  DRDY */
#define STATUS_RESET 0x01 // Reset STATUS Register
/*Bits 7 - 4 ID3, ID2, ID1, ID0 Factory Programmed Identification Bits(Read
Only)*/
/*ORDER1:0  Data Output Bit Order*/
#define ORDER_MSB B00000000 // Most significant Bit first (default)
#define ORDER_LSB B00001000//Least significant Bit first
/*Input data is always shifted in most significant byte and bit first.
Output data is always shifted out most significant
byte first. The ORDER bit only controls the bit order of the output data
within the byte.*/
/*ACAL1:0 Auto Calibration*/
#define ACAL_OFF B00000000 // Auto Calibration Disabled (default)
#define ACAL_ON  B00000100 // Auto Calibration Enabled
/*When Auto-Calibration is enabled, self-calibration begins at the
completion of the WREG command that changes
```

```
the PGA (bits 0-2 of ADCON register), DR (bits 7-0 in the DRATE register) or
BUFEN (bit 1 in the STATUS register)
values.*/
/*BUFEN1:0 Analog Input Buffer Enable*/
#define BUFEN_OFF B00000000 //Buffer Disabled (default)
#define BUFEN_ON  B00000010 //BUffer Enabled
/*DRDY1:0 Data Ready (Read Only) Duplicates the state of the DRDY pin*/

/* MUX - Multiplexer Control Register 0 (see p31 - bring together with
bitwise OR | */
/* BIT7  - BIT6  -  BIT5   -  BIT4   -  BIT3   -  BIT2   -  BIT1   -  BIT0
*/
/* PSEL3 - PSEL2 -  PSEL1  -  PSEL0  -  NSEL3  -  NSEL2   - NSEL1   - NSEL0
*/
#define MUX_RESET 0x01      // Reset MUX0 Register
/* PSEL3:0 Positive input channel selection bits */
#define P_AIN0 B00000000 //(default)
#define P_AIN1 B00010000
#define P_AIN2 B00100000
#define P_AIN3 B00110000
#define P_AIN4 B01000000
#define P_AIN5 B01010000
#define P_AIN6 B01100000
#define P_AIN7 B01110000
#define P_AINCOM B10000000
/* NSEL3:0 Negativ input channel selection bits */
#define N_AIN0 B00000000
#define N_AIN1 B00000001 //(default)
#define N_AIN2 B00000010
#define N_AIN3 B00000011
#define N_AIN4 B00000100
#define N_AIN5 B00000101
#define N_AIN6 B00000110
#define N_AIN7 B00000111
#define N_AINCOM B00001000

/*ADCON - A/D Control Register 0 ( see p31)*/
/* BIT7 - BIT6   -  BIT5   -  BIT4   -  BIT3   -  BIT2   -  BIT1   -  BIT0
*/
/* 0    - CLK1   -  CLK0   -  SDCS1  -  SDCS0  -  PGA2   -  PGA1   -  PAG0
*/
#define ADCON_RESET 0x20 // Reset ADCON Register
/*CLK2:0 D0/CLKOUT Clock Out Rate Setting*/
#define CLK_OFF B00000000 //Clock Out off
#define CLK_1   B00100000 //Clock Out Frequency = fCLKIN (default)
#define CLK_2   B01000000 //Clock Out Frequency = fCLKIN/2
#define CLK_4   B01100000 //Clock Out Frequency = fCLKIN/4
/*When not using CLKOUT, it is recommended that it be turned off. These bits
can only be reset using the RESET pin.*/
/*SDCS2:0 Sensor Detection Current Sources*/
#define SDCS_OFF B00000000//Sensor Detect Off (default)
#define SDCS_05  B00001000//Sensor Detect Current 0.5?A
#define SDCS_2   B00010000//Sensor Detect Current 2?A
#define SDCS_10  B00011000//Sensor Detect Current 10?A
/*The Sensor Detect Current Sources can be activated to verify the integrity
of an external sensor supplying a signal to the
```

```
ADS1255/6. A shorted sensor produces a very small signal while an
open-circuit sensor produces a very large signal.*/
/*PGA3:0 Programmable Gain Amplifier Setting*/
#define PGA_1 //(default)
#define PGA_2
#define PGA_4
#define PGA_8
#define PGA_16
#define PGA_32
#define PGA_64 B00100111

/*DRATE - A/D Data Rate Register 0 ( see p32)*/
/* BIT7 - BIT6   -  BIT5   -  BIT4   -  BIT3   -  BIT2   -  BIT1   -  BIT0
*/
/* DR7  - DR6    -  DR5    -  DR4    -  DR3    -  DR2    -  DR1    -  DR0 */
#define DRATE_RESET 0xF0 // Reset DRATE Register
/*DR7:0 Data Rate Setting*/
#define DR_30000 B11110000 //30.000 SPS (default)
#define DR_15000 B11100000 //15.000 SPS
#define DR_7500  B11010000 //7.500 SPS
#define DR_3750  B11000000 //3.750 SPS
#define DR_2000  B10110000 //2.000 SPS
#define DR_1000  B10100001 //1.000 SPS
#define DR_500   B10010010 //500 SPS
#define DR_100   B10000010 //100 SPS
#define DR_60    B01110010 //60 SPS
#define DR_50    B01100011 //50 SPS
#define DR_30    B01010011 //30 SPS
#define DR_25    B01000011 //25 SPS
#define DR_15    B00110011 //15 SPS
#define DR_10    B00100011 //10 SPS
#define DR_5     B00010011 //5 SPS
#define DR2_5    B00000011 //2,5 SPS

/*IO - GPIO Control Register 0 ( see p32)*/
/* BIT7 - BIT6   -  BIT5   -  BIT4   -  BIT3   -  BIT2   -  BIT1   -  BIT0
*/
/* DIR3 - DIR2   -  DIR1   -  DIR0   -  DIO3   -  DIO2   -  DIO1   -  DIO0
*/
#define IO_RESET 0xE0 // Reset IO Register
/*DIR3 - Digital I/O Direction for Pin D3*/
#define DIR3_OUT B00000000 //D3 is an output
#define DIR_IN   B10000000 //D3 is an input (default)
/*DIR2 - Digital I/O Direction for Pin D3*/
#define DIR2_OUT B00000000 //D2 is an output
#define DIR2_IN   B01000000 //D2 is an input (default)
/*DIR1 - Digital I/O Direction for Pin D3*/
#define DIR1_OUT B00000000 //D1 is an output
#define DIR1_IN   B00100000 //D1 is an input (default)
/*DIR0 - Digital I/O Direction for Pin D3*/
#define DIR0_OUT B00000000 //D0/CLKOUT is an output
#define DIR0_IN   B00010000 //D0/CLKOUT is an input (default)
/*DIO3:0 Status of Digital I/O, Read Only*/

/* SPI COMMAND DEFINITIONS (p34) */
/*SYSTEM CONTROL */
#define   WAKEUP    0x00  //Exit Sleep Mode
```

```
#define   STANDBY   0xFD  //Enter Sleep Mode
#define   SYNC    0xFC    //Synchornize the A/D Conversion
#define   RESET   0xFE  //Reset To Power UP values
#define   NOP     0xFF  //No operation
/*DATA READ*/
#define   RDATA   0x01  //Read data once
#define   RDATAC    0x03  //Read data continously
#define   SDATAC    0x0F  //Stop reading data continously
/*READ REGISTER */
#define   RREG    0x10  //Read From Register
#define   WREG    0x50  //Write To Register
/*Calibration */
#define   SYSOCAL   0xF3  //System Offset Calibration
#define   SYSGCAL   0xF2  //System Gain Calibration
#define   SELFCAL     0xF0  //Self Offset Calibration
```

Appendix D – Arduino Uno ADS library code

```
//my code
//note...this is written for TEENSY meaning I am using DigitalWriteFast to
speed things up.
//thus the CS pin must be hard coded. in my case, this is pin 21 but you
will have to change it for yourself if needed
//see this post about how to use:
https://forum.pjrc.com/threads/24573-Speed-of-digitalRead-and-digitalWrite-w
ith-Teensy3-0

//built up on the work of:
//https://github.com/Flydroid/ADS12xx-Library
//https://gist.github.com/dariosalvi78/f2e990b4317199d235bbf5963c3486ae
//https://github.com/adienakhmad/ADS1256


void initADS() {
  attachInterrupt(digitalPinToInterrupt(ADS_RDY_PIN), DRDY_Interuppt,
FALLING);

  digitalWrite(ADS_RST_PIN, LOW);
  delay(10); // LOW at least 4 clock cycles of onboard clock. 100
microsecons is enough
  digitalWrite(ADS_RST_PIN, HIGH); // now reset to deafult values

  delay(1000);

  //now reset the ADS
  Reset();

  //let the system power up and stabilize (datasheet pg 24)
  delay(2000);
  //this enables the buffer which gets us more accurate voltage readings
 // SetRegisterValue(STATUS,B00110010);

  Serial.println(GetRegisterValue(STATUS));
```

```
  //next set the mux register
  //we are only trying to read differential values from pins 0 and 1. your
needs may vary.
  //this is the default setting so we can just reset it
  SetRegisterValue(MUX,MUX_RESET); //set the mux register
   //B00001000 for single ended measurement

  //now set the ADCON register
  //set the PGA to 64x
  //you need to adjust the constants for the other ones according to
datasheet pg 31 if you need other values
  SetRegisterValue(ADCON, PGA_64); //set the adcon register

  //next set the data rate
  SetRegisterValue(DRATE, DR_30000); //set the drate register

  //we're going to ignore the GPIO for now...

  //lastly, we need to calibrate the system

  //let it settle
  delay(2000);

  //then do calibration
  SendCMD(SELFCAL); //send the calibration command

  //then print out the values
  delay(5);

  Serial.print("OFC0: ");
  Serial.println(GetRegisterValue(OFC0));
  Serial.print("OFC1: ");
  Serial.println(GetRegisterValue(OFC1));
  Serial.print("OFC2: ");
  Serial.println(GetRegisterValue(OFC2));
  Serial.print("FSC0: ");
  Serial.println(GetRegisterValue(FSC0));
  Serial.print("FSC1: ");
  Serial.println(GetRegisterValue(FSC1));
  Serial.print("FSC2: ");
  Serial.println(GetRegisterValue(FSC2));
}

//function to read a value
//this assumes that we are not changing the mux action
int32_t read_Value() {
  int32_t adc_val;
  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready
  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  adc_val |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val <<= 8;
```

```
  adc_val |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val <<= 8;
  adc_val |= SPI.transfer(NOP);
  //delayMicroseconds(5);
  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val > 0x7fffff) { //if MSB == 1
    adc_val = adc_val - 16777216; //do 2's complement, keep the sign this
time!
  }

  return adc_val;
}

//this function will go through the process of resetting the system and
cycling through the inputs
//it assumes that whatever was called before left the mux at the default
(0,1)
//this means it will treat the first value it receives as from (0,1)
//and the second from (2,3).

void read_two_values() {
  //datasheet page 21 at the bottom gives the timing
  int32_t adc_val1;
  int32_t adc_val2;

  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready

  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x23);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(2);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
```

```
  //This is the reading in the Data register from whatever the mux was set
to to the last
  //time this function was called. By default, it is configured to leave
that value at 0
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val1 <<= 8;
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val1 <<= 8;
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);

  //now change the mux register back to 0 so we left things how we found
them
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(MUX_RESET);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(2);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  //this should now be the value from the mux change we just did (0,1 to
2,3)
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val2 <<= 8;
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val2 <<= 8;
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(5);
  //this is the value for the

  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val1 > 0x7fffff) { //if MSB == 1
```

```
    adc_val1 = adc_val1 - 16777216; //do 2's complement, keep the sign this
time!
  }

  if (adc_val2 > 0x7fffff) { //if MSB == 1
    adc_val2 = adc_val2 - 16777216; //do 2's complement, keep the sign this
time!
  }

  val1 = adc_val1;

  val2 = adc_val2;
}

void read_all_channels() {
  //datasheet page 21 at the bottom gives the timing
  int32_t adc_val1;
  int32_t adc_val2;
  int32_t adc_val3;
  int32_t adc_val4;
  int32_t adc_val5;
  int32_t adc_val6;
  int32_t adc_val7;
  int32_t adc_val8;

  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready

  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x67);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);

  //This is the reading in the Data register from whatever the mux was set
to the last
```

```
  //time this function was called. By default, it is configured to leave
that value at 0
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val1 <<= 8;
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val1 <<= 8;
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);

  //now change the mux register back to 0 so we left things how we found
them
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x67);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  //this should now be the value from the mux change we just did (0,1 to
2,3)
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val2 <<= 8;
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val2 <<= 8;
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(5);
  //this is the value for the

  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val1 > 0x7fffff) { //if MSB == 1
    adc_val1 = adc_val1 - 16777216; //do 2's complement, keep the sign this
time!
  }
```

```
  if (adc_val2 > 0x7fffff) { //if MSB == 1
    adc_val2 = adc_val2 - 16777216; //do 2's complement, keep the sign this
time!
  }

  val1 = adc_val1;

  val2 = adc_val2;

  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready

  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x45);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);

  //This is the reading in the Data register from whatever the mux was set
to the last
  //time this function was called. By default, it is configured to leave
that value at 0
  adc_val3 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val3 <<= 8;
  adc_val3 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val3 <<= 8;
  adc_val3 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);
```

```
  //now change the mux register back to 0 so we left things how we found
them
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x45);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  //this should now be the value from the mux change we just did (0,1 to
2,3)
  adc_val4 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val4 <<= 8;
  adc_val4 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val4 <<= 8;
  adc_val4 |= SPI.transfer(NOP);
  //delayMicroseconds(5);
  //this is the value for the

  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val3 > 0x7fffff) { //if MSB == 1
    adc_val1 = adc_val3 - 16777216; //do 2's complement, keep the sign this
time!
  }

  if (adc_val4 > 0x7fffff) { //if MSB == 1
    adc_val2 = adc_val4 - 16777216; //do 2's complement, keep the sign this
time!
  }

  val3 = adc_val3;

  val4 = adc_val4;

  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready
```

```
  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x23);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);

  //This is the reading in the Data register from whatever the mux was set
to the last
  //time this function was called. By default, it is configured to leave
that value at 0
  adc_val5 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val5 <<= 8;
  adc_val5 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val5 <<= 8;
  adc_val5 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);

  //now change the mux register back to 0 so we left things how we found
them
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x23);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
```

```
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  //this should now be the value from the mux change we just did (0,1 to
2,3)
  adc_val6 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val6 <<= 8;
  adc_val6 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val6 <<= 8;
  adc_val6 |= SPI.transfer(NOP);
  //delayMicroseconds(5);
  //this is the value for the

  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val5 > 0x7fffff) { //if MSB == 1
    adc_val5 = adc_val5 - 16777216; //do 2's complement, keep the sign this
time!
  }

  if (adc_val2 > 0x7fffff) { //if MSB == 1
    adc_val6 = adc_val6 - 16777216; //do 2's complement, keep the sign this
time!
  }

  val5 = adc_val5;

  val6 = adc_val6;

  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready

  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(MUX_RESET);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
```

```
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);

  //This is the reading in the Data register from whatever the mux was set
to the last
  //time this function was called. By default, it is configured to leave
that value at 0
  adc_val7 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val7 <<= 8;
  adc_val7 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val7 <<= 8;
  adc_val7 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);

  //now change the mux register back to 0 so we left things how we found
them
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(MUX_RESET);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(3);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  //this should now be the value from the mux change we just did (0,1 to
2,3)
  adc_val8 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val8 <<= 8;
  adc_val8 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val8 <<= 8;
  adc_val8 |= SPI.transfer(NOP);
```

```
  //delayMicroseconds(5);
  //this is the value for the

  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val7 > 0x7fffff) { //if MSB == 1
    adc_val7 = adc_val7 - 16777216; //do 2's complement, keep the sign this
time!
  }

  if (adc_val2 > 0x7fffff) { //if MSB == 1
    adc_val8 = adc_val8 - 16777216; //do 2's complement, keep the sign this
time!
  }

  val7 = adc_val7;

  val8 = adc_val8;
}

void read_three_values() {
  //datasheet page 21 at the bottom gives the timing
  int32_t adc_val1;
  int32_t adc_val2;
  int32_t adc_val3;

  waitforDRDY(); // Wait until DRDY is LOW
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1));
  digitalWriteFast(10, LOW); //Pull SS Low to Enable Communications with
ADS1247
  //delayMicroseconds(5); // RD: Wait 25ns for ADC12xx to get ready

  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x23);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(2);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
```

```
  //This is the reading in the Data register from whatever the mux was set
to to the last
  //time this function was called. By default, it is configured to leave
that value at 0
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val1 <<= 8;
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val1 <<= 8;
  adc_val1 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);

  //now change the mux register
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(0x45);     //pins registers 4 and 5

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(2);
  SPI.transfer(SYNC);

  //again delay by t11
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);

  //this is the reading from pins 2,3
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val2 <<= 8;
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val2 <<= 8;
  adc_val2 |= SPI.transfer(NOP);
  //delayMicroseconds(5);

  //now wait for the next dataready
  waitforDRDY(); // Wait until DRDY is LOW
  //delayMicroseconds(5);

  //now change the mux register back to 0 so we left things how we found
them
```

```
  SPI.transfer(WREG | MUX); // send 1st command byte, address of the
register
  SPI.transfer(0x00);   // send 2nd command byte, write only one register
  SPI.transfer(MUX_RESET);     //pins registers 2 and 3

  //now we need to sync
  //need to delay by 4x SPI clock = 2.35 uS (t1)
  //to be safe 5 uS
  delayMicroseconds(2);
  SPI.transfer(SYNC);

  //again delay by t1
  delayMicroseconds(5);
  //send wakeup
  SPI.transfer(WAKEUP);

  //then delay one more time by t1 before rdata
  delayMicroseconds(1);

  SPI.transfer(RDATA); //Issue RDATA
  delayMicroseconds(7);
  //this should now be the value from the mux change we just did (2,3 to
4,5)
  adc_val3 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val3 <<= 8;
  adc_val3 |= SPI.transfer(NOP);
  //delayMicroseconds(10);
  adc_val3 <<= 8;
  adc_val3 |= SPI.transfer(NOP);
 // delayMicroseconds(5);
  //this is the value for the

  digitalWriteFast(10, HIGH);
  SPI.endTransaction();

  if (adc_val1 > 0x7fffff) { //if MSB == 1
    adc_val1 = adc_val1 - 16777216; //do 2's complement, keep the sign this
time!
  }

  if (adc_val2 > 0x7fffff) { //if MSB == 1
    adc_val2 = adc_val2 - 16777216; //do 2's complement, keep the sign this
time!
  }

  if (adc_val3 > 0x7fffff) { //if MSB == 1
    adc_val3 = adc_val3 - 16777216; //do 2's complement, keep the sign this
time!
  }


  val1 = adc_val1;

  val2 = adc_val2;

  val3 = adc_val3;
```

```
}


//library files

volatile int DRDY_state = HIGH;

void waitforDRDY() {
  while (DRDY_state) {
    continue;
  }
  noInterrupts();
  DRDY_state = HIGH;
  interrupts();
}

//Interrupt function
void DRDY_Interuppt() {
  DRDY_state = LOW;
}

long GetRegisterValue(uint8_t regAdress) {
  uint8_t bufr;
  digitalWriteFast(10, LOW);
  delayMicroseconds(10);
  SPI.transfer(RREG | regAdress); // send 1st command byte, address of the
register
  SPI.transfer(0x00);     // send 2nd command byte, read only one register
  delayMicroseconds(10);
  bufr = SPI.transfer(NOP); // read data of the register
  delayMicroseconds(10);
  digitalWriteFast(10, HIGH);
  //digitalWrite(_START, LOW);
  SPI.endTransaction();
  return bufr;

}

void SendCMD(uint8_t cmd) {
  waitforDRDY();
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1)); //
initialize SPI with 4Mhz clock, MSB first, SPI Mode0
  digitalWriteFast(10, LOW);
  delayMicroseconds(10);
  SPI.transfer(cmd);
  delayMicroseconds(10);
  digitalWriteFast(10, HIGH);
  SPI.endTransaction();
}

void Reset() {
  SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1)); //
initialize SPI with  clock, MSB first, SPI Mode1
  digitalWriteFast(10, LOW);
  delayMicroseconds(10);
  SPI.transfer(RESET); //Reset
  delay(2); //Minimum 0.6ms required for Reset to finish.
```

```
  SPI.transfer(SDATAC); //Issue SDATAC
  delayMicroseconds(100);
  digitalWriteFast(10, HIGH);
  SPI.endTransaction();
}

void SetRegisterValue(uint8_t regAdress, uint8_t regValue) {

  uint8_t regValuePre = GetRegisterValue(regAdress);
  if (regValue != regValuePre) {
    //digitalWrite(_START, HIGH);
    delayMicroseconds(10);
    waitforDRDY();
    SPI.beginTransaction(SPISettings(SPI_SPEED, MSBFIRST, SPI_MODE1)); //
initialize SPI with SPI_SPEED, MSB first, SPI Mode1
    digitalWriteFast(10, LOW);
    delayMicroseconds(10);
    SPI.transfer(WREG | regAdress); // send 1st command byte, address of the
register
    SPI.transfer(0x00);   // send 2nd command byte, write only one register
    SPI.transfer(regValue);         // write data (1 Byte) for the register
    delayMicroseconds(10);
    digitalWriteFast(10, HIGH);
    //digitalWrite(_START, LOW);
    if (regValue != GetRegisterValue(regAdress)) {   //Check if write was
succesfull
      Serial.print("Write to Register 0x");
      Serial.print(regAdress, HEX);
      Serial.println(" failed!");
    }
    else {
      Serial.println("success");
    }
    SPI.endTransaction();

  }

}
```

Appendix E – Python serial storage code

```
import serial
import csv
from datetime import datetime

#Open a csv file and set it up to receive comma delimited input

#with open('logging.csv', 'w', newline='') as logging:
#    writer = csv.writer(logging, delimiter=",", escapechar=' ',
quoting=csv.QUOTE_NONE)

logging = open('20Hz700mA100dmg01-05.csv', newline='', mode='a')
writer = csv.writer(logging, delimiter=",", escapechar=' ',
quoting=csv.QUOTE_NONE)
```

```
#Open a serial port that is connected to an Arduino (below is Linux, Windows
and Mac would be "COM4" or similar)
#No timeout specified; program will wait until all serial data is received
from Arduino
#Port description will vary according to operating system. Linux will be in
the form /dev/ttyXXXX
#Windows and MAC will be COMX
ser = serial.Serial('COM4', 921600)
ser.flushInput()

#Write out a single character encoded in utf-8; this is defalt encoding for
Arduino serial comms
#This character tells the Arduino to start sending data
ser.write(bytes('x', 'utf-8'))


#while True:
    #Read in data from Serial until \n (new line) received
    #Retreive current time
c = datetime.now()
current_time = c.strftime('%H:%M:%S')

for x in range(15000):
    ser_bytes = ser.readline()
    #print(ser_bytes)

    #Convert received bytes to text format
    decoded_bytes = (ser_bytes[0:len(ser_bytes)-2].decode("utf-8"))
    #print(decoded_bytes)
    #If Arduino has sent a string "stop", exit loop
    #if (decoded_bytes == "stop"):
        #break
    print(decoded_bytes)
    #Write received data to CSV file
    writer.writerow([decoded_bytes])
print(current_time)
c = datetime.now()
current_time = c.strftime('%H:%M:%S')
print(current_time)
# Close port and CSV file to exit
ser.close()
logging.close()
print("logging finished")
```

Appendix F – Python CSV modification code

```
import csv
import numpy as np
import re

combinations1 = ["20", "20", "20", "15", "25", "30"]
combinations2 = ["700", "500", "300", "300", "300", "300"]
point1 = ["01", "05", "09", "13"]
point2 = ["05", "09", "13", "17"]
damage = ["33", "66", "100"]
```

```
def Average(lst):
    return sum(lst) / len(lst)

def Rotright(pointloc, quant):
    pointint = int(pointloc)
    if (pointloc >= 1) and (pointloc <= 4) and ((pointloc + quant) > 4):
        print(pointloc, "line18")
        print(quant, "line19")
        print(pointloc - (4 - quant), "line20")
        return (pointloc - (4 - quant))
    elif (pointloc >= 5) and (pointloc <= 8) and ((pointloc + quant) > 8):
        print(pointloc, "line23")
        print(quant, "line24")
        print(pointloc - (4 - quant), "line25")
        return (pointloc - (4 - quant))
    elif (pointloc >= 9) and (pointloc <= 12) and ((pointloc + quant) > 12):
        return (pointloc - (4 - quant))
    elif (pointloc >= 13) and (pointloc <= 16) and ((pointloc + quant) >
16):
        return (pointloc - (4 - quant))
    elif (pointloc >= 17) and (pointloc) <= 20 and ((pointloc + quant) >
20):
        return (pointloc - (4 - quant))
    else:
        return (pointloc + quant)

def Rotleft(pointloc, quant):
    pointint = int(pointloc)

    if((pointint % 4) - quant < 1):
        return (pointloc + (4 - quant))
    else:
        return (pointloc - quant)

def Mirror(pointloc):
    if(pointloc % 4 == 2):
        return (pointloc + 2)
    if(pointloc % 4 == 0):
        return (pointloc - 2)
    else:
        return pointloc

#begin column conversion

for i in range(6):
    for j in range(3):
        for k in range(4):
            loadcell1 = []
            loadcell2 = []
            loadcell3 = []
            loadcell4 = []
            piezo1 = []
            piezo2 = []
            piezo3 = []
            piezo4 = []
```

```
            piezo5 = []
            piezo6 = []
            piezo7 = []
            piezo8 = []
            filename = combinations1[i] + "Hz" + combinations2[i] + "mA" +
damage[j] + "dmg" + point1[k] + "-" + point2[k] + ".csv"
            print(filename)
            logging = open(filename)
            reader = csv.reader(logging)
            for row in reader:
                csvdata = row
                #print(row[0])
                loadcell1.append(float(row[0].replace(' ', '')))
                #print(row[1])
                loadcell2.append(float(row[1].replace(' ', '')))
                #print(row[2])
                loadcell3.append(float(row[2].replace(' ', '')))
                #print(row[3])
                loadcell4.append(float(row[3].replace(' ', '')))
                #print(row[4])
                piezo1.append(float(row[4].replace(' ', '')))
                #print(row[5])
                piezo2.append(float(row[5].replace(' ', '')))
                #print(row[6])
                piezo3.append(float(row[6].replace(' ', '')))
                #print(row[7])
                piezo4.append(float(row[7].replace(' ', '')))
                #print(row[8])
                piezo5.append(float(row[8].replace(' ', '')))
                #print(row[9])
                piezo6.append(float(row[9].replace(' ', '')))
                #print(row[10])
                piezo7.append(float(row[10].replace(' ', '')))
                #print(row[11])
                piezo8.append(float(row[11].replace(' ', '')))
            for l in range(11):
                loadcell1temp = loadcell1[(l * 1250):((l + 2) * 1250)]
                loadcell2temp = loadcell2[(l * 1250):((l + 2) * 1250)]
                loadcell3temp = loadcell3[(l * 1250):((l + 2) * 1250)]
                loadcell4temp = loadcell4[(l * 1250):((l + 2) * 1250)]
                piezo1temp = piezo1[(l * 1250):((l + 2) * 1250)]
                piezo2temp = piezo2[(l * 1250):((l + 2) * 1250)]
                piezo3temp = piezo3[(l * 1250):((l + 2) * 1250)]
                piezo4temp = piezo4[(l * 1250):((l + 2) * 1250)]
                piezo5temp = piezo5[(l * 1250):((l + 2) * 1250)]
                piezo6temp = piezo6[(l * 1250):((l + 2) * 1250)]
                piezo7temp = piezo7[(l * 1250):((l + 2) * 1250)]
                piezo8temp = piezo8[(l * 1250):((l + 2) * 1250)]
                loadcell1avg = Average(loadcell1temp)
                loadcell2avg = Average(loadcell2temp)
                loadcell3avg = Average(loadcell3temp)
                loadcell4avg = Average(loadcell4temp)
                piezo1avg = Average(piezo1temp)
                piezo2avg = Average(piezo2temp)
                piezo3avg = Average(piezo3temp)
                piezo4avg = Average(piezo4temp)
                piezo5avg = Average(piezo5temp)
```

```
                piezo6avg = Average(piezo6temp)
                piezo7avg = Average(piezo7temp)
                piezo8avg = Average(piezo8temp)
                loadcell1temp = [x - loadcell1avg for x in loadcell1temp]
                loadcell2temp = [x - loadcell2avg for x in loadcell2temp]
                loadcell3temp = [x - loadcell3avg for x in loadcell3temp]
                loadcell4temp = [x - loadcell4avg for x in loadcell4temp]
                piezo1temp = [x - piezo1avg for x in piezo1temp]
                piezo2temp = [x - piezo2avg for x in piezo2temp]
                piezo3temp = [x - piezo3avg for x in piezo3temp]
                piezo4temp = [x - piezo4avg for x in piezo4temp]
                piezo5temp = [x - piezo5avg for x in piezo5temp]
                piezo6temp = [x - piezo6avg for x in piezo6temp]
                piezo7temp = [x - piezo7avg for x in piezo7temp]
                piezo8temp = [x - piezo8avg for x in piezo8temp]
                loadcell1max = max(map(abs, loadcell1temp))
                loadcell2max = max(map(abs, loadcell2temp))
                loadcell3max = max(map(abs, loadcell3temp))
                loadcell4max = max(map(abs, loadcell4temp))
                piezo1max = max(map(abs, piezo1temp))
                piezo2max = max(map(abs, piezo2temp))
                piezo3max = max(map(abs, piezo3temp))
                piezo4max = max(map(abs, piezo4temp))
                piezo5max = max(map(abs, piezo5temp))
                piezo6max = max(map(abs, piezo6temp))
                piezo7max = max(map(abs, piezo7temp))
                piezo8max = max(map(abs, piezo8temp))
                loadcell1temp = [x / loadcell1max for x in loadcell1temp]
                loadcell2temp = [x / loadcell2max for x in loadcell2temp]
                loadcell3temp = [x / loadcell3max for x in loadcell3temp]
                loadcell4temp = [x / loadcell4max for x in loadcell4temp]
                piezo1temp = [x / piezo1max for x in piezo1temp]
                piezo2temp = [x / piezo2max for x in piezo2temp]
                piezo3temp = [x / piezo3max for x in piezo3temp]
                piezo4temp = [x / piezo4max for x in piezo4temp]
                piezo5temp = [x / piezo5max for x in piezo5temp]
                piezo6temp = [x / piezo6max for x in piezo6temp]
                piezo7temp = [x / piezo7max for x in piezo7temp]
                piezo8temp = [x / piezo8max for x in piezo8temp]
                outputfilename = combinations1[i] + "Hz" + combinations2[i]
+ "mA" + damage[j] + "dmg" + point1[k] + "-" + point2[k] + "set" + str(l +
1) + ".csv"
                outputfile = open(outputfilename, newline='', mode='a')
                writer = csv.writer(outputfile, delimiter=",",
quoting=csv.QUOTE_NONE)
                writer.writerow(["time", "loadcell1", "loadcell2",
"loadcell3", "loadcell4", "piezo1", "piezo2", "piezo3", "piezo4", "piezo5",
"piezo6", "piezo7", "piezo8", "frequency", "amplitude", "point01",
"point02", "point03", "point04", "point05", "point06", "point07", "point08",
"point09", "point10", "point11", "point12", "point13", "point14", "point15",
"point16", "point17", "point18", "point19", "point20"])
                points = [0] * 20
                points[int(point1[k]) - 1] = int(damage[j])
                points[int(point2[k]) - 1] = int(damage[j])
                for m in range(2500):
                    time = m * (1/250)
```

```
                    writer.writerow([time, loadcell1temp[m],
loadcell2temp[m], loadcell3temp[m], loadcell4temp[m], piezo1temp[m],
piezo2temp[m], piezo3temp[m], piezo4temp[m], piezo5temp[m], piezo6temp[m],
piezo7temp[m], piezo8temp[m], combinations1[i], combinations2[i]] + points)



#Begin crossbeam conversion

point1 = ["01", "05", "09", "13"]
point2 = ["06", "10", "14", "18"]
damage = ["50", "100"]

for i in range(6):
    for j in range(2):
        for k in range(4):
            loadcell1 = []
            loadcell2 = []
            loadcell3 = []
            loadcell4 = []
            piezo1 = []
            piezo2 = []
            piezo3 = []
            piezo4 = []
            piezo5 = []
            piezo6 = []
            piezo7 = []
            piezo8 = []
            filename = combinations1[i] + "Hz" + combinations2[i] + "mA" +
damage[j] + "dmg" + point1[k] + "-" + point2[k] + ".csv"
            print(filename)
            logging = open(filename)
            reader = csv.reader(logging)
            for row in reader:
                csvdata = row
                #print(row[0])
                loadcell1.append(float(row[0].replace(' ', '')))
                #print(row[1])
                loadcell2.append(float(row[1].replace(' ', '')))
                #print(row[2])
                loadcell3.append(float(row[2].replace(' ', '')))
                #print(row[3])
                loadcell4.append(float(row[3].replace(' ', '')))
                #print(row[4])
                piezo1.append(float(row[4].replace(' ', '')))
                #print(row[5])
                piezo2.append(float(row[5].replace(' ', '')))
                #print(row[6])
                piezo3.append(float(row[6].replace(' ', '')))
                #print(row[7])
                piezo4.append(float(row[7].replace(' ', '')))
                #print(row[8])
                piezo5.append(float(row[8].replace(' ', '')))
                #print(row[9])
                piezo6.append(float(row[9].replace(' ', '')))
                #print(row[10])
                piezo7.append(float(row[10].replace(' ', '')))
```

```
        #print(row[11])
        piezo8.append(float(row[11].replace(' ', '')))
    for l in range(11):
        loadcell1temp = loadcell1[(l * 1250):((l + 2) * 1250)]
        loadcell2temp = loadcell2[(l * 1250):((l + 2) * 1250)]
        loadcell3temp = loadcell3[(l * 1250):((l + 2) * 1250)]
        loadcell4temp = loadcell4[(l * 1250):((l + 2) * 1250)]
        piezo1temp = piezo1[(l * 1250):((l + 2) * 1250)]
        piezo2temp = piezo2[(l * 1250):((l + 2) * 1250)]
        piezo3temp = piezo3[(l * 1250):((l + 2) * 1250)]
        piezo4temp = piezo4[(l * 1250):((l + 2) * 1250)]
        piezo5temp = piezo5[(l * 1250):((l + 2) * 1250)]
        piezo6temp = piezo6[(l * 1250):((l + 2) * 1250)]
        piezo7temp = piezo7[(l * 1250):((l + 2) * 1250)]
        piezo8temp = piezo8[(l * 1250):((l + 2) * 1250)]
        loadcell1avg = Average(loadcell1temp)
        loadcell2avg = Average(loadcell2temp)
        loadcell3avg = Average(loadcell3temp)
        loadcell4avg = Average(loadcell4temp)
        piezo1avg = Average(piezo1temp)
        piezo2avg = Average(piezo2temp)
        piezo3avg = Average(piezo3temp)
        piezo4avg = Average(piezo4temp)
        piezo5avg = Average(piezo5temp)
        piezo6avg = Average(piezo6temp)
        piezo7avg = Average(piezo7temp)
        piezo8avg = Average(piezo8temp)
        loadcell1temp = [x - loadcell1avg for x in loadcell1temp]
        loadcell2temp = [x - loadcell2avg for x in loadcell2temp]
        loadcell3temp = [x - loadcell3avg for x in loadcell3temp]
        loadcell4temp = [x - loadcell4avg for x in loadcell4temp]
        piezo1temp = [x - piezo1avg for x in piezo1temp]
        piezo2temp = [x - piezo2avg for x in piezo2temp]
        piezo3temp = [x - piezo3avg for x in piezo3temp]
        piezo4temp = [x - piezo4avg for x in piezo4temp]
        piezo5temp = [x - piezo5avg for x in piezo5temp]
        piezo6temp = [x - piezo6avg for x in piezo6temp]
        piezo7temp = [x - piezo7avg for x in piezo7temp]
        piezo8temp = [x - piezo8avg for x in piezo8temp]
        loadcell1max = max(map(abs, loadcell1temp))
        loadcell2max = max(map(abs, loadcell2temp))
        loadcell3max = max(map(abs, loadcell3temp))
        loadcell4max = max(map(abs, loadcell4temp))
        piezo1max = max(map(abs, piezo1temp))
        piezo2max = max(map(abs, piezo2temp))
        piezo3max = max(map(abs, piezo3temp))
        piezo4max = max(map(abs, piezo4temp))
        piezo5max = max(map(abs, piezo5temp))
        piezo6max = max(map(abs, piezo6temp))
        piezo7max = max(map(abs, piezo7temp))
        piezo8max = max(map(abs, piezo8temp))
        loadcell1temp = [x / loadcell1max for x in loadcell1temp]
        loadcell2temp = [x / loadcell2max for x in loadcell2temp]
        loadcell3temp = [x / loadcell3max for x in loadcell3temp]
        loadcell4temp = [x / loadcell4max for x in loadcell4temp]
        piezo1temp = [x / piezo1max for x in piezo1temp]
        piezo2temp = [x / piezo2max for x in piezo2temp]
```

```
                piezo3temp = [x / piezo3max for x in piezo3temp]
                piezo4temp = [x / piezo4max for x in piezo4temp]
                piezo5temp = [x / piezo5max for x in piezo5temp]
                piezo6temp = [x / piezo6max for x in piezo6temp]
                piezo7temp = [x / piezo7max for x in piezo7temp]
                piezo8temp = [x / piezo8max for x in piezo8temp]

                outputfilename = combinations1[i] + "Hz" + combinations2[i]
+ "mA" + damage[j] + "dmg" + point1[k] + "-" + point2[k] + "set" + str(l +
1) + ".csv"
                outputfile = open(outputfilename, newline='', mode='a')
                writer = csv.writer(outputfile, delimiter=",",
quoting=csv.QUOTE_NONE)
                writer.writerow(["time", "loadcell1", "loadcell2",
"loadcell3", "loadcell4", "piezo1", "piezo2", "piezo3", "piezo4", "piezo5",
"piezo6", "piezo7", "piezo8", "frequency", "amplitude", "point01",
"point02", "point03", "point04", "point05", "point06", "point07", "point08",
"point09", "point10", "point11", "point12", "point13", "point14", "point15",
"point16", "point17", "point18", "point19", "point20"])
                points = [0] * 20
                points[int(point1[k]) - 1] = int(damage[j])
                points[int(point2[k]) - 1] = int(damage[j])
                for m in range(2500):
                    time = m * (1/250)
                    writer.writerow([time, loadcell1temp[m],
loadcell2temp[m], loadcell3temp[m], loadcell4temp[m], piezo1temp[m],
piezo2temp[m], piezo3temp[m], piezo4temp[m], piezo5temp[m], piezo6temp[m],
piezo7temp[m], piezo8temp[m], combinations1[i], combinations2[i]] + points)

#begin 0 dmg

for i in range(6):
    for j in range(1):
        for k in range(1):
            loadcell1 = []
            loadcell2 = []
            loadcell3 = []
            loadcell4 = []
            piezo1 = []
            piezo2 = []
            piezo3 = []
            piezo4 = []
            piezo5 = []
            piezo6 = []
            piezo7 = []
            piezo8 = []
            filename = combinations1[i] + "Hz" + combinations2[i] +
"mA0dmg.csv"
            print(filename)
            logging = open(filename)
            reader = csv.reader(logging)
            for row in reader:
                csvdata = row
                #print(row[0])
                loadcell1.append(float(row[0].replace(' ', '')))
                #print(row[1])
                loadcell2.append(float(row[1].replace(' ', '')))
```

```
        #print(row[2])
        loadcell3.append(float(row[2].replace(' ', '')))
        #print(row[3])
        loadcell4.append(float(row[3].replace(' ', '')))
        #print(row[4])
        piezo1.append(float(row[4].replace(' ', '')))
        #print(row[5])
        piezo2.append(float(row[5].replace(' ', '')))
        #print(row[6])
        piezo3.append(float(row[6].replace(' ', '')))
        #print(row[7])
        piezo4.append(float(row[7].replace(' ', '')))
        #print(row[8])
        piezo5.append(float(row[8].replace(' ', '')))
        #print(row[9])
        piezo6.append(float(row[9].replace(' ', '')))
        #print(row[10])
        piezo7.append(float(row[10].replace(' ', '')))
        #print(row[11])
        piezo8.append(float(row[11].replace(' ', '')))
    for l in range(11):
        loadcell1temp = loadcell1[(l * 1250):((l + 2) * 1250)]
        loadcell2temp = loadcell2[(l * 1250):((l + 2) * 1250)]
        loadcell3temp = loadcell3[(l * 1250):((l + 2) * 1250)]
        loadcell4temp = loadcell4[(l * 1250):((l + 2) * 1250)]
        piezo1temp = piezo1[(l * 1250):((l + 2) * 1250)]
        piezo2temp = piezo2[(l * 1250):((l + 2) * 1250)]
        piezo3temp = piezo3[(l * 1250):((l + 2) * 1250)]
        piezo4temp = piezo4[(l * 1250):((l + 2) * 1250)]
        piezo5temp = piezo5[(l * 1250):((l + 2) * 1250)]
        piezo6temp = piezo6[(l * 1250):((l + 2) * 1250)]
        piezo7temp = piezo7[(l * 1250):((l + 2) * 1250)]
        piezo8temp = piezo8[(l * 1250):((l + 2) * 1250)]
        loadcell1avg = Average(loadcell1temp)
        loadcell2avg = Average(loadcell2temp)
        loadcell3avg = Average(loadcell3temp)
        loadcell4avg = Average(loadcell4temp)
        piezo1avg = Average(piezo1temp)
        piezo2avg = Average(piezo2temp)
        piezo3avg = Average(piezo3temp)
        piezo4avg = Average(piezo4temp)
        piezo5avg = Average(piezo5temp)
        piezo6avg = Average(piezo6temp)
        piezo7avg = Average(piezo7temp)
        piezo8avg = Average(piezo8temp)
        loadcell1temp = [x - loadcell1avg for x in loadcell1temp]
        loadcell2temp = [x - loadcell2avg for x in loadcell2temp]
        loadcell3temp = [x - loadcell3avg for x in loadcell3temp]
        loadcell4temp = [x - loadcell4avg for x in loadcell4temp]
        piezo1temp = [x - piezo1avg for x in piezo1temp]
        piezo2temp = [x - piezo2avg for x in piezo2temp]
        piezo3temp = [x - piezo3avg for x in piezo3temp]
        piezo4temp = [x - piezo4avg for x in piezo4temp]
        piezo5temp = [x - piezo5avg for x in piezo5temp]
        piezo6temp = [x - piezo6avg for x in piezo6temp]
        piezo7temp = [x - piezo7avg for x in piezo7temp]
        piezo8temp = [x - piezo8avg for x in piezo8temp]
```

```
                loadcell1max = max(map(abs, loadcell1temp))
                loadcell2max = max(map(abs, loadcell2temp))
                loadcell3max = max(map(abs, loadcell3temp))
                loadcell4max = max(map(abs, loadcell4temp))
                piezo1max = max(map(abs, piezo1temp))
                piezo2max = max(map(abs, piezo2temp))
                piezo3max = max(map(abs, piezo3temp))
                piezo4max = max(map(abs, piezo4temp))
                piezo5max = max(map(abs, piezo5temp))
                piezo6max = max(map(abs, piezo6temp))
                piezo7max = max(map(abs, piezo7temp))
                piezo8max = max(map(abs, piezo8temp))
                loadcell1temp = [x / loadcell1max for x in loadcell1temp]
                loadcell2temp = [x / loadcell2max for x in loadcell2temp]
                loadcell3temp = [x / loadcell3max for x in loadcell3temp]
                loadcell4temp = [x / loadcell4max for x in loadcell4temp]
                piezo1temp = [x / piezo1max for x in piezo1temp]
                piezo2temp = [x / piezo2max for x in piezo2temp]
                piezo3temp = [x / piezo3max for x in piezo3temp]
                piezo4temp = [x / piezo4max for x in piezo4temp]
                piezo5temp = [x / piezo5max for x in piezo5temp]
                piezo6temp = [x / piezo6max for x in piezo6temp]
                piezo7temp = [x / piezo7max for x in piezo7temp]
                piezo8temp = [x / piezo8max for x in piezo8temp]


                outputfilename = combinations1[i] + "Hz" + combinations2[i]
+ "mA0dmgset" + str(l + 1) + ".csv"
                outputfile = open(outputfilename, newline='', mode='a')
                writer = csv.writer(outputfile, delimiter=",",
quoting=csv.QUOTE_NONE)
                writer.writerow(["time", "loadcell1", "loadcell2",
"loadcell3", "loadcell4", "piezo1", "piezo2", "piezo3", "piezo4", "piezo5",
"piezo6", "piezo7", "piezo8", "frequency", "amplitude", "point01",
"point02", "point03", "point04", "point05", "point06", "point07", "point08",
"point09", "point10", "point11", "point12", "point13", "point14", "point15",
"point16", "point17", "point18", "point19", "point20"])
                points = [0] * 20
                for m in range(2500):
                    time = m * (1/250)
                    writer.writerow([time, loadcell1temp[m],
loadcell2temp[m], loadcell3temp[m], loadcell4temp[m], piezo1temp[m],
piezo2temp[m], piezo3temp[m], piezo4temp[m], piezo5temp[m], piezo6temp[m],
piezo7temp[m], piezo8temp[m], combinations1[i], combinations2[i]] + points)
```

Appendix G – MATLAB plot generation code

```
Fs = 2500;
Fsdown = Fs/5;
T = 1/Fs;
L = 60*Fs;
t = (0:L-1)*T;
t2 = 0:1/Fs:2.96;
timedata = csvread('mid15Hz0dmg.csv',4,1,[4 1 150003 1]);
timedata21 = csvread('mid15Hz50dmg01-17.csv',4,1,[4 1 150003 1]);
```

```
timedata31 = csvread('mid15Hz50dmg02-03.csv',4,1,[4 1 150003 1]);
timedata12 = csvread('mid15Hz0dmg.csv',4,3,[4 3 150003 3]);
timedata22 = csvread('mid15Hz50dmg01-17.csv',4,3,[4 3 150003 3]);
timedata32 = csvread('mid15Hz50dmg02-03.csv',4,3,[4 3 150003 3]);
%timedata2 = csvread('20max100damage.csv',4,1,[4 1 150003 1]);
%timedatadown = downsample(timedata,5)
avg = mean(timedata);
timedata = timedata - avg;
avg = mean(timedata21);
timedata21 = timedata21 - avg;
fftdata = fft(timedata);
spectrodata = spectrogram(timedata);
stftdata = stft(timedata);
N = length(timedata);
nfft = 2^nextpow2(length(timedata));
Pxx = abs(fft(timedata,nfft)).^2/length(timedata)/Fs;
Hpsd = dspdata.psd(Pxx(1:length(Pxx)/2),'Fs',Fs);
%[c,diff] = xcorr(timedata,timedata2);
%xdft = fftdata(1:N/2+1);
%psddata = (1/(Fs*N)) * abs(xdft).^2;
%psdx(2:end-1) = 2*psdx(2:end-1);
%freq = 0:Fs/length(x):Fs/2;

%figure()

%plot(1000*t,timedata)
%xlim([0 500])
%ylim([-1 1])
%xlabel("t (milliseconds)")
%ylabel("X(t)")

%figure()

%plot(Fs/L*(0:L-1),abs(fftdata),"LineWidth",3)
%title("Complex Magnitude of fft Spectrum")
%xlim([0 200])
%ylim([0 17500])
%xlabel("f (Hz)")
%ylabel("|fft(X)|")

%figure()
%spectrogram(timedata,'yaxis')

subplot(2,3,1)

[s,f,t] =
stft(timedata,Fs,Window=kaiser(256,5),OverlapLength=220,FFTLength=512);

sdb = mag2db(abs(s));
mesh(t,f/1000,sdb);
colormap gray
ylim([-1 1])

cc = max(sdb(:))+[-60 0];
ax = gca;
ax.CLim = cc;
view(2)
```

```
xlabel('No Damage')
ylabel('Sensor 1')
set(gca, 'XAxisLocation','top','xtick',[],'ytick',[])

subplot(2,3,2)

[s,f,t] =
stft(timedata21,Fs,Window=kaiser(256,5),OverlapLength=220,FFTLength=512);

sdb = mag2db(abs(s));
mesh(t,f/1000,sdb);
colormap gray
ylim([-1 1])

cc = max(sdb(:))+[-60 0];
ax = gca;
ax.CLim = cc;
view(2)

xlabel('Crossbeam Damage')
set(gca, 'XAxisLocation','top','xtick',[],'ytick',[])

subplot(2,3,3)

[s,f,t] =
stft(timedata31,Fs,Window=kaiser(256,5),OverlapLength=220,FFTLength=512);

sdb = mag2db(abs(s));
mesh(t,f/1000,sdb);
colormap gray
ylim([-1 1])

cc = max(sdb(:))+[-60 0];
ax = gca;
ax.CLim = cc;
view(2)

xlabel('Column Damage')
ylabel('Frequency (kHz)')
set(gca, 'XAxisLocation','top','xtick',[])
set(gca, 'YAxisLocation','right')

subplot(2,3,4)

[s,f,t] =
stft(timedata12,Fs,Window=kaiser(256,5),OverlapLength=220,FFTLength=512);

sdb = mag2db(abs(s));
mesh(t,f/1000,sdb);
colormap gray
ylim([-1 1])

cc = max(sdb(:))+[-60 0];
ax = gca;
ax.CLim = cc;
view(2)
```

```
xlabel('Seconds')
ylabel('Sensor 2')
set(gca, 'ytick',[])

subplot(2,3,5)

[s,f,t] =
stft(timedata22,Fs,Window=kaiser(256,5),OverlapLength=220,FFTLength=512);

sdb = mag2db(abs(s));
mesh(t,f/1000,sdb);
colormap gray
ylim([-1 1])

cc = max(sdb(:))+[-60 0];
ax = gca;
ax.CLim = cc;
view(2)
xlabel('Seconds')

subplot(2,3,6)

[s,f,t] =
stft(timedata32,Fs,Window=kaiser(256,5),OverlapLength=220,FFTLength=512);

sdb = mag2db(abs(s));
mesh(t,f/1000,sdb);
colormap gray
ylim([-1 1])

cc = max(sdb(:))+[-60 0];
ax = gca;
ax.CLim = cc;
view(2)
xlabel('Seconds')
ylabel('Frequency (kHz)')

set(gca, 'YAxisLocation','right')


sfh1 = subplot(2,3,1);
sfh2 = subplot(2,3,2);
sfh3 = subplot(2,3,3);
sfh4 = subplot(2,3,4);
sfh5 = subplot(2,3,5);
sfh6 = subplot(2,3,6);
sfh1.Position = sfh1.Position + [-0.05 -0.075 0.05 0.05];
sfh2.Position = sfh2.Position + [-0.05 -0.075 0.05 0.05];
sfh3.Position = sfh3.Position + [-0.05 -0.075 0.05 0.05];
sfh4.Position = sfh4.Position + [-0.05 -0.05 0.05 0.075];
sfh5.Position = sfh5.Position + [-0.05 -0.05 0.05 0.075];
sfh6.Position = sfh6.Position + [-0.05 -0.05 0.05 0.075];
set(sfh1,'Color','red')
%sfh1.XAxis.Visible = 'off';
%sfh1.YAxis.Visible = 'off';
%sfh2.YAxis.Visible = 'off';
%sfh2.XAxis.Visible = 'off';
```

```
%sfh3.YAxis.Visible = 'off';
%sfh3.XAxis.Visible = 'off';
%sfh4.YAxis.Visible = 'off';
sfh5.YAxis.Visible = 'off';

%figure()
%plot_data = abs(s);
%surf(t,f,plot_data,'EdgeColor','none');
%xlabel('Time');
%ylabel('Frequency');
%zlabel('Amplitude');
%title('Spectrogram');

%periodogram(timedata,rectwin(N),N,Fs)
%Hpsd = dspdata.psd(Data)
%figure()
%plot(Hpsd)

%figure()
%cwt(timedata,Fs)
%figure()
%stem(diff,c)

%fftdata
```