Development of a low-cost solution for remote Structural Health Monitoring

FINAL YEAR PROJECT UG 2018

By

Saif Ur Rehman (G.L)	**00000244249**
Muhammad Huzaifa Younus Toor	**00000257578**
Qalandar Ali Hussaini	**00000244205**
Abdul Rafay	**00000264230**

NUST Institute of Civil Engineering (NICE)

School of Civil and Environmental Engineering (SCEE)

National University of Sciences and Technology (NUST), Islamabad, Pakistan

2022

This is to certify that the

Thesis titled

Development of a low-cost solution for remote Structural Health Monitoring

Submitted by

Saif Ur Rehman (G.L)	**00000244249**
Muhammad Huzaifa Younus Toor	**00000257578**
Qalandar Ali Hussaini	**00000244205**
Abdul Rafay	**00000264230**

has been accepted towards the requirements

for the undergraduate degree

in

CIVIL ENGINEERING

Dr. Muhammad Usman
Associate Professor
NUST Institute of Civil Engineering
School of Civil and Environmental Engineering
National University of Sciences and Technology, Islamabad, Pakistan

DECLARATION

We certify that this research work titled "Development of a low-cost solution for remote structural health monitoring" is our work carried out during our study under the supervision of **Dr. Muhammad Usman**.

We assert the statements made and conclusions are drawn are an outcome of our researchwork. We further certify that

- The work contained in the report is original and has been done by us under the general supervision of my supervisor.
- The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or the any other University of Pakistan or abroad.
- We have followed the guidelines provided by the university in writing the report.
- Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the report's text and their details in the references.

Signature of students

Saif Ur Rehman

Muhammad Huzaifa Younus Toor

Qalandar Ali Hussaini

Abdul Rafay

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

In the name of Allah, the Most Beneficent, the Most Merciful as well as peace and blessings upon Prophet Muhammad, His servant, and final messenger.

First, we would like to thank our Creator Allah the Almighty, who has blessed us with the opportunity to perform this research. His guidance through each adversity while carrying out our research work is what has led us to complete and compile our work in time.

We are also greatly thankful to our parents who have supported us through every thick and thin of our life's journey. They have supported us when we were not able to walk, and they are still supporting us to continue and provide solutions for humanity.

From the depths of our hearts, we are obliged to pay special thanks to our supervisor,**Dr. Muhammad Usman**. His guidance throughout the project has been a great learning experience for us and the knowledge he has imparted to us will be one of the primary factors for our growth. He has also helped us in every part of our thesis and above all, his constant efforts to expose our project on a bigger stage will be a key factor to help us dive into the practical world. We feel the responsibility to thank him for his patience, guidance, and cooperation throughout our final year project.

We are also extremely grateful to and value the assistance provided by the entire staff of Structures Lab at NICE. They have been a vital element of our success.

Finally, I, Saif Ur Rehman, the group leader for this project, would like to pay my sincerest gratitude to my dear group members, who took matters into hand, followed my guidance by heart, sacrificed their sleep, and carried out experimental work in scorching heat despite fasting, when the time of adversity came upon me. Lastly, we as a group would like to pay our gratitude to anyone and everyone, who participated knowing and unknowingly, to assist us in our endeavours while carrying out this study.

DEDICATION

To our beloved parents, teachers, mentors, and colleagues who have stood by our side in rough and tough times, and who have taught us to show perseverance in the face of adversity.

ABSTRACT

In the modern world, civil engineering structures are of paramount importance. With the increasing population, the need for the development of new infrastructure is at a historic high. Although this need is being amply met with the modernization of construction techniques, increasing maintenance costs of existing structures is still a critical issue. Early damage detection through continuous health monitoring instead of conventional periodic inspection, is the promising way forward for healthy structures with the least maintenance requirement. As the damage is identified at the initial stage, the cost associated with its repair can be reduced. Considering the immense scale of civil infrastructure, currently available structural health monitoring systems are not feasible for mass application as they are prohibitively expensive. Hence, it is the need of the hour to develop a remote real-time Structural Health Monitoring System (SHMS).

Our proposed system utilized a vibration-based structural health monitoring approach and extracted dynamic parameters from acceleration data. The system consists of a node tasked with data collection and transmission and a server tasked with data post-processing and display. The node comprises the following main hardware components: ESP32 as a microcontroller unit and ADXL345 as an accelerometer. The system was developed with the capability of multiple node integration, onboard data storage to eliminate sampling rate inconsistencies, dual trigger modes to achieve a high signal-to-noise ratio (SNR), power usage optimization to prolong stand-by time, and wireless data transmission along with automated post-processing. To minimize the cost of the system, Internet of Things (IoT) technology was adopted in the development of the system.

Keywords: Low-cost Accelerometers; IoT; SNR; ADXL345; ESP32

Table of Contents

List of Figures 10

Overview of Chapter 12

CHAPTER 1 13

INTRODUCTION 13

1.1 Background 13

1.2 Problem Statement 15

1.3 Objectives 15

CHAPTER 2 16

LITERATURE REVIEW 16

CHAPTER 3 24

THEORETICAL BACKGROUND 24

3.1 Dynamic Sysytem 24

3.2 Damping Force 26

3.3 Natural Frequencies and Modes of Vibration 27

3.5 System Identification 28

3.6 Time Domain Data 28

3.7 Frequency Domain Data 29

3.8 Requirement for Data Quality 29

3.9 Damage Detection 29

3.10 Damage Detection Methods 30

CHAPTER 4 31

METHODOLOGY 31

4.1 Overview 31
4.2 Hardware 31
4.3 Software 33
CHAPTER 5 39
Experimental Validation 39
5.1 Lab scale Validation 39
5.2 Full-scale Validation 41
5.3 Results 43
CHAPTER 6 47
CONCLUSIONS 47
REFERENCES 48
APPENDICES 51

List of Figures

Figure 1 Single Degree of Freedom 24
Figure 2 Methodology Flowchart 31
Figure 3 Node without the power module 33
Figure 4 Node with the power module 33
Figure 5 Traditional approach 34
Figure 6 With FreeRTOS 34
Figure 7 Intercommunication between nodes and communicate with the server 35
Figure 8 Post-processing steps 37
Figure 9 X2-2 Time domain data 40
Figure 10 Node Time Domain Data 40
Figure 11 X2-2 Frequency domain data 40
Figure 12 Node Frequency domain data 40
Figure 13 Beam under monitoring 41
Figure 14 Node mounted on the beam flange 41
Figure 15 Schematic of the room with the dots indicating node locations 41
Figure 16 Acceleration-triggered data set 1 44
Figure 17 Superimposed frequency graph of 3 nodes with fundamental frequencies 45
Figure 18 Graphical result on the website with green baseline 46

.

List of Tables

Table 1 Testing parameters 39
Table 2 Full scale validation parameters 42
Table 3 Collected data samples 43
Table 4 Fundamental Frequencies from 3 data sets 45

OVERVIEW OF CHAPTERS

Chapter 1 is related to the Introduction of the project thesis which contains background, problem statement, and objectives.

Chapter 2 elaborates all the literature review which has been done for this study. To understand Structure Health Monitoring properly all the traditional techniques including non-destructive testing and vibration-based damage detection are studied. To understand specifically Operational Model Analysis and Internet of things (IoT).

Chapter 3 contains the theoretical background of dynamic system and damping force. It also explains the Structure of Health Monitoring methods and methods of system identification. Furthermore, it explains natural frequencies, modes of vibrations, time domain data and frequency domain data.

Chapter 4 contains the methodology of our study which includes the detailed descriptions of our hardware components and software which we were used for post processing.

Chapter 5 consists of experimental validation, results, and discussions.

Chapter 6 consists of conclusions.

CHAPTER 1

INTRODUCTION

1.1 Background:

As the safety requirements are increasing in the regulatory framework for the structures being built and with the advent of new technologies new challenges are being faced for the scientific understanding of existing structures. It is necessary to identify the dynamic behaviour of structures and monitor them over time to ensure structural safety. Structural Health Monitoring has been defined in the literature as the "acquisition, validation and analysis of technical data to facilitate life-cycle management decisions". The change in dynamic properties of structures is gradual and cannot be detectable with the naked eye. We can study Static as well as dynamic properties of structures by using identification techniques. In static system identification, the change in static properties for example deformations, the opening of the cracks, etc. can be observed due to changes in the geometric properties of the structure (Moreu et al., 2018).

In Dynamic Structural Health monitoring we can extract vibration-based modal properties like natural frequencies and model shapes etc. we can also observe changes in these properties due to variations in the physical properties of the structure. Because whenever a change in physical properties occurs, vibration-based properties also change. And by studying these vibrations-based properties we can easily detect damage in structures (Fritzen, 2005).

The main theme for VBDD is that damage and cracks cause variation to the physical properties such as mass, damping, stiffness. Due to these it results in changes in modal properties like frequencies, mode shapes etc. Therefore, it is feasible to judge presence of damage in terms of analysing these modal properties (Fan & Qiao, 2011).

For some huge structures it is very difficult to provide a large, required input force to do realistic forced vibration test. We can detect initial damages with the help of processes for measuring the health of the subject by applying SHM techniques for analysis of dynamic properties of structures. We can apply these techniques either in frequency domain or time domain. Any kind of change in the dynamic properties of a structure can indicate a decrease of its load-carrying capacity (Antunes et al., 2012).

We can perform two kinds of modal analysis for the purpose of Structural Health Monitoring. One type is more advance and has some advantages over other. These two types of model analysis are as follow:

- Traditional Modal Analysis
- Operational Modal Analysis

The traditional modal analysis method uses the change in mode shapes in undamaged condition as well as damage condition of a structure to carry out damage identification. For this purpose, first

we must do experiment on specimen under undamaged condition and then after inducing damage in same specimen testing must be done on damage specimen by keeping same boundary conditions (Wong, 2004).

"Some of basic features of Operational Model Analysis are given below:

- In-situ testing of a structure – True boundaries.
- Natural environment – True excitation forces even in the presence of deterministic signals (harmonics).
- Test during normal service state – No interruption needed - increased productivity.
- Use operational forces - No artificial excitation needed.
- Modal parameter results describe the true service state of the structure.
- Can be used on extremely small or large structures – size does not matter (Harms et al., 2010)."

The structure is fully functional during its monitoring as in OMA structure is subjected to natural excitations or ambient vibrations and it does not require shaker or impact tests to measure the response. For the case of our structure, the loading can be due to wind, occupancy load, or micro-seismic vibrations. As it does not interrupt the functionality so the analysis can be run for longer period to get better response.

The damage assessment falls under the category of structural health monitoring. Structural health monitoring (SHM) deals with observing and analysing changes in the material or geometric qualities that are monitored over time using sampling response measurements. SHM for civil infrastructure is carried out for serviceability and stability. The goal of the SHM paradigm is to assess damage in the structures. And this, as compared to aerospace, mechanical engineering, is constrained by various factors, one of which is the lack of reproducibility i.e., every structure is a unique system. Even two similarly designed structures will not be the same. Therefore, developing a general SHM technique is the need of time, and research is extensively being carried out on this subject.

There are various methods of assessment, such as acoustic or ultrasonic, radiography, magnetic field process, thermal field methods, and Eddy-Current, for structural health monitoring. Various non-destructive testing (NDT) technologies are used mainly to track, locate, and characterize structural damage and to determine their state as compared to the state when the structure was constructed (undamaged state). Among all the damage assessment methods, vibration-based methods are the most popular because of various advantages like getting the global response of the structure from a few key measurements. Among vibration-based methods, damage assessment using the natural frequency is the most common and is successfully used in various fields like aerospace, automobile, and machine manufacturing industries.

However, in civil engineering, it is not as popular as in other fields for the following reasons:

- Massiveness of structures
- Lack of testing at full scale
- Complexity of construction materials

To address the above-mentioned constraints, the development of SHM is supplemented by the power of computational modelling, where the damage in the structure is simulated and the vibration response is predicted from the simulation, and then compared with the experiments.

1.2 Problem Statement

Structure Health Monitoring is very important nowadays. The purpose is to detect any kind of damage at the initial stages to make sure safety and to save repair costs. There are various methods to measure structure health monitoring but most of them are costly and there is a need for complementing software to automate the calculations. So, there is a need for the **development of a low-cost solution for remote Structural Health Monitoring.**

Although there has been progress in damage assessment using degradation in natural frequencies (determined by vibration methods), these methods still haven't addressed the constraints in the successful deployment of damage assessment methods. Furthermore, these methods are very sensitive, and, while deploying in the field, may overcome the environmental effects. Therefore, there is a need to address the sensitivity issue. Lastly, there is a rapid increase in the infrastructure, and there will be human resource and financial problems in the future, where there will be a need to employ an efficient and reasonably automated method for damage assessment.

1.3 Objectives:

There are four main objectives of this study described below:

- Development of a prototype
- Using multiple nodes for data acquisition
- Making a server and website
- Making a real-time warning system

CHAPTER 2

LITERATURE REVIEW

2.1 Significance of Real-time Structure Health Monitoring:

Structural health monitoring is the process of identifying and tracking the integrity of a structure, assessing structural damage, and implementing a damage identification plan for civil and mechanical engineering. The primary goal of structural health monitoring is to identify the condition and location of structural damage following a major impact, such as a strong earthquake. Monitoring structural health is critical in post-event emergency response, rescue, and management. The primary focus of the structural health monitoring is to study damage in reinforced concrete. In most of the structures, minor damage is permissible. However, the degree of damage increases with time. When damage has progressed to the point that the structure is no longer serviceable, i.e., system operation is hampered, this is referred to as a failure. Poor construction practices, environmental problems, and natural hazards are the most common causes of concrete structure failure.

It is essential to treat structural damage as quickly as possible. This is done to ensure the structure's reliability and safety. Damage detection can be performed before a failure occurs, i.e., the structure is no longer suitable for use. There is a need for an extensive evaluation of civil infrastructures after any serious catastrophe. The ability of a structural health monitoring system to quickly evaluate the structure's damaged condition after a severe event will ensure the structure's safe operation.

With modern techniques of SHM i.e., by using sensors and extraction of dynamics parameters from data, the contribution of engineers towards public safety has been increased. We can determine the strength of old buildings and bridges and can figure out their expected future life (Sohn et al., 2001).

The functionality of a structure in all conditions depends on its structural health, obtained through structural health monitoring (SHM) practices. The main objective of SHM research is to evaluate the structure's condition with the least possible human involvement and to trigger a timely warning if the structural damage exceeds serviceable limits. The structure is expected to be serviceable for its design life. However, structures deteriorate before completing the design life due to structural overloads, changes in load frequency, and environmental factors.

2.2 Operational modal analysis:

Usually, we perform operational modal analysis for damage identification and various techniques have been identified for the Non-Destructive Testing of structure. Damage in structure can be of any kind from micro cracks to macro one. Damage is not limited to changes in the material level but also in the boundary conditions and system connectivity.

2.3 Damage Indicators:

A damage indicator is required to identify the existing damage. Damage indicators can be determined as a dynamic quantity that can be used to indicate that the structure has damage (Rytter, 1993).

2.3.1 Natural Frequencies:

Natural frequency changes are known for being one of the most conventional indicators of damage. Natural frequencies became very popular because they were easily determined with a high degree of precision. Furthermore, natural frequencies are sensitive both to local and global damage. You can estimate several natural frequencies by placing a minimum number of sensors on a structure connected to a frequency analyser. Research has shown that natural frequencies are the global property of a structure.

2.3.2 Rytter's damage identification:

"A system of classification for damage-identification methods, as presented by Rytter (1993), defines four levels of damage identification, as follows:

Level 1: Determination that damage is present in the structure

Level 2: Determination of the geometric location of the damage

Level 3: Quantification of the severity of the damage

Level 4: Prediction of the remaining service life of the structure (Chan et al., 2016)"

Nevertheless, there is still room to discuss the ability to change natural frequencies for damages above Level 1. In another word, it is usually impossible for this parameter to give spatial details about structural changes. Anyway, this limitation is excluded. Spatial information is provided when modes are linked to higher natural frequencies with local response.

Old methods of identifying damages are of visual or localized experiments such as:

• Acoustic or ultrasonic methods

• Radiography

• Thermal field methods (Kobayashi 1987).

• Eddy-current methods

2.3.3 Mode Shapes:

That is why mode shapes were used to increase the accuracy and rapidity of the experimental equipment as damages indicators in the 1980s. With the performance of damage detection at offshore platforms, used stiffness dependence to recommend stiffness monitoring using modal shapes rather than natural frequencies.

2.3.4 Destructive and Non-Destructive Methods:

Destructive tests involve damaging (at least) some of an element or structure. Some like to include purpose-moulded cylinders or cubes or prisms. For example, drilling cores or load testing to failure are thought of as being destructive. Tests on hardened specimens made specifically for testing is destructive, but it certainly does not damage a structure, of course. Visual Inspection is of limited use as they are time consuming, depend upon experience, and quality of work is limited.

The method used to inspect, evaluate, and test materials and systems without damaging the serviceability of a component and system is referred to as non-destructive testing (NDT). Similarly, these inspections are not feasible in large structures such as building, long span bridges, dams. The choice of specific NDT methods, including accessibility, availability, and adequacy based on analysis and experience, is affected by several factors (Zhu et al., 2019). Five important factors are to be considered in the design of an NDT survey, according to McCann and Forde (McCann Forde, 2001).

The following factors are:

- The necessary penetration depth in the structure.
- The vertical and lateral resolution is necessary for the intended objectives.
- The contrast between the target and its environment in physical properties.
- Signal to noise for the physical property measured in the investigated structure.
- History of the methods used in the construction of the structure.

2.4 Damage Detection Using Vibration Method:

In industrialized countries, the application of global and automated damage detection and monitoring techniques in existing structures is widely acknowledged. The importance of these methods of structural health monitoring is undeniable. Vibration methods are determined based on changes in vibration properties. The characteristics of vibration include natural frequencies, damping, etc. The fundamental idea is that these properties are functions of the structure's physical properties. Therefore, any structural health changes can be detected through the changes in vibrational characteristics.

Structural health monitoring can divide into two methods, local and global. The local method is vulnerable to minor damage but normally small in its range. While the global methods only need

a small number of sensors, the acquired information is frequently not sufficiently sensitive to minor structural harm. The focus of this study is the global method. Global methods can generally be split into two categories. It is linear and non-linear. A linear method assumes that the answer of the structure can be modelled with linear equations of motion even after damage. The response modelled in the vibration is linear for a wide range of structures with different types of damage.

To make health monitoring of structures more feasible it is necessary to adopt modern techniques instead of old traditional methods. With time constructions methods are improving day by day and modern methods are so fast and reliable that high-rise buildings are now a trend. But there is still an area for improvement in the health monitoring of structures. Traditional techniques of Health monitoring and damage detection are cumbersome and very costly. While modern techniques are way better and easy to deal even for high-rise structures. The basic concept behind VBDD is that due to a change in physical properties (mass, damping, and stiffness) of structures a change occurs in its vibration-based parameters (natural frequencies, modal damping, and mode shapes). This change in vibration-based features can be detected. We can analyse data either in the time domain or frequency domain.

The excitation produced in the structure and response that the structure gives is measured and documented in the form of a timeline. But in time history analysis it is very difficult to examine the data for damage detection. So, the time domain data is transferred into the frequency domain, and the structure's modal properties are recovered from the frequency domain data. We need more improvements in this method because working in one domain out of three domains (i.e., time, frequency, and modal domains) cannot give us more reliable results. For the past few decades, researchers found that the modal domain analysis is a more efficient method and gives reliable results compared to the other two methods.

Modal properties (i.e., modal damping, modal shapes, natural frequencies, etc.) are preferable to work with as they have their physical meanings and can easily be interpreted and interrogated compared to those parameters extracted from the time or frequency domain (Enayati & Pishro-Nik, 2020). The natural frequency change is given importance and selected to work with it because it can be measured for a few accessible points of structure and is less interfered with by noise during testing.

2.4.1 The Forward Problem:

By using forward problem technique, we can determine loss of mass as well loss in stiffness of a structure. Forward problem method detect change in change in natural frequencies of a given structure based on the location and severity of damage. It provides theoretical foundation for natural frequency-based techniques.

2.4.2 The Inverse Problem:

This technique is somehow different than previous one as in inverse problem the determination of location of damage and size of damage is determined based on natural frequency measurement. Some features and functions related to MATLAB are discussed here which are very important to understand for the purpose of a better understanding of all process.

2.5 Filtration of Noise:

Various filters can also be applied to the compiled data. The filters can be used for:

1. **Removing high frequency data (Noise):**

   High frequency data content is usually noise present in the signals. These can be removed by using low pass filters such as Butterworth, FIR, and IIR. (C, 1991).

2. **Removing outliers:**

   Outliers or chirps can be removed using Hampel and remove outliers command in MATLAB.

3. **Decimating:**

   Decimation is the process of reducing the sampling rate of any given sample. If the raw data is highly oversampled, decimation is carried out by applying either using low pass filters or using decimate function in MATLAB (Hill et al., 2019).

4. **Smoothing the data:**

   There are various filters to smooth the data signals such moving average filters, Savitzky-Golay filter, etc.

2.6 Power spectral density function (PSD):

PSD allows presenting the frequencies concentration in terms of intensity, and energy. In simple terms, it helps in ranking the frequencies in descending order. The unit of PSD is the singular value of the chart, and we can obtain the total energy of a system by incorporating PSD with a range of frequency bandwidths. The power spectral density (PSD) gives value in terms of the spectral density of its mean square value (Garnier & Young, 2014).

2.7 Internet of things (IoT):

The world is shifting towards automation, sustainability, and technology advancement. To improve the civil engineering application there is a need to adopt new technology, innovative approach, and promote technology adoption in the field.

- **History:**

The Internet of things (IoT) is an innovative idea developed by Kevin Ashton in 1999 (Ashton, 2009). It is also known as the "Internet of the future". It can also be explained as a system of devices, mechanisms for computing, and data exchange/transfer connected to the internet to give benefits and ease of use.

- **Concept of IoT:**

The term Internet of Things (IoT) was first coined to describe uniquely identifiable and interoperable objects linked to RFID technology. Later, specialists referred to IoT with modern technology and advancements like actuators, GPS gadgets, cell phones, and sensors. Nowadays, an acknowledged definition for IoT is a framework of a powerful worldwide organization with self-organizing abilities that rely on interoperable and normative conventions in which virtual and physical 'Things' have real-world attributes and virtual personalities.

The idea of IoT is that of an ecosystem of objects and things that can communicate and co-operate with each other through some means so that the work is done more efficiently. The means here are sensors, radio frequency identification (RFID) tags, mobile phones, actuators, etc. for pre-set objectives.

2.7.1 Advantages in IoT Adoption:

IoT in civil engineering has many benefits. Some of them are on-site monitoring, improving quality and cost reduction, effectively controlling, and timesaving (Ning & Xu, 2010) (Gubbi et al, 2013) (Dave et al., 2016). It can also improve emergency and crisis management with well-organized monitoring health of the structure (Zhao et al, 2013).

Extracted from the literature following are some of the advantages of IoT adoption in civil engineering.

1) **Real time Structural health monitoring:**

IoT is also used in structural health monitoring before and after construction to detect vibrations, cracks, and the states of critical building members and civil structures. Structure Health Monitoring is very important nowadays. The purpose is to detect any kind of damage at the initial stages to make sure safety and to save repair costs. IoT can also improve emergency and crisis management with well-organized monitoring health of the structure.

2) Health, Safety & Security:

Health and safety are one of the extreme dares on a project site and it can cost more than the total project cost. Health monitoring officers are not satisfactory enough to monitor a complex construction project of large scale. With IoT adoption health and safety problems can be tackled easily. Moreover, IoT enabled tags can help in recovering any displacement of items, as these instruments will identify the exact location of material or items and inform the concerned. Furthermore, IoT can play a role in risk management by allowing us to be updated about associated risks of a project and it can also enable workers to work in a risk-free environment by notifying them when getting close to a hazard or entering a hazardous setting.

3) Productivity:

The construction industry is retrained with timelines and goals. It is essential to avoid backlogs because they increase the project cost. With IoT adoption, there will be more efficiency and readiness so that productivity is enhanced. IoT allows individuals with modest work, and, all things considered, they have presented a better opportunity to communicate effectively with team members and project proprietors, producing ground-breaking plans to improve project conveyance and consumer loyalty. Moreover, sufficient supply of materials is necessary in construction projects. Though, there might be a possibility of late supply of material at site due to scheduling errors. By adopting IoT the materials supply branch can be equipped with proper sensors which can determine the quantity of materials and inform the concerned.

4) Maintenance:

If not actively managed consumption of energy can cause wastage leading to increase in project costs. With real time information analysis, it is more likely to knows the condition of every working and idle equipment, to organize timely upkeep and refuelling of non-working equipment. Moreover, field sensors can play an important role in preventing problems from happening eventually reducing warranty claims. Moreover, sensors can also be used for quality assurance of materials and structure health monitoring.

5) Real time Concrete Monitoring:

IoT in concrete curing is trending now a days. In this process sensors are installed in mix during concrete placement on-site, and then the construction managers follow the curing in real time and can manage their schedule more accurately. This accurate in situ estimation of concrete compressive strength provides opportunity to enhance crucial on-site construction processes such as time for removal of scaffoldings and improve properties of concrete mix design. In addition, time for opening of rehabilitated road and bridge for traffic can also be determined. With concrete maturity known the loss of scheduling and cycling of form work can be optimized, thus reducing labour requirement and reduction in costs.

2.7.2 Barriers to the adoption of IoT:

With new technology, there come barriers. In this section, some of the barriers are discussed to the adoption of IoT in civil engineering. These barriers are extracted through a literature review.

1) **The inability of existing systems:**

This means that there is an inability in existing systems to manage errors while functioning. Because IoT requires advanced systems (Stankovic & J.A, 2014).

2) **Safety and security issues:**

This means that there is the possibility that the connected devices' security can be compromised because they are after all connected to the internet. Therefore, without proper protection Internet of things might be weak (Babar et al, 2010).

3) **Improper understanding of IoT:**

There is not enough knowledge about the techniques used to apply IoT in the field, this clarifies that there is an improper understanding of IoT. There is limited research on IoT in civil engineering which leads to a lack of IoT technology in construction (Bertino & Islam, 2017).

4) **Big data Issue and Inaccuracy of data:**

Due to the enormous amount of data, it is more difficult to understand. Also, as information is extracted from a large set of data, this causes ambiguity in information selection to perform a certain function (Matharu et al, 2014).

5) **Incompatibility between the systems:**

The contrariness between the framework of IOT, its gadgets and the trouble to share and impart administration. (Noura & Atiquzzaman, 2018).

CHAPTER 3

THEORETICAL BACKGROUND

3.1 Dynamic System

Each structural member in a structure contributes with some degree to the mass, stiffness and damping of the system (Garnier & Young, 2014). The degrees of freedom in the system are the number of independent displacements needed to describe the relative change in position of the masses in the system. Structures are composed of an infinite number of degrees of freedom. In structural dynamic the most basic way to describe a system is as a single degree of freedom (SDOF) system. A SDOF system can be represented by e.g., an oscillator, allowed to only displace in the lateral direction. Figure illustrates the oscillator consisting of a mass, m, a massless frame with stiffness k and a viscous damper with damping coefficient c. It is excited by an externally applied dynamic force $F(t)$ that varies with time,t. The resulting time dependent displacement, velocity and acceleration that occur is denoted$u(t)$, $\dot{u}(t)$ and$\ddot{u}(t)$, respectively

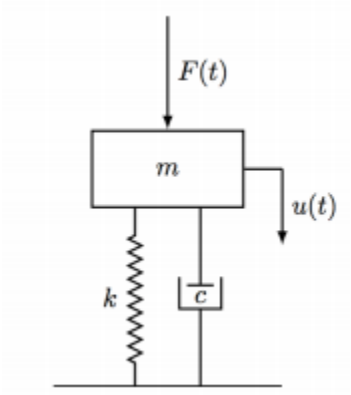

Figure 1 Single Degree of Freedom

The external- and inertia force resisting the acceleration of the mass give rise to a resulting force,$P(t)$ in accordance with Newton's second law of motion:

$$P(t) = F(t) - m\ddot{u}(t) \tag{3.1}$$

Elastic- and damping forces are internal forces that arise resisting the deformation and velocity of the mass, respectively. The resultant is hence giving as

$$P(t) = c\dot{u}(t) + ku(t) \tag{3.2}$$

The equation governing the physical behaviour of the structure i.e., the equation of motion is given by:

$$m\ddot{u}(t) + c\dot{u}(t) + ku(t) = F(t) \tag{3.3}$$

Before proceeding to solving the equation of motion, some central concepts regarding structural dynamics should be addressed. The natural cyclic frequency, w_n is given in units of radians per second and is for an undamped SDOF free vibration system given as

$$\omega_n = \sqrt{\frac{k}{m}} \tag{3.4}$$

The natural period of vibration, TN, is the time required for the undamped system to complete one cycle of free vibration.

$$T_n = \frac{2\pi}{\omega_n} \tag{3.5}$$

The natural frequency represents the number of cycles that occur during one second and is given in hertz (Hz) as

$$f_n = \frac{1}{T_n} \tag{3.6}$$

The SDOF system described above is not always applicable to real life structures. The reason being that structures are composed of an infinite number of degrees of freedom and not all structures can be idealized as SDOF systems since such idealizations may yield inaccurate results in terms of e.g., frequencies and mode shapes. In such cases, the dynamic behaviour can be described more accurately by discretizing structures into systems of elements with a finite number of degrees of freedom. This system is referred to as a multi degree of freedom system (MDOF). The basis of the theory describing a MDOF system is analogous to the theory described above for the SDOF system. It is a generalization from one to N number of dimensions where N is the number of degrees of freedom in the system. The equation of motion for a multi degree of freedom (MDOF) system is thus given as

$$m\ddot{u}(t) + c\dot{u}(t) + ku(t) = F(t) \tag{3.7}$$

where m is the mass matrix, c is the damping matrix and k is the stiffness matrix. They are all matrices of order $N \times N$. Displacements, velocities, and accelerations for each degree of freedom are now given as time-dependent vectors of order $N \times 1$ and are denoted as $u(t)$, $\dot{u}(t)$ and $\ddot{u}(t)$, respectively.

$F(t)$ is referred to as a load vector of order $N \times 1$ describing the external forces acting on each degree of freedom.

The mass matrix is determined by assuming the mass of the system to be concentrated at the nodes i.e., where the degrees of freedoms are located. The stiffness matrix is constructed by assembling the local stiffness matrix of each element. There are several ways to construct the damping matrix which is further discussed in the following section.

3.2 Damping Force:

The procedure by which free vibration relentlessly reduces in amplitude is called damping. In damping, the vitality of the vibrating framework is disseminated by different components, and frequently more than one system might be available in the meantime. In genuine structures, be that as it may, numerous different instruments likewise add to the vitality dispersal. In a vibrating building these incorporate contacts at steel associations, opening and shutting of micro cracks in cement, and erosion between the structure itself and non-structural components, for example, parcel dividers. It appears to be difficult to distinguish or depict scientifically every one of these vitality disseminating components in a real structure.

Subsequently, the damping in real structures is normally spoken to in a very idealized way. For the same reasons, the real damping in SDF structure can be idealized by a direct gooey damper or dashpot. The damping coefficient is chosen with the goal that the vibrational vitality it disseminates is identical to the vitality scattered in all the damping systems, present in the real structure.

3.3 Natural Frequencies and Modes of Vibration

The natural frequencies and modes of a structure are determined by evaluating the undamped free vibration i.e., in the absence of damping and external loading. In each natural mode, the system oscillates at its natural frequency with all degrees of freedoms of the system vibrating in the same phase, passing through their maximum, minimum and zero displacement positions at the same instant of time. Free vibration occurs when a structure is disturbed from its static equilibrium position and allowed to vibrate without an externally applied dynamic force acting on it. This is achieved by introducing an initial displacement and velocity.

$$u = u(0)\ ;\ \dot{u} = \dot{u}(0) \tag{3.8}$$

The equation of motion of an undamped MDOF system undergoing free vibration is governed by.

$$\boldsymbol{m}\ddot{u} + \boldsymbol{k}u = 0 \tag{3.9}$$

This equation depends on N number of degrees of freedom and homogeneous differential equations coupled through the mass and stiffness matrix. The displacement can mathematically be described as

$$\boldsymbol{u}(t) = \phi_n q_n(t) \tag{3.10}$$

where $\boldsymbol{\varphi_n}$ is the deflected shape of the system i.e., the mode shape and $q_n(t)$ is the time variation of the displacement which can be described as a harmonic function:

$$q_n(t) = A_n \cos \omega_n t + B_n \sin \omega_n t \tag{3.11}$$

A_n and B_n are constants that are to be determined using the initial conditions. Combining Eq. 2.14 and Eq. 2.15 yields

$$u(t) = \phi_n (A_n \cos \omega_n t + B_n \sin \omega_n t) \tag{3.12}$$

Differentiating the displacement twice with respect to time gives the time variation of the acceleration:

$$\ddot{u}(t) = -\omega_n^2 \phi_n (A_n \cos \omega_n t + B_n \sin \omega_n t) \tag{3.13}$$

Substituting the above two equations into the equation of motion yields

$$[-\omega_n^2 \boldsymbol{m}\phi_n + \boldsymbol{k}\phi_n]q_n(t) = 0 \tag{3.14}$$

There are two ways to satisfy this equation. The first solution qn(t) = 0 implies that there is no motion of the system since u(t) always will be equal to zero. In the second solution, the natural cyclic frequencies $\boldsymbol{\omega_n}$ and mode shapes φn fulfil the following equation

$$[\boldsymbol{k} - \omega_n^2 \boldsymbol{m}]\phi_n = 0 \tag{3.15}$$

The trivial solution $\phi_n = 0$ implies no motion and is hence not of interest. Nontrivial solutions are obtained if

$$\det[\boldsymbol{k} - \omega_n^2 \boldsymbol{m}] = 0 \tag{3.16}$$

This is known as the Eigenvalue Problem.

3.4 System Identification:

System identification is the process in which we use signals from a system's input and output and can evaluate other parameters of that model. The accuracy of your system model is determined by how well your measured data represents the system's behaviour. Two different data are obtaining and processing methods such as time domain and frequency domain. These two domains are described below:

3.4.1 Time Domain Data:

The system's input and output variables are captured as time-domain data at a consistent sample interval.

If you measure the input force, F(t), and mass displacement, y(t), of the spring-mass-damper system at a uniform sampling frequency of 10 Hz, you receive the following vectors of measured values:

T represents the time since the previous measurement, and NTs represent the time since the previous measurement.

The data vectors u_{means} and y_{means}, as well as the sample time Ts, provide enough information to develop a discrete-time model from this data.

If you want to construct a continuous-time model, you should also understand the inter-sample behaviour of the input signals during the experiment. For example, the input might be piecewise constant (zero-order hold) or piecewise linear between samples (first-order hold).

3.4.2 Frequency Domain Data:

You record or store frequency domain data as measurements of the system's input and output variables. Frequency-domain signals are created via Fourier transforms of associated time-domain signals.

Frequency domain data may be used to depict a system's frequency response, which is expressed as a series of complex response values across a given frequency range. The frequency response describes the outputs of sinusoidal inputs. If the input signal has a frequency ω, the output signal has frequency, amplitude A(ω) times the input signal amplitude, and phase shift of **Φ(ω)** concerning the signal input This is the frequency response: $\mathbf{A(\omega)e^{(i\Phi(\omega))}}$**.**

Frequency-domain data may be used to develop discrete-time and continuous-time models of your system.

3.4.3 Requirements for Data Quality:

To identify your system, your data must capture its basic dynamics. A solid experimental design ensures that the right variables are measured correctly and for long enough to capture the dynamics you're interested in. In general, your experiment must:

- Use inputs that excite the system's dynamics appropriately. For example, a single step is rarely thrilling enough.
- Take long enough measurements to capture all the important time constants.
- Create a signal-to-noise ratio-optimized data acquisition system.
- Use proper sampling intervals or frequency resolution when measuring data (Kim et al., 2003).

3.5 Damage Detection:

Some of the approaches developed for detecting damage in structures based on changes in dynamic parameters are based on changes in mode shape. To acquire natural frequencies and mode shapes, these methods involve the employment of modal identification techniques. Modal identification is a time-consuming operation, and curve fitting always introduces some inevitable inaccuracies. Furthermore, simply employing mode shapes causes us to lose a lot of information. As a result, using measured raw data for damage identification (e.g., in the form of frequency response functions (FRFs)) may offer a significant benefit in most circumstances. When employing only

mode forms, it's possible that some of them won't show any discernible change if the damage is close to a node. The use of FRFs will, in principle, overcome this problem.

3.6 Damage Detection Methods:

All the approaches offered for comparison assume that damage occurs between the places where a mode shape function's change is highest. All of these methods require identification algorithms and may employ one or more mode shapes to detect damage." "Modal analysis of vibration response is used for non-destructive examination of structures." The relationship between natural frequencies and modal damping coefficients and structural degradation is investigated. The intensity and location of structural degradation determine the extent of change in natural frequencies. Specific degradation occurrences have a characteristic ensemble of natural frequency change ratios that may be gathered ahead of time to provide a database for subsequent analysis of observed modal parameter changes. Experiments on a welded steel frame subjected to fatigue loading and wire ropes damaged by saw cuts show the modal analysis approach. The technology has potential as a bridge and other skeletal structural condition monitoring tools.

CHAPTER 4

METHODOLOGY

4.1 Overview:

The methodology was based on the steps shown below, starting from the procurement of the individual components, their assembly in the form a complete node capable of acceleration data measurement and its on board storage, then the integration of multiple nodes to make a complete system and ultimately storing the post processed data from the nodes on server along with the display of the results in a graphical manner on the web page.

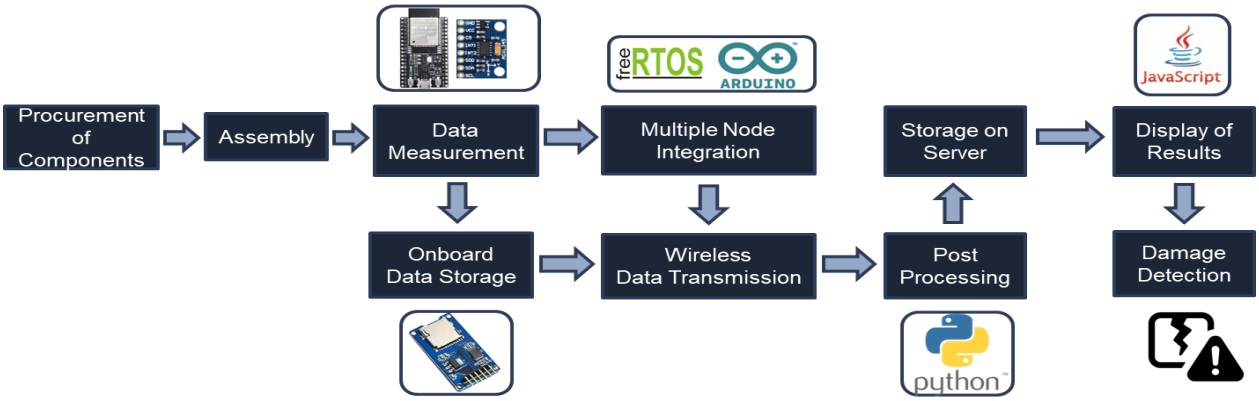

Figure 2 Methodology Flowchart

4.2 Hardware:

The hardware end comprises ADXL345, ESP32, SD card module, and power module.

Various micro-electromechanical systems (MEMS) based accelerometers were considered for the application, but ADXL series accelerometers were shortlisted because of their reported performance in structural health monitoring applications. The ADXL series of accelerometers consists of ADXL335, ADXL345, and ADXL355 (Chang & Shirazi, 2021). It was desirable to use ADXL355 however it was not available locally and procuring it internationally proved to be cost-prohibitive and time-intensive. Both ADXL335 and ADXL345 were available locally and ultimately ADXL345 was chosen as it offered superior specifications over the ADXL335. The

data range of ADXL345 is wider than ADXL335 and the power consumption of ADXL345 is also lower than ADXL335 (Pcr & Kit, 2012).

The selected microcontroller unit (MCU) should have the following features: low power consumption, a built-in Wi-Fi transmitter, and a system capable of fast read-write speed. As ESP32 has dual cores which allow for the simultaneous collection of data and writing it to the MicroSD card enabling consistent sampling rate up to 500Hz, also it has a built-in Wi-Fi receiver and transmitter which is used for the intercommunication between nodes as well as the transmission of data to the server (Anshori et al., 2022). Furthermore, it has excellent deep sleep implementation which is crucial in reducing the power consumption for longer periods of monitoring in addition to the built-in ESP-NOW protocol which allows for easy and reliable communication between the nodes. So, ESP 32 from Espressif Systems was selected as the microcontroller over the more commonly used Arduino-based microcontroller for the above-mentioned reasons.

Serial Peripheral Interface (SPI) with a 4-wire configuration was used by the accelerometer for communicating with the MCU. Similarly, SPI was also used for the communication between the MCU and the MicroSD card module. The power supply module uses a single 18650 Lithium-ion cell, with a built-in voltage converter capable of delivery regulated 3.3V and 5.0V. The MCU and the accelerometer operate on 3.3V while the MicroSD card module operates on 5.0V.

All the components were temporarily mounted on a breadboard and were tested to ensure their functionality. For the final assembly of the node, all the individual components were permanently soldered on a solderable breadboard. This is done to reduce the noise in the system due to wires and to ensure that under vibrations no connection comes loose. The completed circuit board was then placed in a plastic enclosure and was attached to the bottom of the enclosure with machine screws that passed through the accelerometer. As the accelerometer is rigidly attached to the enclosure and so the vibrations experienced by the enclosure will pass directly to the accelerometer without any attenuation.

The figures below show the node in its final form with all the components permanently soldered and mounted in the enclosure.

Figure 3 Node without the power module

Figure 4 Node with the power module

4.3 Software

The software end is divided into two parts. The first part consists of the code which runs on the nodes and is responsible for the monitoring, collection, storage, and transmission of data along with the intercommunication among nodes. The second part consists of the code which runs at the server end and is responsible for the storage and post-processing of the incoming data along with its display on the webpage.

The MCU programming was done in Microsoft Visual Studio Code and the code was written in C++. The typical MCU programming approach runs through a code line by line from top to bottom. This results in no control over the individual tasks being performed. Also, in the typical approach, the MCU itself controls both the cores and assigns them tasks; this negates the benefits of having a dual-core MCU. Due to this a continuous loop of operations is formed, starting with reading the acceleration values, temporarily storing them, and permanently writing them to the MicroSD card. Initially, the MCU can keep up with the sampling rate but gradually it starts to slow down, this

results in the sampling rate increasing from the originally set sampling rate. This changing sampling rate renders the data unusable.

To mitigate this issue a real-time operating system (RTOS) known as FreeRTOS was utilized for controlling the sequence in which the code runs. RTOS through its task scheduler enabled the precise control of the sequence in which the code proceeds along with accurate timekeeping. Furthermore, RTOS can control both the cores and their assigned tasks separately. In this way, both the cores can be dedicated to different tasks. Three tasks were defined in RTOS, the first task was assigned with taking acceleration data from the accelerometer, the second task was assigned with writing this data to a queue, and the third task was assigned with permanently writing the queue data to the microSD card. Core 1 was dedicated to the first task while core 2 was dedicated to tasks two and three. A single delay occurs when core 2 momentarily switches from task two to task three and back. This delay is insignificant as the sample length is less than the interval at which the delay happens. Using RTOS, the delays in individual acceleration values and the associated inconsistency in the sampling rate were eliminated.

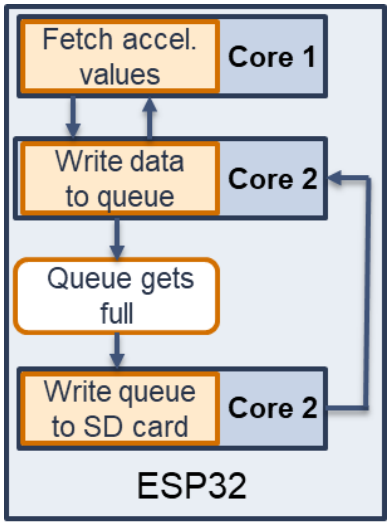

Figure 5 Traditional approach

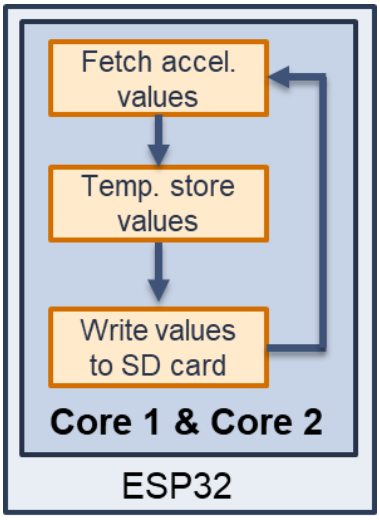

Figure 6 With FreeRTOS

The master and slave nodes are interlinked through a connectionless Wi-Fi communication protocol called ESP-NOW. This communication link is one-way in nature and is utilized by the master node for triggering the slave nodes. Using this link, all the nodes start their acceleration recording simultaneously. The ESP-NOW protocol uses Wi-Fi that is also used by the nodes for

connecting with the internet. This resulted in a conflict in which both of the systems cannot be used at the same time because the Wi-Fi channel used by the ESP-NOW is different from the Wi-Fi channel used by the node for internet connectivity. This issue was resolved through a piece of code that runs during the initialization phase of the node. This piece of code reads the Wi-Fi channel which is used by the node for the internet connection and then sets the ESP-NOW Wi-Fi channel to be the same as that used for the internet connection. Now the Wi-Fi channels used by both the systems are the same so both systems can operate simultaneously without any interference.

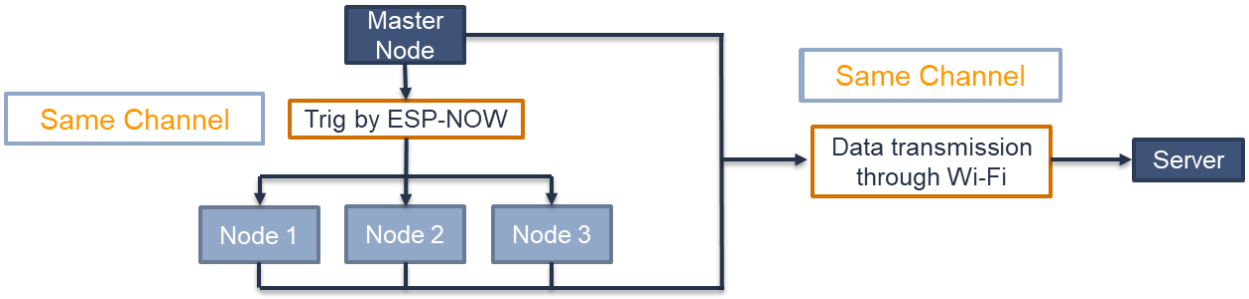

Figure 7 Intercommunication between nodes and communicate with the server

As the acceleration values are collected at a high sampling rate of 128Hz so they could not be collected and transmitted to the server simultaneously without inducing transmission delays. To mitigate this issue, permanent onboard storage in the form of a MicroSD card module was added. The acceleration values, fetched from the accelerometer, are written in a comma-separated values (CSV) file which is stored on the MicroSD card. Once the data collection is completed, a CSV file containing the acceleration data from a single event is generated. This CSV file is then sent to the server. As the combined operation of data collection and transmission is divided into two independent operations so the delays associated with the simultaneous collection and transmission of data are eliminated. In case of a failed data transmission, this approach has an additional advantage, that recorded data is not lost as it is permanently stored in the MicroSD card.

To increase the standby time of the node built-in low-power mode of ESP32 known as deep sleep was utilized. During deep sleep, all the peripherals of the MCU are powered off, this results in a substantial lowering of the power consumption from 240mA to 10µA. The time spent by the node in deep sleep is controlled by a timer.

The data collection routine is programmed into ESP32. On initialization, the node connects with the internet through Wi-Fi and gets the current time from Network Time Protocol (NTP) server. It then checks the current time against the predefined start time. If they coincide it immediately moves to the data collection phase and if they do not coincide, it then calculates the time difference from the current time to the start time. And then the node goes to sleep for that time interval. When the node wakes up from sleep at the start time, it initializes the ESP-Now protocol for the intercommunication between the master node and the slave nodes and then creates a file for storing the acceleration data. It then passes the control to the data collection phase.

The data collection routine is further divided into two sub-routines on basis of the trigger modes used for data collection. The first mode is acceleration triggered while the second mode is time triggered. During the first 50 minutes of the start time, the node is in acceleration trigger mode, it monitors the acceleration and if the acceleration exceeds the preset acceleration threshold, it triggers all other nodes via ESP-NOW and then starts recording acceleration data. The recorded data is then sent to the server for post-processing and the node goes to sleep again without going into the time-triggered mode. The slave nodes also record and transmit their data to the server after getting triggered from the master node. On the other hand, if the first 50 minutes have passed and the acceleration values have not yet triggered the master node, then the node goes into time-triggered mode. It completes the above-mentioned process but at the pre-set time.

The captured acceleration data is post-processed to extract the fundamental frequencies of the structure. The post-processing in the initial phase was done in MATLAB to validate the prototype and the post-processing procedure however ultimately all the post-processing is done in Python and NodeJS.

As the accelerometers also measure static acceleration so the first step in post-processing is detrending. The inbuilt function of MATLAB is utilized to perform the detrending. After detrending the data is passed through a 2nd lowpass Butterworth filter with a normalized frequency cutoff limit of 0.5. The purpose of the low pass filter is to remove the high-frequency noise which is not a part of the fundamental frequencies of the structure. After the data is filtered, a Fast Fourier Transform is used to convert the data from the time domain to the frequency domain. The frequency-domain data is then analyzed to determine the change if any in the fundamental frequency of the structure.

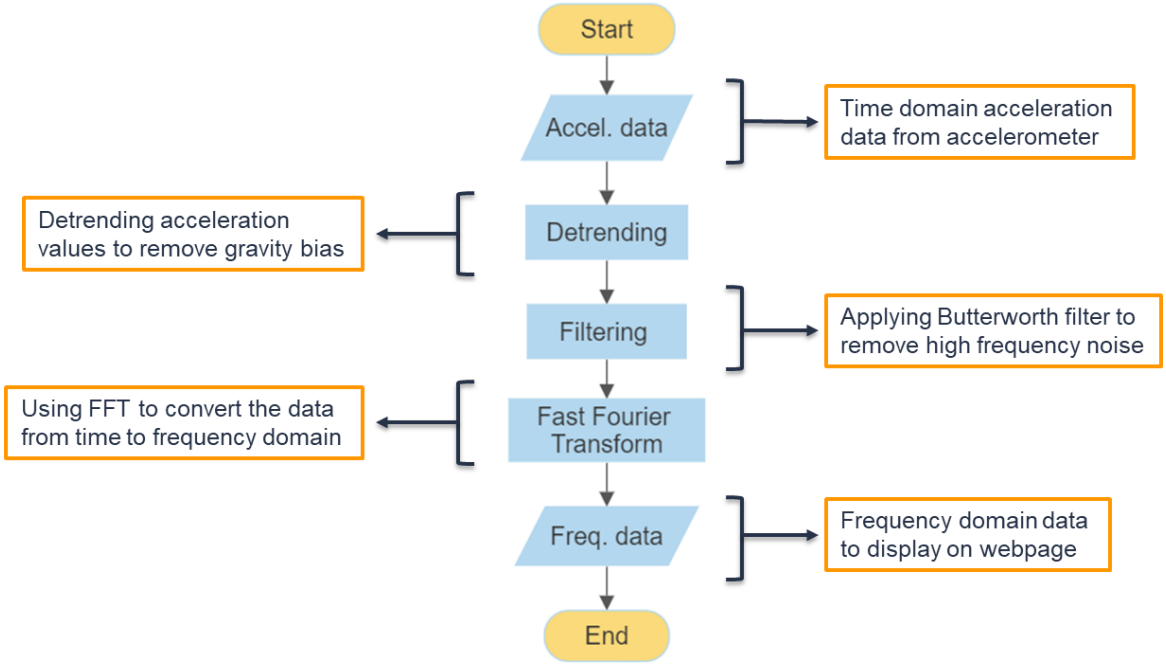

Figure 8 Post-processing steps

For ease in automation, Python was selected to automate the post-processing of data. The sequence of operations used in Python is the same as the one used in MATLAB with the exception that this process is automated. Firstly, the data is taken from the folder in which incoming acceleration data is stored. After the data is detrended and filtered it is converted into a frequency domain using FFT. The frequency-domain data is then stored for display on the web page.

Node JS is responsible for both receiving the incoming data from the nodes at the server end and for the display of the results on the web page. The prototype sends the data over Wi-Fi to the server

which is built in Node JS and the server downloads the data and then stores it in the designated folder. Then the post-processing is performed on the data and the results are stored in another folder. The results are then picked and displayed on the web page when it gets updated.

CHAPTER 5

EXPERIMENTAL VALIDATION

5.1 Lab-scale validation

The experimental procedure for lab-scale validation consisted of testing the node against a commercially available accelerometer on a beam apparatus. The commercially available sensor used was Gulf Coast Data Concepts X2-2 (Khan et al., 2022). An aluminum beam was held in the beam apparatus in a fix-fix support condition. The node was attached at the top of the beam while X2-2 was attached to the underside of the beam. The node and X2-2 were mounted on the beam using zip ties. The sampling rate was set at 128Hz, giving a Nyquist frequency of 64Hz. The testing parameters are given in the table below.

Table 1 Testing parameters

Property	Description
Test platform	Beam apparatus
Material	Aluminum
Dimensions	1000mm x 25.4mm x 4.8mm
End conditions	Fix-Fix
Node accelerometer	ADXL345
Reference accelerometer	GCDC X2-2
Placement	Middle of the beam
Excitation method	Manual
Sampling frequency	128Hz

To excite the beam, it was pulled from the center and released, and its acceleration response under free vibration was simultaneously recorded on both the prototype and the X2-2. The recorded acceleration data was then processed and FFTs were generated. The fundamental frequency from both the accelerometers was the same, validating the accelerometer and the data acquisition method. In this way, the fundamental frequency of the beam fixed in the beam apparatus under manual excitation was determined and validated against the commercially available reference accelerometer.

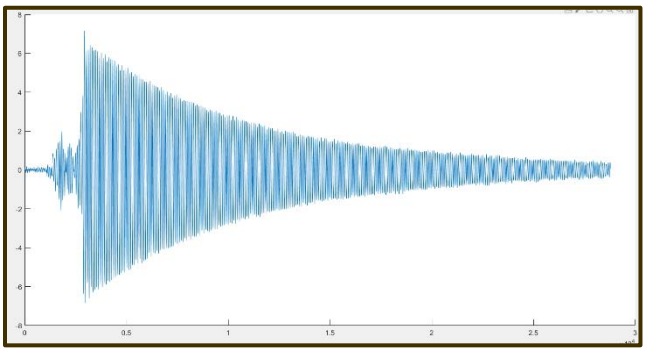

Figure 9 X2-2 Time domain data

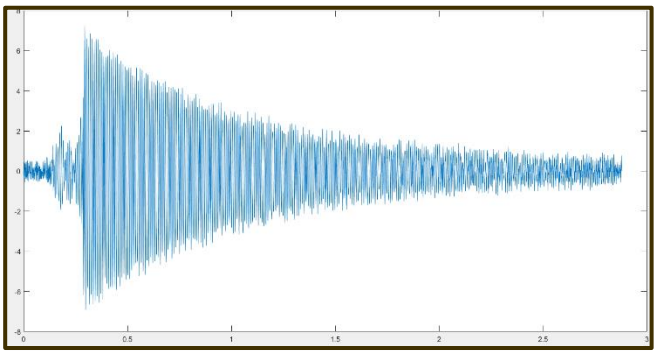

Figure 10 Node Time domain data

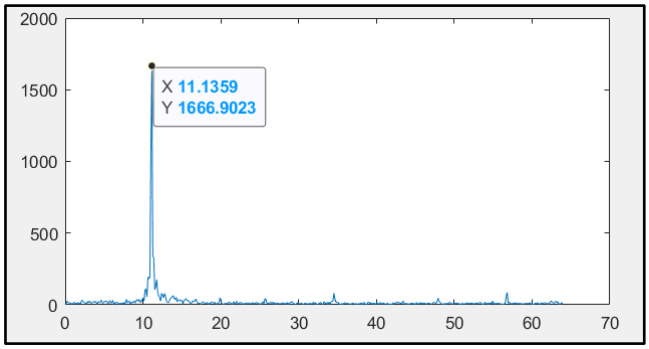

Figure 11 X2-2 Frequency domain data

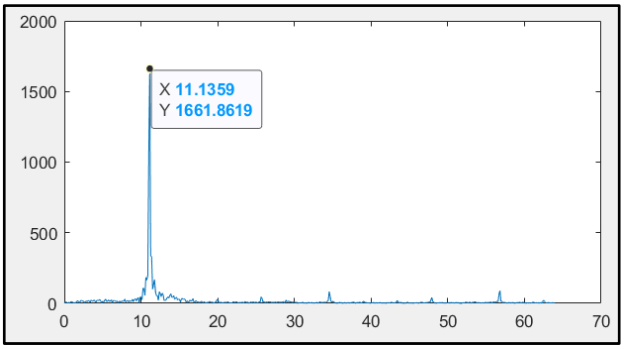

Figure 12 Node Frequency domain data

5.2 Full-scale validation

The full-scale validation of the system was performed on steel I beam which supports the 1st-floor slab of a double-story steel building. The beam had fixed ends and supported a steel deck which in turn supported a 6-inch concrete slab that formed the floor of the 1st story. The room measured 57'-9" by 30'. Three nodes were attached to the bottom flange of the beam at equal intervals. The nodes were mounted using double-sided tape.

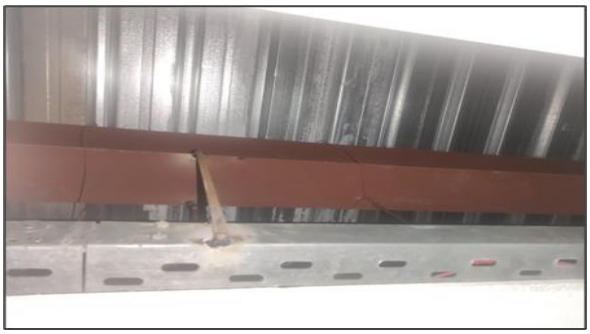

Figure 13 Beam under monitoring

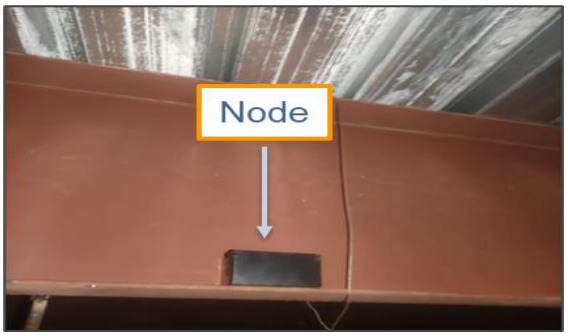

Figure 14 Node mounted on the beam flange

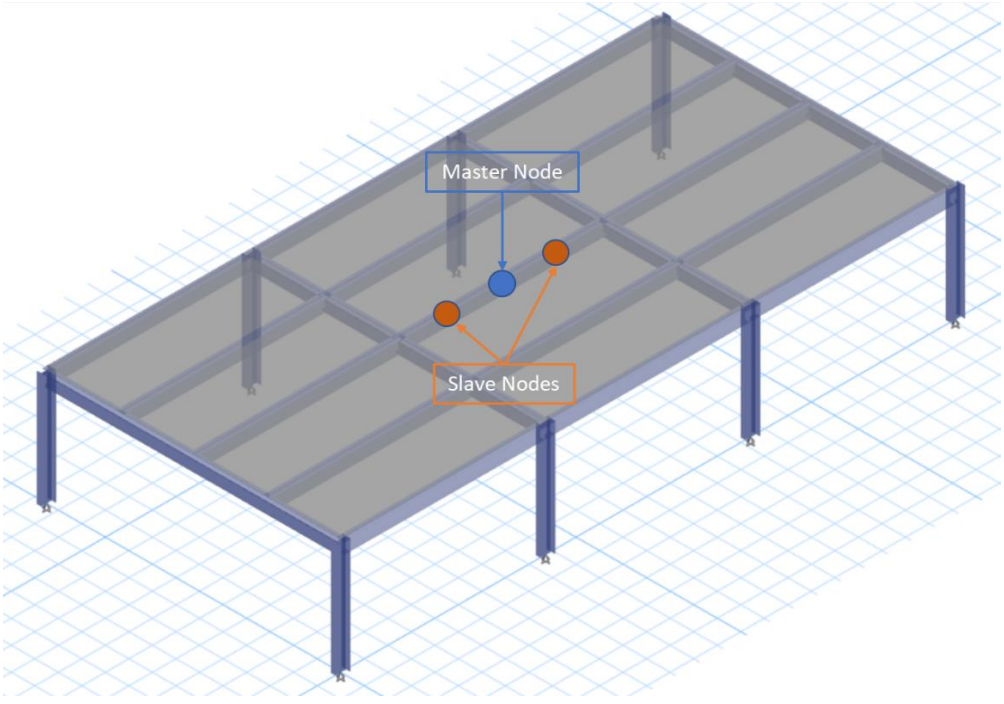

Figure 15 Schematic of the room with the dots indicating node locations

Table 2 Full scale validation parameters

Parameter	Description
Test platform	Steel I-beam
Total no. of nodes	3
No. of the master node	1
No. of slave nodes	2
Location on beam	Equidistant along the length of the beam
Attachment method	Double-sided tape
Excitation method	Vibration due to footsteps
Sampling frequency	128Hz
Nyquist frequency	64Hz
Sampling duration	30sec
Sampling time	11am-12am
Acceleration threshold	± 0.3 m/s^2
Total monitoring duration	1 month

The sampling rate was set to 128Hz with a Nyquist frequency of 64Hz. The recording length was set to 30 secs which gives 3840 acceleration data points. The programmed data collection routine consisted of 1 hour of monitoring every 24 hours. As the room above the beam was routinely occupied from 11 am to 12 pm for two days a week, so the start time selected for the data collection was 11 am. The system was set up and left to run for 1 month. During the one-month testing period of the system, acceleration-triggered data, as well as time-triggered data, were collected.

5.3 Results

The full-scale validation was carried out for a month. The following table shows the number of total acceleration samples, time-triggered samples and acceleration-triggered samples collected during the monitoring period of one month.

Table 3 Collected data samples

Sample type	No. of samples
Total data samples	28
Time triggered samples	25
Acceleration triggered samples	3

The time-triggered data samples contained only ambient vibrations. In the absence of any significant external vibrations, the ambient vibrations showed an acceleration signal without any substantial change and with a maximum variation of only 0.01. This negligible variation in acceleration resulted in a signal with low SNR. After post processing, it was found that time-triggered data samples were unusable due to the noise in the signal. The noise in the signal made it impossible to distinguish the fundamental frequencies from the noise.

As in the case of this structure the acceleration threshold was set to ± 0.3 m/s^2, so the system collected acceleration data only when the acceleration of the system exceeded that preset acceleration value. This resulted in the acceleration triggered data having variation of greater than ± 0.3 m/s^2 over the static acceleration of gravity. Due to the significant vibrations, the system was able to capture a good vibrational response of the structure. Therefore, the acceleration triggered signal had high SNR. After post processing, distinct peaks were visible in the FFT generated from the acceleration triggered data.

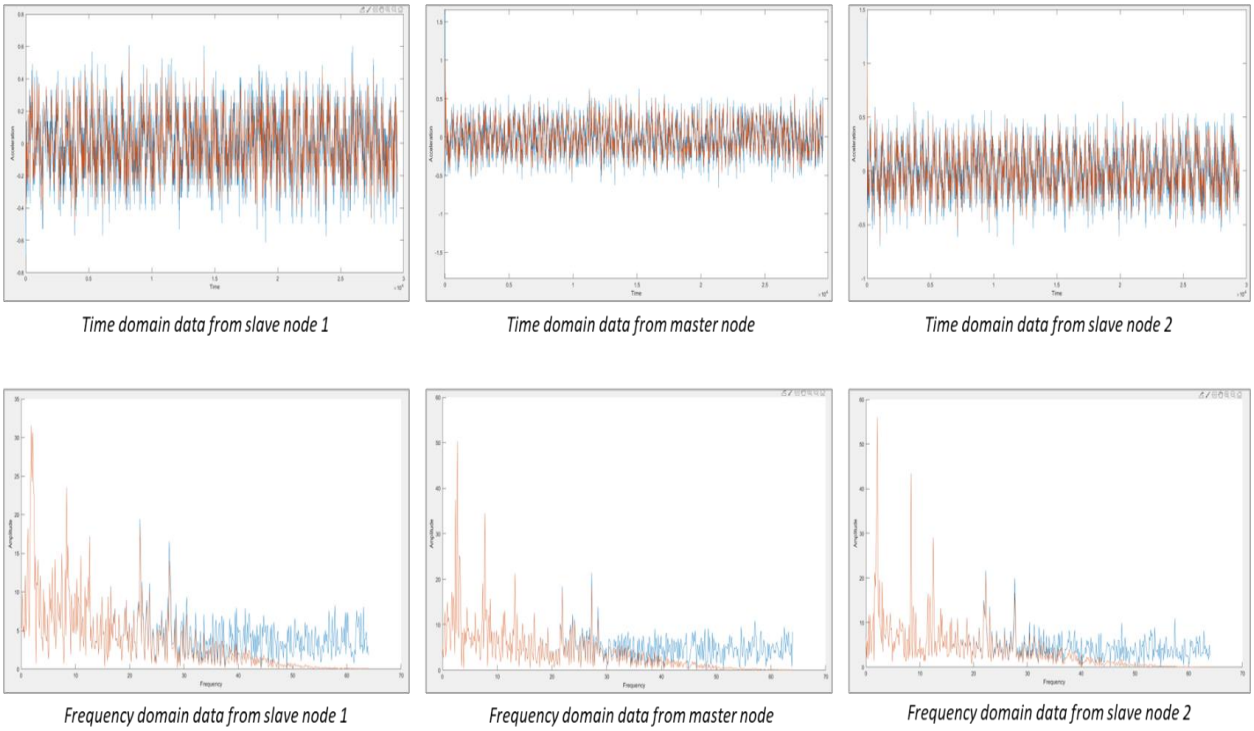

Figure 16 Acceleration-triggered data set 1

Four distinct peaks in the frequency spectrum showed the first four fundamental frequencies of the structure with each frequency corresponding to a particular mode shape. The lower frequencies showing the lower mode shapes and the higher frequencies showing the higher mode shapes respectively. Every data set corresponding to a single acceleration event consisted of three separate FFTs, one FFT from every node. The fundamental frequency from each FFT was averaged to give a single fundamental frequency corresponding to a particular mode for that data set. This was done for the four fundamental frequencies in each data set.

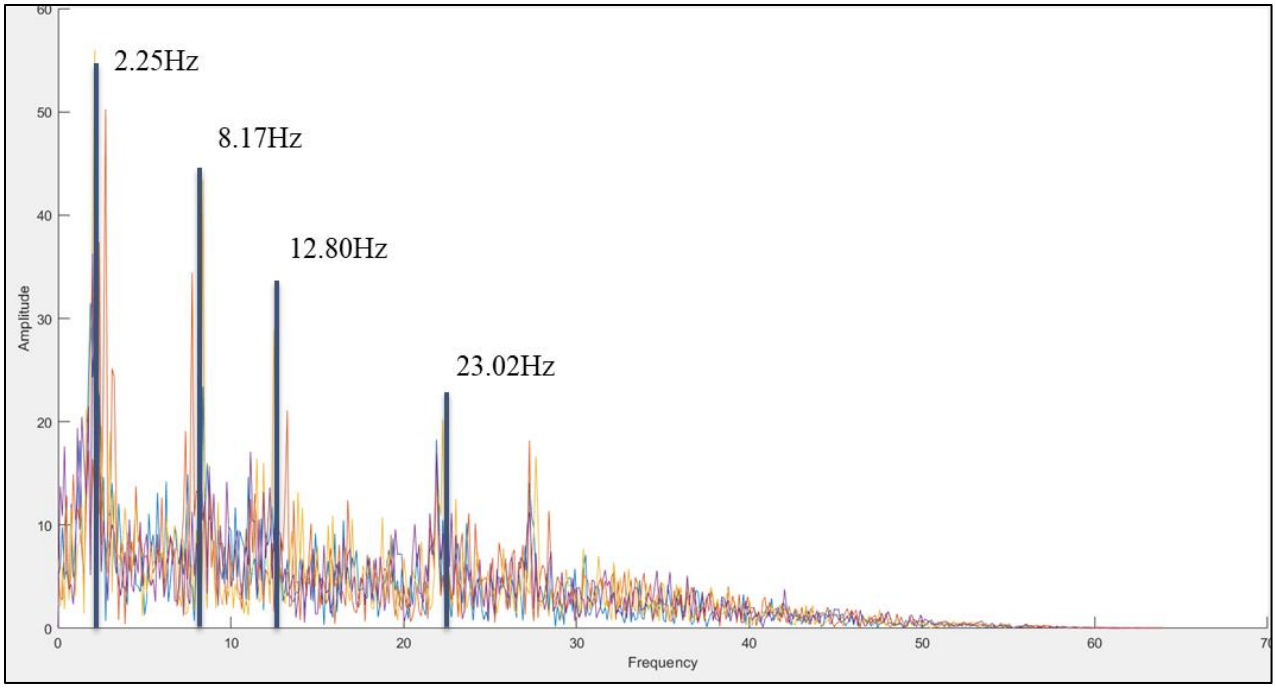

Figure 17 Superimposed frequency graph of three nodes with fundamental frequencies highlighted

The following table contains the three acceleration triggered data sets containing the fundamental frequencies corresponding to the four modes along with their average fundamental frequency and standard deviation.

Table 4 Fundamental frequencies from three data sets

Fundamental Frequency	**Data Set 1 (Hz)**	**Data Set 2 (Hz)**	**Data Set 3 (Hz)**	**Average (Hz)**	**Standard Deviation**
1st mode	2.2522	2.5517	2.7527	2.5187	0.21
2nd mode	8.1747	8.8303	9.2590	8.7546	0.44
3rd mode	12.8042	13.6383	13.1388	13.5271	0.34
4th mode	23.0215	24.3988	24.1486	23.8563	0.60

From the standard deviation, it was concluded that there was no significant variation between the fundamental frequencies determined from the three data sets. This was consistent with the condition of the monitored structure as remained unchanged during the duration of monitoring.

The results of the monitoring of the structure were displayed on a locally hosted website in the form of two-line graphs, of the frequency response of the structure, overlaid on each other. One of the line graphs is based on the frequency response of the structure in the healthy state acts as a baseline. The second frequency response graph is generated from the latest acceleration data that the nodes have sent to the server. The baseline frequency graph remains fixed while the changing frequency graph gets updated after the preset interval of 24 hours. For the ease of comparison, the baseline and the changing graph are colored green and blue respectively. Vertical black lines present on the baseline graph border the fundamental frequencies and are drawn at specific percentages of the fundamental frequencies based on the structure under monitoring. In our particular structure, they were at ±10% of fundamental frequencies. These lines set the maximum allowable deviation in the fundamental frequencies. The presence of damage is detected through the deviation of the current graph from the baseline.

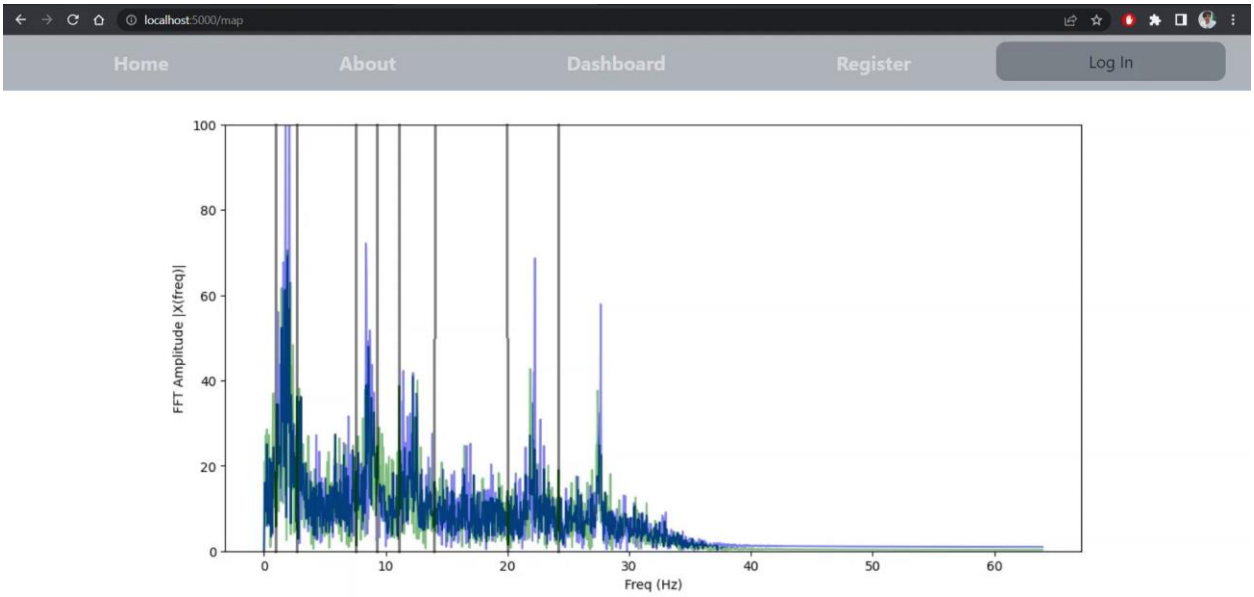

Figure 18 Graphical result on the website with green baseline and blue colored current graph.

CHAPTER 6

CONCLUSIONS

In this study, the suitability and effectiveness of microcontroller units and MEMS accelerometers for the purpose of vibrational based structural health monitoring was studied. And a low-cost remote real time structural health monitoring system based on components and sensors derived from IoT was developed. The system was tested at lab scale and full scale to ascertain its effectiveness. It is concluded that structure health monitoring system based on IoT tech offer a potential low-cost solution compared to existing systems. Further research and development in this regard through the use of advanced post processing techniques and more sensitive sensors will enable to enhance the effectiveness of future IoT based structural health monitoring systems.

REFERRENCES

- Anshori, I., Mufiddin, G. F., Ramadhan, I. F., Ariasena, E., Harimurti, S., Yunkins, H., & Kurniawan, C. (2022). Design of smartphone-controlled low-cost potentiostat for cyclic voltammetry analysis based on ESP32 microcontroller. *Sensing and Bio-Sensing Research*, *36*(March), 100490. https://doi.org/10.1016/j.sbsr.2022.100490
- Antunes, P., Travanca, R., Rodrigues, H., Melo, J., Jara, J., Varum, H., & André, P. (2012). Dynamic structural health monitoring of slender structures using optical sensors. *Sensors (Switzerland)*, *12*(5), 6629–6644. https://doi.org/10.3390/s120506629
- C, L. (1991). *Analysis of structural repsonse using discrete modal filters.pdf*. *May*, 71.
- Chan, T. C. Y., Ng, A. L. K., Cheng, G. P. M., Wang, Z., Woo, V. C. P., & Jhanji, V. (2016). Corneal Astigmatism and Aberrations After Combined Femtosecond-Assisted Phacoemulsification and Arcuate Keratotomy: Two-Year Results. *American Journal of Ophthalmology*, *170*, 83–90. https://doi.org/10.1016/j.ajo.2016.07.022
- Chang, H. F., & Shirazi, M. S. (2021). Integration with 3d visualization and iot-based sensors for real-time structural health monitoring. *Sensors*, *21*(21). https://doi.org/10.3390/s21216988
- Enayati, S., & Pishro-Nik, H. (2020). A Framework for Probabilistic Decision-Making Using Change-of-Probability Measures. *IEEE Access*, *8*, 159331–159350. https://doi.org/10.1109/ACCESS.2020.3020928
- Fan, W., & Qiao, P. (2011). Vibration-based damage identification methods: A review and comparative study. *Structural Health Monitoring*, *10*(1), 83–111. https://doi.org/10.1177/1475921710365419
- Fritzen, C. P. (2005). Vibration-Based Structural Health Monitoring – Concepts and Applications. *Key Engineering Materials*, *293–294*, 3–20. https://doi.org/10.4028/www.scientific.net/kem.293-294.3
- Garnier, H., & Young, P. C. (2014). The advantages of directly identifying continuous-time transfer function models in practical applications. *International Journal of Control*, *87*(7), 1319–1338. https://doi.org/10.1080/00207179.2013.840053
- Harms, T., Sedigh, S., & Bastianini, F. (2010). Structural Health Monitoring of Bridges Using Wireless Sensor Networks. *IEEE Instrumentation and Measurement Magazine*, *13*(6), 14–18. https://doi.org/10.1109/MIM.2010.5669608

- Hill, J., Rastas, P., Hornett, E. A., Neethiraj, R., Clark, N., Morehouse, N., De La Paz Celorio-Mancera, M., Cols, J. C., Dircksen, H., Meslin, C., Keehnen, N., Pruisscher, P., Sikkink, K., Vives, M., Vogel, H., Wiklund, C., Woronik, A., Boggs, C. L., Nylin, S., & Wheat, C. W. (2019). Unprecedented reorganization of holocentric chromosomes provides insights into the enigma of lepidopteran chromosome evolution. *Science Advances*, *5*(6). https://doi.org/10.1126/sciadv.aau3648
- Khan, S. M., Hanif, M. U., Khan, A., Hassan, M. U., Javanmardi, A., & Ahmad, A. (2022). Damage assessment of reinforced concrete beams using cost-effective MEMS accelerometers. *Structures*, *41*(March), 602–618. https://doi.org/10.1016/j.istruc.2022.04.101
- Kim, J. T., Ryu, Y. S., Cho, H. M., & Stubbs, N. (2003). Damage identification in beam-type structures: Frequency-based method vs mode-shape-based method. *Engineering Structures*, *25*(1), 57–67. https://doi.org/10.1016/S0141-0296(02)00118-9
- Moreu, F., Li, X., Li, S., & Zhang, D. (2018). Technical specifications of structural health monitoring for highway bridges: New chinese structural health monitoring code. *Frontiers in Built Environment*, *4*(March), 1–12. https://doi.org/10.3389/fbuil.2018.00010
- Pcr, A., & Kit, L. (2012). Data Sheet Data Sheet. 고생물학회지, *31402*(September 2004), 0–1. http://www.papersearch.net/view/detail.asp?detail_key=10000715
- Sohn, H., Farrar, C. R., Hemez, F., & Czarnecki, J. (2001). A Review of structural health. *Library.Lanl.Gov*, 1–7. https://library.lanl.gov/cgi-bin/getfile?00796820.pdf
- Wong, K. Y. (2004). Instrumentation and health monitoring of cable-supported bridges. *Structural Control and Health Monitoring*, *11*(2), 91–124. https://doi.org/10.1002/stc.33
- Zhu, X., Cao, M., Ostachowicz, W., & Xu, W. (2019). Damage identification in bridges by processing dynamic responses to moving loads: Features and evaluation. *Sensors (Switzerland)*, *19*(3). https://doi.org/10.3390/s19030463
- C. R. Farrar and K. Worden, “An introduction to structural health monitoring,” *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 365, no. 1851, pp. 303–315, 2007.
- S. Engineering, “Aalborg Universitet Vibrational Based Inspection of Civil Engineering Structures Rytter , Anders,” 1993.
- “APPLICATION OF ADVANCED NON- DESTRUCTIVE TESTING METHODS ON ANALYSIS,” 2012.

- "Experimental Modal Analysis of Civil Engineering Structures Elsa de Sá Caetano Trabalho RI-7," no. April 2014, 2006.
- E. P. Carden and P. Fanning, "Vibration Based Condition Monitoring: A Review," *Struct. Heal. Monit. An Int. J.*, vol. 3, no. 4, pp. 355–377, 2004.
- K. Worden, C. R. Farrar, G. Manson, and G. Park, "The fundamental axioms of structural health monitoring," *Proc. R. Soc. A Math. Phys. Eng. Sci.*, vol. 463, no. 2082, pp. 1639–1664, 2007.
- J. M. Ko and Y. Q. Ni, "Technology developments in structural health monitoring of large-scale bridges," *Eng. Struct.*, vol. 27, no. 12 SPEC. ISS., pp. 1715–1725, 2005.
- A. Pau and F. Vestroni, "Vibration assessment and structural monitoring of the Basilica of Maxentius in Rome," *Mech. Syst. Signal Process.*, vol. 41, no. 1–2, pp. 454–466, 2013.
- L. A. Lorenzon, I. Civile, and E. Ambientale, "Structural identification of bridges : development of an integrated system for continuous dynamic monitoring . Identificazione strutturale di ponti : sviluppo di un sistema integrato per il monitoraggio dinamico in tempo reale .," 2013.
- A. E. Aktan and J. M. W. Brownjohn, "Structural Identification: Opportunities and Challenges," *J. Struct. Eng.*, vol. 139, no. 10, pp. 1639–1647, 2013.
- L. Zhang, R. Brincker, and P. Andersen, "An Overview of Operational Modal Analysis : Major Development and Issues 1 . Major Developments of OMA," *1St Int. Oper. Modal Anal. Conf.*, no. 1, p. 12, 2005.
- M. Pastor, M. Binda, and T. Harčarik, "Modal assurance criterion," *Procedia Eng.*, vol. 48, pp. 543–548, 2012.

APPENDICES

CODE

Master node

```
#include "time.h"
#include <esp_now.h>
#include <WiFi.h>
#include <Arduino.h>
#include <Adafruit_SPIDevice.h>
#include <Adafruit_I2CDevice.h>
#include <Adafruit_BusIO_Register.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include <Wire.h>
#include <SPI.h>
#include <SdFat.h>
#include <stdio.h>
#include <NTPClient.h>
#include "esp_system.h"
#include "esp_wifi.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/timers.h"
#include "freertos/event_groups.h"
#include "freertos/queue.h"
#include <HttpClient.h>
#include "esp_wpa2.h"
#include <ThingSpeak.h>
```

```
#define uS_TO_S_FACTOR 1000000ULL

#define Sampling_Period 30000

#define Flush_Interval 29000

#define acc_thresh 11

#define time_period 3000000ULL

#define trans_time 120000

#define time_frequency 10800

#define EAP_IDENTITY "srehman.bece18nice" //if connecting from another corporation, use identity@organisation.domain in Eduroam

#define EAP_PASSWORD "*******" //your Eduroam password

const char* ssid = "eduroam"; // Eduroam SSID

const uint16_t intervalTicks = 7;

RTC_NOINIT_ATTR int bootCount;

int time_check;

int sleep_interval;

int start_time=11;

WiFiClient client;

HTTPClient http;

constexpr char WIFI_SSID[] = "eduroam";

const char* serverName = "https://8064-111-68-97-201.in.ngrok.io/upload";

String parameter = "hey";

char buffer[1000];

char filename[15];

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP, "pool.ntp.org", 18000);

// REPLACE WITH YOUR ESP RECEIVER'S MAC ADDRESS

uint8_t broadcastAddress1[] = {0x9C,0x9C,0x1F,0xE9,0x5A,0x40};

uint8_t broadcastAddress2[] = {0x84,0xCC,0xA8,0x54,0x11,0xE4};


int32_t getWiFiChannel(const char *ssid) {
```

```
  if (int32_t n = WiFi.scanNetworks()) {
      for (uint8_t i=0; i<n; i++) {
          if (!strcmp(ssid, WiFi.SSID(i).c_str())) {
              return WiFi.channel(i);
          }
      }
  }
  return 0;
}
typedef struct check_struct {
  int x;
} check_struct;
check_struct check;
esp_now_peer_info_t peerInfo;
// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  Serial.print("Packet to: ");
  // Copies the sender mac address to a string
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print(macStr);
  Serial.print(" send status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}
//Use ESP32 duo core
const int TaskCore1  = 1;
const int TaskCore0 = 0;
//----------------------------------------------------------------------------
int LED_BUILTIN = 2;
```

```
// SD file definitions
const uint8_t sdChipSelect = 5;
SdFat sd;
SdFile file;
File logfile;
//------------------------------------------------------------------------------
//Task Handles
TaskHandle_t task1_handle=NULL;
TaskHandle_t task2_handle=NULL;
TaskHandle_t task3_handle=NULL;
//------------------------------------------------------------------------------
// Queue definitions
// data type for Queue item
struct Data_t {
  uint32_t usec;
  float valueZ;
} xData_t;
//Declare Queue data type for FreeRTOS
QueueHandle_t DataQueue = NULL;
// Accel ADXL_345 definitions, SPI
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
//------------------------------------------------------------------------------
// define two tasks for Sensor Data and SD Write
void TaskGetData( void *pvParameters );
void TaskSDWrite( void *pvParameters );
void TaskSDFlush( void *pvParameters );
//------------------------------------------------------------------------------
void sendData(String data){
  if(WiFi.status()== WL_CONNECTED){
     http.begin(client, serverName);
```

```
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    String payload = "data=";
    int httpResponseCode = http.POST(payload + parameter);

    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
    http.end();
   }
   else {
    Serial.println("WiFi Disconnected");
   }
}
int counter = 0;
void setup()
{
  Serial.begin(115200);
  // Connect to Wi-Fi
  int32_t channel = getWiFiChannel(WIFI_SSID);
  WiFi.printDiag(Serial); // Uncomment to verify channel number before
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
  esp_wifi_set_promiscuous(false);
  WiFi.printDiag(Serial); // Uncomment to verify channel change after
  Serial.print("Connecting to network: ");
  Serial.println(ssid);
  WiFi.disconnect(true);  //disconnect form wifi to set new wifi connection
  WiFi.mode(WIFI_STA); //init wifi mode
  esp_wifi_sta_wpa2_ent_set_identity((uint8_t   *)EAP_IDENTITY,   strlen(EAP_IDENTITY));
//provide identity
```

```
  esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EAP_IDENTITY, strlen(EAP_IDENTITY)); //provide username --> identity and username is same
  esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EAP_PASSWORD, strlen(EAP_PASSWORD)); //provide password
  esp_wpa2_config_t config = WPA2_CONFIG_INIT_DEFAULT(); //set config settings to default
  esp_wifi_sta_wpa2_ent_enable(&config); //set config settings to enable function
  WiFi.begin(ssid); //connect to wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    counter++;
    if(counter>=60){ //after 30 seconds timeout - reset board
      ESP.restart();
    }
  }
  if(bootCount!=0)
  {
  Serial.println("Initialization!");
  bootCount= 0;
  timeClient.update();
  int hours_now=timeClient.getHours();
  int minutes_now=timeClient.getMinutes();
  Serial.println(hours_now);
  Serial.println(minutes_now);
  time_check=start_time-hours_now;
  time_check=abs(time_check);
   if (time_check!=0)
   {
     int minutes_now=timeClient.getMinutes();
     int minutes_rem=minutes_now-60;
```

```
  minutes_rem=abs(minutes_rem);
  if (time_check==1)
  {
    sleep_interval=(minutes_rem*60);
    Serial.println("Less than 1 hour remaining.");
    Serial.println(sleep_interval);
  }
  else
  {
    sleep_interval=(time_check*3600)+(minutes_rem*60);
    Serial.println("More than 1 hour remaining.");
    Serial.println(sleep_interval);
  }
  Serial.println(sleep_interval);
  Serial.println("\nEntering deep sleep.");
  esp_sleep_enable_timer_wakeup(sleep_interval * uS_TO_S_FACTOR);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
 }
 }
Serial.println("Boot count=");
Serial.print(bootCount);
if (esp_now_init() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
}
esp_now_register_send_cb(OnDataSent);
// register peer
peerInfo.channel = 0;
```

```
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  pinMode (LED_BUILTIN, OUTPUT);
  //ACCEL Setup and RUN
    if(!accel.begin()){
    while(1);
    }
//Setting acceleration range and data rate
  accel.setRange(ADXL345_RANGE_4_G);
  accel.setDataRate(ADXL345_DATARATE_800_HZ);
//SD                                        CARD                                        SETUP
============================================================================
// see if the card is present and can be initialized
if (!sd.begin(sdChipSelect, SD_SCK_MHZ(15))) {
  Serial.println("Card init. failed!");
  //error(2);
}
  //Create                                    filename                                    scheme
============================================================================
```

```
  strcpy(filename, "/DATA00.CSV");
  for (uint8_t i = 0; i < 100; i++) {
    filename[5] = '0' + i/10;
    filename[6] = '0' + i%10;
    // create if does not exist, do not open existing, write, sync after write
    if (! sd.exists(filename)) {
      break;
    }
  }
//Create file and prepare it
============================================================
  logfile = sd.open(filename, O_CREAT | O_WRITE );
  if( ! logfile ) {
    Serial.print("Couldnt create ");
    Serial.println(filename);
  }
  Serial.print("Writing to ");
  Serial.println(filename);
  Serial.println("Ready!");
   //Queue Setup
  DataQueue = xQueueCreate(5000, sizeof( &xData_t ));
  if(DataQueue == NULL){
     Serial.println("Error Creating the Queue");
   }
//Setup up Tasks and where to run
============================================================
  xTaskCreatePinnedToCore(
    TaskGetData
    ,  "Get Data from Accel to Queue"
    ,  2048
```

```
  , NULL
  , 3
  , &task1_handle
  , TaskCore0);

xTaskCreatePinnedToCore(
  TaskSDWrite
  , "Get Data from Queue"
  , 2048 // Stack size
  , NULL
  , 2  // Priority
  , &task2_handle
  , TaskCore1);

xTaskCreatePinnedToCore(
  TaskSDFlush
  , "Write Data to Card"
  , 1024 // Stack size
  , NULL
  , 1  // Priority
  , &task3_handle
  , TaskCore1);

vTaskSuspend(task1_handle);
Serial.print("\nTask 1 suspended(Initialization).");
vTaskSuspend(task2_handle);
Serial.print("\nTask 2 suspended(Initialization).");
vTaskSuspend(task3_handle);
Serial.print("\nTask 3 suspended(Initialization).");
delay(1000);
```

```
}
void loop()
{
  for(;;)
 {
  check.x=1;
  timeClient.update();
  int minutes_now=timeClient.getMinutes();
  unsigned long time_start=millis();
  unsigned long time_elasped=0;
  while(time_elasped<time_period)
  {
    sensors_event_t event;
    accel.getEvent(&event);
    float acc_z=event.acceleration.z;
    Serial.println(acc_z);
     if(abs(acc_z)>acc_thresh)
      {
        Serial.println("Acceleration triggered");
        timeClient.update();
        int minutes_now=timeClient.getMinutes();
        int min_rem_1=minutes_now-60;
        min_rem_1=abs(min_rem_1);
        int                                    sleep_interval_1=min_rem_1*60-Sampling_Period/1000-
trans_time/1000+time_frequency;
        esp_err_t result = esp_now_send(0, (uint8_t *) &check, sizeof(check_struct));
          if (result == ESP_OK)
          {
            Serial.println("Sent with success");
          }
```

```
 else
 {
 Serial.println("Error sending the data");
 }

digitalWrite(LED_BUILTIN, HIGH);
vTaskResume(task1_handle);
Serial.print("\nTask 1 resumed(Accel. triggered).\n");
vTaskResume(task2_handle);
Serial.print("\nTask 2 resumed(Accel. triggered).\n");
vTaskResume(task3_handle);
Serial.print("\nTask 3 resumed(Accel. triggered).\n");
vTaskDelay(Sampling_Period);
vTaskSuspend(task1_handle);
Serial.print("\nTask 1 suspended.\n");
vTaskSuspend(task2_handle);
Serial.print("\nTask 2 suspended.\n");
vTaskSuspend(task3_handle);
Serial.print("\nTask 3 suspended.\n");
digitalWrite(LED_BUILTIN, LOW);
vTaskDelay(trans_time);
logfile = sd.open(filename); // | O_TRUNC)
Serial.print(logfile);
if (logfile)
{
   while (logfile.position() < logfile.size())
   {
   int bytesRead = logfile.readBytes(buffer, sizeof(buffer));
   parameter.concat(buffer);
   sendData(parameter);
```

```
          parameter = "";
          }
         logfile.close();
       }
      Serial.println(sleep_interval_1);
      Serial.println("\nEntering deep sleep.");
      esp_sleep_enable_timer_wakeup(sleep_interval_1 * uS_TO_S_FACTOR);
      Serial.flush();
      esp_deep_sleep_start();
      Serial.println("This will never be printed");
     }
   vTaskDelay(100);
   unsigned long time_now=millis()+(minutes_now*60000);
   time_elasped=time_now-time_start;
   //Serial.println(time_elasped);
 }
 break;
}
unsigned long sleep_interval_2=599-Sampling_Period/1000-trans_time/1000+time_frequency;
esp_err_t result = esp_now_send(0, (uint8_t *) &check, sizeof(check_struct));
        if (result == ESP_OK)
        {
          Serial.println("Sent with success");
        }
        else
        {
        Serial.println("Error sending the data");
        }
      digitalWrite(LED_BUILTIN, HIGH);
      vTaskResume(task1_handle);
```

```
Serial.print("\nTask 1 resumed(Time triggered).\n");
vTaskResume(task2_handle);
Serial.print("\nTask 2 resumed(Time triggered).\n");
vTaskResume(task3_handle);
Serial.print("\nTask 3 resumed(Time triggered).\n");
vTaskDelay(Sampling_Period);
vTaskSuspend(task1_handle);
Serial.print("\nTask 1 suspended.\n");
vTaskSuspend(task2_handle);
Serial.print("\nTask 2 suspended.\n");
vTaskSuspend(task3_handle);
Serial.print("\nTask 3 suspended.\n");
digitalWrite(LED_BUILTIN, LOW);
vTaskDelay(trans_time);
logfile = sd.open(filename);
      if (logfile)
{
   while (logfile.position() < logfile.size())
   {
   int bytesRead = logfile.readBytes(buffer, sizeof(buffer));
   parameter.concat(buffer);
   sendData(parameter);
   parameter = "";
   }
 logfile.close();
}
Serial.println(sleep_interval_2);
Serial.println("\nEntering deep sleep for"+String(sleep_interval_2));
esp_sleep_enable_timer_wakeup(sleep_interval_2 * uS_TO_S_FACTOR);
Serial.flush();
```

```
      esp_deep_sleep_start();
      Serial.println("This will never be printed");
 }
void TaskGetData(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  struct Data_t *pxPointerToxData_t;

  for (;;) // A Task shall never return or exit.
  {
    pxPointerToxData_t = &xData_t;

    if(xQueueSend( DataQueue, (void *) &pxPointerToxData_t, 10 ) != pdPASS )
//portMAX_DELAY
      {
        Serial.println("xQueueSend is not working");
      }

    sensors_event_t event;
    accel.getEvent(&event);
    pxPointerToxData_t->usec = millis();
    pxPointerToxData_t->valueZ = event.acceleration.z;
    Serial.print(pxPointerToxData_t->usec);
    Serial.print(',');

    Serial.print(pxPointerToxData_t->valueZ,5);
    Serial.println();
```

```
    vTaskDelay(intervalTicks);  // one tick delay (1000 uSec/1 mSec) in between reads for 1000 Hz reading
  }
}
//--------------------------------------------------------------------------
void TaskSDWrite(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
  struct Data_t xData_RCV, *pxData_RCV;

  for (;;)
  {

    if(DataQueue != NULL )
    {
      if( xQueueReceive( DataQueue, &( pxData_RCV ), 1000 ) != pdPASS )  //portMAX_DELAY
      {
        Serial.println("xQueueRecieve is not working");
      }
        logfile.print(pxData_RCV->usec);
        logfile.print(',');
        logfile.print(pxData_RCV->valueZ,5);
        logfile.println();
    }
   }
}
//--------------------------------------------------------------------------
void TaskSDFlush(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
```

```
  for (;;)
  {
    logfile.flush();
    vTaskDelay(Flush_Interval);
    Serial.println("Flushed file");
  }
}
```

Slave node

```
#include "time.h"
#include <esp_now.h>
#include <WiFi.h>
#include <Arduino.h>
#include <Adafruit_SPIDevice.h>
#include <Adafruit_I2CDevice.h>
#include <Adafruit_BusIO_Register.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include <Wire.h>
#include <SPI.h>
#include <SdFat.h>
#include <stdio.h>
#include <NTPClient.h>
#include "esp_system.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/timers.h"
#include "freertos/event_groups.h"
```

```
#include "freertos/queue.h"
#include <HttpClient.h>
#include "esp_wpa2.h"

#define uS_TO_S_FACTOR 1000000ULL
#define Sampling_Period 30000
#define Flush_Interval 29000
#define time_period 3000000ULL
#define trans_delay 60000
#define trans_time 120000
#define time_frequency 10800
#define EAP_IDENTITY "srehman.bece18nice" //if connecting from another corporation, use identity@organisation.domain in Eduroam
#define EAP_PASSWORD "student.123" //your Eduroam password
const char* ssid = "eduroam"; // Eduroam SSID
unsigned long millis_start=millis();
const uint16_t intervalTicks = 7;
WiFiClient client;
HTTPClient http;
const char* serverName = "http://5eb3-111-68-97-200.ngrok.io/uploads";
String parameter;
char buffer[1000];
char filename[15];
RTC_NOINIT_ATTR int bootCount;
int start_time=11;
int time_check;
int sleep_interval;
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "3.pk.pool.ntp.org", 18000);
//Structure example to receive data
```

```
typedef struct check_struct {
  int x;
} check_struct;
//Create a struct_message called myData
check_struct myData;
//Use ESP32 duo core
const int TaskCore1  = 1;
const int TaskCore0 = 0;
//--------------------------------------------------------------------------
int LED_BUILTIN = 2;
// SD file definitions
const uint8_t sdChipSelect = 5;
SdFat sd;
SdFile file;
File logfile;
//--------------------------------------------------------------------------
//Task Handles
TaskHandle_t task1_handle=NULL;
TaskHandle_t task2_handle=NULL;
TaskHandle_t task3_handle=NULL;
//--------------------------------------------------------------------------
// Queue definitions
// data type for Queue item
struct Data_t {
  uint32_t usec;
  float valueZ;
} xData_t;
//Declare Queue data type for FreeRTOS
QueueHandle_t DataQueue = NULL;
```

```
//--------------------------------------------------------------------------------
// Accel ADXL_345 definitions, SPI
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
//--------------------------------------------------------------------------------
// define two tasks for Sensor Data and SD Write
void TaskGetData( void *pvParameters );
void TaskSDWrite( void *pvParameters );
void TaskSDFlush( void *pvParameters );
//--------------------------------------------------------------------------
void sendData(String data){
  if(WiFi.status()== WL_CONNECTED){
      http.begin(client, serverName);
      http.addHeader("Content-Type", "application/x-www-form-urlencoded");
      String payload = "dataSlave=";
      int httpResponseCode = http.POST(payload + parameter);

      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      http.end();
    }
    else {
      Serial.println("WiFi Disconnected");
    }
}
//--------------------------------------------------------------------------------
//callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.print("Bytes received: ");
```

```
Serial.println(len);
Serial.print("x: ");
Serial.println(myData.x);
Serial.println();

timeClient.update();
int minutes_now=timeClient.getMinutes();
int min_rem_1=minutes_now-55;
min_rem_1=abs(min_rem_1);
int sleep_interval_1=min_rem_1*60-Sampling_Period/1000-trans_delay/1000+time_frequency;

digitalWrite(LED_BUILTIN, HIGH);
vTaskResume(task1_handle);
Serial.print("\nTask 1 resumed.\n");
vTaskResume(task2_handle);
Serial.print("\nTask 2 resumed.\n");
vTaskResume(task3_handle);
Serial.print("\nTask 3 resumed.\n");
vTaskDelay(Sampling_Period);
vTaskSuspend(task1_handle);
Serial.print("\nTask 1 suspended.\n");
vTaskSuspend(task2_handle);
Serial.print("\nTask 2 suspended.\n");
vTaskSuspend(task3_handle);
Serial.print("\nTask 3 suspended.\n");
vTaskDelay(1);
digitalWrite(LED_BUILTIN, LOW);
Serial.println("Waiting for transmission");
vTaskDelay(trans_delay);
Serial.println("Wait over!");
```

```
  logfile = sd.open(filename); // | O_TRUNC);
  Serial.print(logfile);
  if (logfile)
       {
          while (logfile.position() < logfile.size())
          {
          int bytesRead = logfile.readBytes(buffer, sizeof(buffer));
          parameter.concat(buffer);
          sendData(parameter);
          parameter = "";
          }
        logfile.close();
       }
  Serial.println(sleep_interval_1);
  Serial.println("\nEntering deep sleep.");
  esp_sleep_enable_timer_wakeup(sleep_interval_1 * uS_TO_S_FACTOR);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}
int counter = 0;
void setup()
{
  Serial.begin(115200);
  //Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  WiFi.setSleep(WIFI_PS_NONE);
  Serial.print("Connecting to network: ");
  Serial.println(ssid);
  WiFi.disconnect(true);  //disconnect form wifi to set new wifi connection
```

```
  esp_wifi_sta_wpa2_ent_set_identity((uint8_t *)EAP_IDENTITY, strlen(EAP_IDENTITY)); //provide identity
  esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EAP_IDENTITY, strlen(EAP_IDENTITY)); //provide username --> identity and username is same
  esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EAP_PASSWORD, strlen(EAP_PASSWORD)); //provide password
  esp_wpa2_config_t config = WPA2_CONFIG_INIT_DEFAULT(); //set config settings to default
  esp_wifi_sta_wpa2_ent_enable(&config); //set config settings to enable function
  WiFi.begin(ssid); //connect to wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    counter++;
    if(counter>=60){ //after 30 seconds timeout - reset board
      ESP.restart();
    }
  }
Serial.print("Station IP Address: ");
Serial.println(WiFi.localIP());
Serial.print("Wi-Fi Channel: ");
Serial.println(WiFi.channel());
Serial.println("Boot count="+String(bootCount));
  if(bootCount!=0)
  {
  Serial.println("Initialization!");
  bootCount= 0;
  timeClient.update();
  int hours_now=timeClient.getHours();
  int minutes_now=timeClient.getMinutes();
  Serial.println(hours_now);
```

```
Serial.println(minutes_now);
time_check=start_time-hours_now;
time_check=abs(time_check);
 if (time_check!=0)
 {
  int minutes_now=timeClient.getMinutes();
  int minutes_rem=minutes_now-60;
  minutes_rem=abs(minutes_rem);
  if (time_check==1)
  {
   sleep_interval=(minutes_rem*60);
   Serial.println("Less than 1 hour remaining.");
   Serial.println(sleep_interval);
  }
  else
  {
   sleep_interval=(time_check*3600)+(minutes_rem*60);
   Serial.println("More than 1 hour remaining.");
   Serial.println(sleep_interval);
  }
  bootCount=0;
  Serial.println(sleep_interval);
  Serial.println("\nEntering deep sleep.");
  esp_sleep_enable_timer_wakeup(sleep_interval * uS_TO_S_FACTOR);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
 }
//Init ESP-NOW
if (esp_now_init() != ESP_OK) {
```

```
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(OnDataRecv);
  pinMode (LED_BUILTIN, OUTPUT);
 //ACCEL Setup and RUN
    if(!accel.begin()){
    while(1);
    }
//Setting acceleration range and data rate
  accel.setRange(ADXL345_RANGE_4_G);
  accel.setDataRate(ADXL345_DATARATE_800_HZ);
//                          SD                          CARD                          SETUP
//============================================================================
// see if the card is present and can be initialized
if (!sd.begin(sdChipSelect, SD_SCK_MHZ(15))) {
  Serial.println("Card init. failed!");
  //error(2);
}
//Create                                    filename                                    scheme
//============================================================================
  strcpy(filename, "/DATA00.CSV");
  for (uint8_t i = 0; i < 100; i++) {
    filename[5] = '0' + i/10;
    filename[6] = '0' + i%10;
    // create if does not exist, do not open existing, write, sync after write
    if (! sd.exists(filename)) {
      break;
```

```
  }
 }
// Create file and prepare it
=============================================================
 logfile = sd.open(filename, O_CREAT | O_WRITE ); // | O_TRUNC);
 if( ! logfile ) {
  Serial.print("Couldnt create ");
  Serial.println(filename);
 }
 Serial.print("Writing to ");
 Serial.println(filename);
 Serial.println("Ready!");
 //Queue Setup
 DataQueue = xQueueCreate(5000, sizeof( &xData_t ));
 if(DataQueue == NULL){
   Serial.println("Error Creating the Queue");
  }
//Setup up Tasks and where to run
=============================================================
 xTaskCreatePinnedToCore(
  TaskGetData
  , "Get Data from Accel to Queue"
  , 2048
  , NULL
  , 3
  , &task1_handle
  , TaskCore0);

 xTaskCreatePinnedToCore(
  TaskSDWrite
```

```
    , "Get Data from Queue"
    , 2048 // Stack size
    , NULL
    , 2  // Priority
    , &task2_handle
    , TaskCore1);

  xTaskCreatePinnedToCore(
    TaskSDFlush
    , "Write Data to Card"
    , 1024 // Stack size
    , NULL
    , 1  // Priority
    , &task3_handle
    , TaskCore1);

  delay(1);
  vTaskSuspend(task1_handle);
  Serial.print("\nTask 1 suspended(Initialization).\n");
  vTaskSuspend(task2_handle);
  Serial.print("\nTask 2 suspended(Initialization).\n");
  vTaskSuspend(task3_handle);
  Serial.print("\nTask 3 suspended(Initialization).\n");
  delay(3000);
}
}

void loop()
{
for(;;)
```

```
{
unsigned long sleep_interval_2;
timeClient.update();
int minutes_now=timeClient.getMinutes();
unsigned long time_start=millis();
unsigned long time_elasped=0;
while(time_elasped<time_period)
{
Serial.println("Waiting");
vTaskDelay(100);
unsigned long time_now=millis()+(minutes_now*60000);
time_elasped=time_now-time_start;
}
break;
}
 unsigned long sleep_interval_2=598-Sampling_Period/1000-trans_time/1000-
trans_delay/1000+time_frequency;
 digitalWrite(LED_BUILTIN, HIGH);
 vTaskResume(task1_handle);
 Serial.print("\nTask 1 resumed.\n");
 vTaskResume(task2_handle);
 Serial.print("\nTask 2 resumed.\n");
 vTaskResume(task3_handle);
 Serial.print("\nTask 3 resumed.\n");
 vTaskDelay(Sampling_Period);
 vTaskSuspend(task1_handle);
 Serial.print("\nTask 1 suspended.\n");
 vTaskSuspend(task2_handle);
 Serial.print("\nTask 2 suspended.\n");
 vTaskSuspend(task3_handle);
```

```
  Serial.print("\nTask 3 suspended.\n");
  digitalWrite(LED_BUILTIN, LOW);
  vTaskDelay(trans_delay);
  logfile = sd.open(filename);
  Serial.print(logfile);
   if (logfile)
    {
          while (logfile.position() < logfile.size())
          {
          int bytesRead = logfile.readBytes(buffer, sizeof(buffer));
          parameter.concat(buffer);
          sendData(parameter);
         //  Serial.print("hmm");
          parameter = "";
          }
          logfile.close();
    }
  Serial.println(sleep_interval_2);
  Serial.println("\nEntering deep sleep.");
  esp_sleep_enable_timer_wakeup(sleep_interval_2 * uS_TO_S_FACTOR);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");
}
void TaskGetData(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  struct Data_t *pxPointerToxData_t;
```

```
  for (;;) // A Task shall never return or exit.
  {
    pxPointerToxData_t = &xData_t;

    if(xQueueSend( DataQueue, (void *) &pxPointerToxData_t, 10 ) != pdPASS ) //portMAX_DELAY
      {
        Serial.println("xQueueSend is not working");
      }

    sensors_event_t event;
    accel.getEvent(&event);
    pxPointerToxData_t->usec = millis();
    pxPointerToxData_t->valueZ = event.acceleration.z;
    Serial.print(pxPointerToxData_t->usec);
    Serial.print(',');

    Serial.print(pxPointerToxData_t->valueZ,5);
    Serial.println();

    vTaskDelay(intervalTicks);  // one tick delay (1000 uSec/1 mSec) in between reads for 1000 Hz reading
  }
}
//---------------------------------------------------------------------------
void TaskSDWrite(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
  struct Data_t xData_RCV, *pxData_RCV;
```

```
  for (;;)
  {

    if(DataQueue != NULL )
    {
      if( xQueueReceive( DataQueue, &( pxData_RCV ), 1000 ) != pdPASS )  //portMAX_DELAY
      {
        Serial.println("xQueueRecieve is not working");
      }
        logfile.print(pxData_RCV->usec);
        logfile.print(',');
        logfile.print(pxData_RCV->valueZ,5);
        logfile.println();
    }
   }
}
//-------------------------------------------------------------------------------
void TaskSDFlush(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  for (;;)
  {
    logfile.flush();
    vTaskDelay(Flush_Interval);
    Serial.println("Flushed file");

  }
}
```

Python (Post processing)

```
import pandas as pd
import matplotlib.pyplot as plt
from numpy.fft import fft, ifft
import numpy as np
from scipy import signal
import time

while True:
 df = pd.read_csv('samples.csv')
 Y=df['Accel']
 detrended=signal.detrend(Y,axis=-1,type='linear',bp=0,overwrite_data=False)
 plt.figure(figsize = (12, 6))
 plt.plot(df['Time'], detrended)
 plt.xlabel('Time')
 plt.ylabel('Detrended Accel')
 plt.savefig('accel.png', bbox_inches='tight')
 plt.show()
 w=0.5
 b, a = signal.butter(5, w, 'low',analog=False)
 output = signal.filtfilt(b, a, detrended)
 X=fft(output)
 N = len(Y)
 n = np.arange(N)
 # get the sampling rate
 sr = 128
 T = N/sr
 freq = n/T
 # Get the one-sided specturm
```

```
n_oneside = N//2
# get the one side frequency
f_oneside = freq[:n_oneside]
print(n_oneside)
print(f_oneside)
plt.figure(figsize = (12, 6))
plt.plot(f_oneside, np.abs(X[:n_oneside]), 'g')
plt.ylim( [ 0, 100 ] )
plt.xlabel('Freq (Hz)')
plt.ylabel('FFT Amplitude |X(freq)|')
plt.savefig('freq.png', bbox_inches='tight')
plt.show()
time.sleep(86400)
```

MATLAB (Post processing)

```
%Butterworth filter coefficents
[b,a] = butter(2,0.5);
%Acceleration data input
accel=readmatrix('DATA08.xlsx','Sheet','DATA08','Range','C1:C3600');
time=readmatrix('DATA08.xlsx','Sheet','DATA08','Range','A1:A3600');
%Detrending to remove gravity bias
ufaccel=detrend(accel);
%Filtering to remove noise
faccel = filter(b,a,ufaccel);
%Sample interval
dt=0.008;
%Sampling rate
fs=128;
```

```
nfft=1024;
f=linspace(0,fs,nfft);
%FFT
freq=abs(fft(faccel,nfft));
%Plotting acceleration vs. time
figure
plot(time,ufaccel);
xlabel('Time');
ylabel('Acceleration');
hold on
plot(time,faccel);
hold off
%Plotting frequency vs. amplitude
figure
xlabel('Frequency');
ylabel('Amplitude');
plot(f(1:nfft/2),freq(1:nfft/2));
```