Griffith School of Engineering

Griffith University

6007ENG – Industry Affiliates Program

Structural Health Monitoring of the Griffith Footbridge Using LoRaWAN Technology

Jessy Barber, s5138877

12th June, Trimester 1, 2023

Griffith University Internal Project

Yong Zhu

A report submitted in partial fulfilment of the degree of Bachelor of Engineering (Honours)

EXECUTIVE SUMMARY

This research presents a novel approach for Structural Health Monitoring (SHM) of the Griffith University footbridge, harnessing the power of LoRaWAN technology within the framework of the Internet of Things (IoT). The study is undertaken in two phases.

The first phase focuses on the software development and testing of the initial three layers of the IoT architecture: coding, perception and network. This involves the utilisation of two Arduino MKRWAN 1300 devices, where one serves as a LoRa node and the other as a LoRaWAN gateway. An ADXL335 3-axis accelerometer is integrated into the system to capture vibrational data, which is processed and visualized through Python scripts.

The second phase extends the reach of the IoT architecture to incorporate middleware and application layers. It also includes hardware modifications like transitioning to a custom PCB and a 3D printed enclosure. The data captured is transmitted via a LoRaWAN connection to a WisGate Edge Lite 2 gateway and then relayed to the TNN cloud. Data visualization and analysis are facilitated using the Arduino IoT Cloud dashboard.

Each phase is guided by defined objectives, including hardware design and testing, software validation, and comprehensive field tests. The overall outcome is a potential new standard for bridge health monitoring using LoRaWAN technology, as exemplified by the Griffith University footbridge.

ACKNOWLEDGEMENTS

I would like this opportunity to acknowledge and thank the following people, without whom I would not be able to complete this project:

My mother and father, for always supporting me every step of the way on my academic journey. For supporting me both mentally and financially during my studies, I thank you deeply.

My group of friends, for respecting the time required for my studies, and always being there to elevate my spirits.

My fellow peers at Griffith who I have met along the way, I thank you for being patient with me and always providing a fun learning experience.

My work staff and bosses, for allowing me to take time off from my dream job to complete my degree, and allowing me to come back full time the moment I finish.

And finally I would like to thank my supervisor Yong Zhu and project partner Huiyue Qiao for their countless hours in meetings and extensive laboratory testing. I hope you enjoyed the project as much as I did.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Report Structure

The report is organized in the following structure; first relevant background information regarding measuring the health of a bridge, WSNs, LoRa and LoRaWAN, and the internet of things will be presented. Following this, the aims and objectives of this report will be outlined. Next, a review of the relevant published literature will be evaluated. The design specifications and methodology for developing protoype one and two will be provided. Results from testing both prototype implementations will be presented and a discussion regarding these results and performance will be provided. Device limitations will be analysed, suggestions for further improvements will be discussed followed by a conclusion. Proceeding the conclusion will be the references and appendices.

1.2. Project Background

1.2.1. Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are simple, low-cost networks that primarily consist of nodes and a base station [8]. WSN nodes usually comprise of some sensing or measuring capability acting as the physical layer, and relay this information via uplink to a base station for processing and then to a network server acting as the network layer. From here, the API from a network cloud service can be used to create GUI's and other applications for researchers and consumers which acts as the application layer.

Innovating many field of industry and research, these distributed networks of nodes have been valuable in many contexts. For example, the use of ZigBee communication technology for air pollution monitoring [9] and the use of Bluetooth for communication between end-devices measuring temperature, luminance, carbon dioxide and humidity for energy-saving establishments [10]. Although these WSNs have worked in the past, the future of this technology lies in developing systems that have high scalability and range, something that ZigBee and Bluetooth inherently lack. Cellular and satellite technology are alternate approaches that offer extremely high data rates and range, however these technologies are not practical to implement in most situations due to exceedingly high costs.

1.2.2. Structural Health Monitoring

Structural Health Monitoring (SHM) is a vital practice for ensuring the safety and longevity of civil and industrial structures [11]. SHM involves continuously tracking change in

structures which can be attributed to material aging, environmental influences or unforeseen incidents such as traffic accidents or natural disasters. These changes can be tracked using WSNs equipped with appropriate sensor modules, and integrating data transmission capabilities. A survey investigating the implementation of IoT technology for structural health monitoring (SHM) determined that WSN technology has revolutionized the health monitoring in various fields including civil engineering [12]. WSN systems can be deployed to measures a vast array of SHM indicators including temperature, velocity, acceleration, frequency and displacement. WSN can be deployed on a structure such as a bridge operating as Internet of Things (IoT) nodes. This deployment highlights the advantage of using WSNs for SHM since the collected data can be uploaded to the cloud for processing and distribution. Within the context of bridge monitoring, the integration of WSN with IoT for SHM can serve various application requirements for real-time data uplink such as monitoring acceleration and frequency characteristics. This data can be plotted on a continuous time spectrum and compared to observational data such as pedestrian load to verify the validity of simulated truss analysis models and finite elements (FE) simulation.

1.2.3. Internet of Things

The Internet of Things (IoT) is an 'interconnected network of things' [13], where 'things' in this context is defined as an end-device with WSN type capability. The IoT architecture comprises of six-layers, the coding layer, perception layer, network layer, middle-ware layer, application layer and business layer [13]. Thus to create this IoT architecture for research purposes the first five layers need to be implemented. LoRa end-devices act as the physical layer encompassing the coding and perception layer. The coding layer involves associating unique ID specifiers to each end device [13] and the perception layer is involved with on-board sensing and data acquisition. The network layer is a relay of this perceptual information to a gateway, and the middle-ware layer is the IoT cloud platform that facilitates these connections and receives information from the network layer. The application layer involves pulling the API or information from the network layer and developing apps or graphical user interfaces (GUIs) to display the data.

The Things Network (TNN) is an open-source LoRaWAN network server used to construct IoT cloud applications with end-to-end encryption and secure communication [14]. TNN exists on the middle-ware layer and can be used to deploy an IoT architecture using LoRa end-devices in the coding and perception layer, and utilize the LoRaWAN communication protocol in the network layer. TNN offers a console and API to develop applications that serve as the architecture's application layer. Figure 1.1 displays an example of an IoT architecture using WSN nodes, a LoRaWAN gateway, cloud storage and user devices.

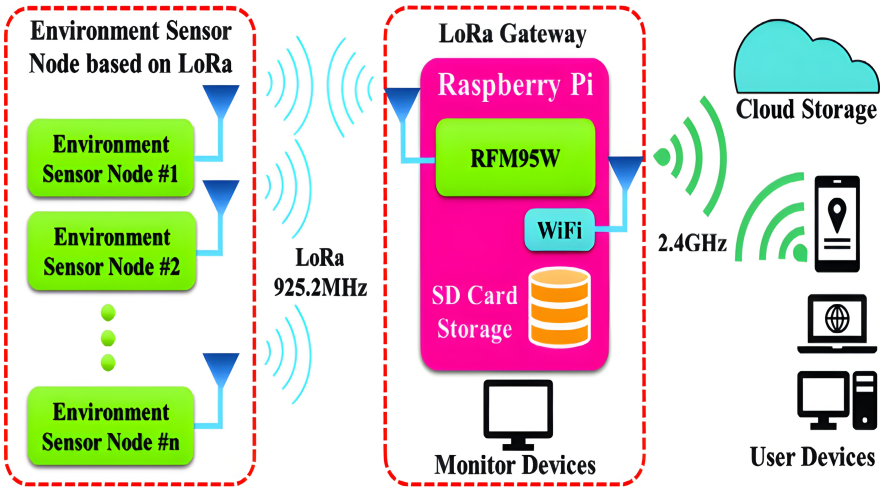


Fig. 1.1. The LoRa network architecture for agriculture area. [1]

1.2.4. LoRa and LoRaWAN

Low Power Wide Area Network (LPWAN) is a communication technology that offers wide coverage, similar to satellite networks, while maintaining lower data rates akin to ZigBee. The technology is distinguished by its ultra-low power consumption and cost-effective deployment and maintenance [15].

LoRa and LoRaWAN, forms of LPWAN technology, were developed to overcome the scalabilty issues associated with traditional WSN configurations that relied on short-range communication protocols such as Zigbee and Bluetooth [8]. These configurations often used a mesh network layout, which introduced challenges in network management and power consumption with increasing network size [15].

LoRaWAN's unique 'star of stars' configuration addresses these challenges by enabling scalable network expansion with reduced complexity. LoRa itself is a Chirp Spread Spectrum (CSS) modulation technique developed by Cycleo, offering a Medium Access Control (MAC) protocol and operating on license-free, region-dependent Industrial, Scientific and Medical (ISM) frequency bands [15].

1.3. Aims and Objectives

The primary goal of this research is to establish an IoT-driven Structural Health Monitoring (SHM) system for Griffith University's footbridge. This project aims to develop, deploy, and test two iterative prototypes designed to monitor the bridge's health by assessing key indicators such as vibration frequency and acceleration. These prototypes incorporate the first five layers of the IoT architecture: coding, perception, network, middleware, and application layers.

The development process for these two prototypes is progressive, with each prototype

incorporating increasing complexity in components and software to ensure the efficient operation of the final system.

1.3.1. Prototype 1: Testing IoT Layers 1-3 and Software

The first prototype primarily tests the software and the first three layers of the IoT architecture. This stage uses two Arduino MKRWAN1300 devices, one functioning as a LoRa node (layers one and two) and the other as a pseudo LoRaWAN gateway (layer 3). An ADXL335 3-axis accelerometer collects vibration data from the beam along the z-axis. This data is processed at the node, with the maximum frequency peak and acceleration values identified and transmitted via LoRa to the gateway. A Python script logs the received data, which is then visualized using a plotting script.

Objectives for Prototype 1 include:

1. Validation of the MKRWAN devices' functionality and their compatibility with the LoRaWAN protocol.
2. Integration of an accelerometer into the node and development of a software module to accurately sample and digitize raw acceleration values.
3. Creation of a software algorithm to conduct FFT analysis on these discrete values, and identification and logging of the peak frequency.
4. Development of software capable of accurately logging and plotting the raw acceleration data from the node.
5. Successful transmission and reception of maximum acceleration and peak frequency data between the node and the gateway with a focus on data accuracy and minimal packet loss.
6. Design and implementation of software for logging and visual display of the maximum acceleration and peak frequency data received from the gateway.
7. Determination of the maximum effective range for LoRa packet transmission between the node and the gateway and identification of factors potentially affecting the range.

1.3.2. Prototype 2: Implementing IoT Layers 4-5 and Hardware Deployment

The second prototype further develops the system by introducing the middleware and application layers. It replaces the breadboard with a custom PCB and encloses the setup within a 3D printed enclosure for durability and protection. The MKRWAN device operates as the LoRa node, while a RAK7268 WisGate Edge Lite 2 serves as the LoRaWAN gateway. The node samples the bridge's acceleration and transmits the average maximum acceleration and frequency to the gateway. The gateway then forwards the data to the TNN cloud (middleware layer). The Arduino IoT Cloud dashboard's advanced graph feature is utilized to visualize and analyze the received data (application layer).

Objectives for Prototype 2 include:

1. Design, fabrication, and testing of a custom PCB to replace the breadboard used in Prototype 1.
2. Design and fabrication of a durable enclosure that can be securely mounted on the Griffith University footbridge without interrupting its usage.
3. Establishment of a reliable LoRaWAN connection between the node and the gateway, with consistent data transmission to the TNN cloud.
4. Integration of Arduino IoT Cloud variables for data logging and visualization using the dashboard's advanced graphs feature.
5. Execution of field testing on the bridge to validate complete system integration, including software functionality and signal strength.

Overall, the project aims to advance the understanding and application of IoT in structural health monitoring systems, potentially contributing to improved maintenance and safety practices.

2. REVIEW OF PUBLISHED LITERATURE

2.1. Review of the Published Literature

This literature review explores a variety of studies revolving around LoRaWAN and focuses on the fundamental end device setup, signal strength measurement, IoT cloud architecture and inherent limitations of the technology. LoRaWAN has gained traction for its promise of long-range, low-power solutions to WSNs which is critical for IoT applications. However, there are significant research gaps in the literature regarding the reliability and concistency of LoRaWAN in various environmental conditions and its practicality for applications requiring real time response. This review aims to evaluate these research gaps and evidence the design and optimization considerations for the LoRaWAN-based Structural Health Monitoring (SHM) deployment for the Griffith footbridge. The literature synthesized in this review is instrumental in shaping the approach and methodology of this research project.

One application of LoRaWAN is in the field of Agro-Informatics, as demonstrated by Gehani et al. (2021) [2]. Their research focused on utilizing LoRaWAN IoT architecture for the detection and classification of plant pathogens. The signal strength of the transmitted LoRa packets was evaluated through Received Signal Strength Indicator (RSSI) and Signal-to-Noise Ratio (SNR) at various depths from 0 cm to 60 cm. Their findings suggest that LoRa transmitters can function effectively provided the depth does not exceed 50 cm. A diagram of the experimental setup is shown in figure 2.1. This research serves as a valuable foundation for designing the enclosure thickness that will be used in this project.

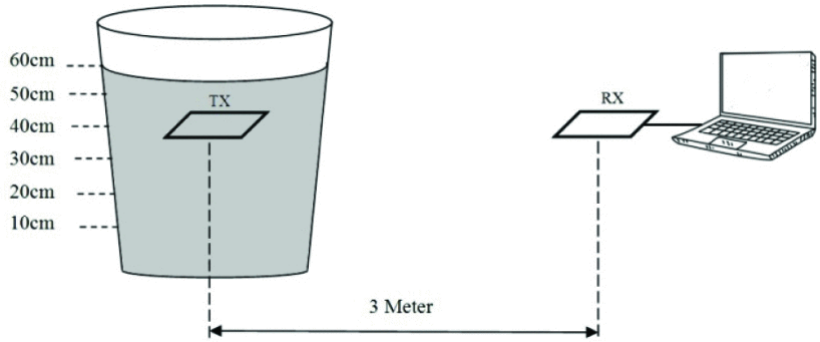


Fig. 2.1. Experimental Setup for LoRa Node in Soil [2]

Fox et al. (2019) [3] expanded on the architecture of LoRaWAN-based IoT systems with a focus on wind turbine monitoring in Ireland. Their study comprised three main components: an end device, a gateway, and an IBM IoT cloud platform. This research demonstrated a comprehensive IoT cloud architecture deployment, covering real-time monitoring, concurrent read/write processes, and database service implementation. De-

spite the successful deployment, challenges such as LoRa's limited transmission capacity and application specific end device battery life were highlighted, emphasizing the need for future optimization. In terms of practical application, end devices must be optimized to strike a balance between computational load and current draw to achieve sufficient longevity. Figure 2.2 displays an example of a GUI displaying real time data visualization from the Red Node Cloud Foundry application flow within the IBM cloud. The LoRa payload is extracted, processed and displayed in an interface by assigning relative values to their respective interface nodes. With the knowledge gained from this research, the variables from the LoRa payload in this project will be extracted using Arduino IoT cloud variables, and an Arduino IoT cloud dashboard will be used to plot relevant time series data. The Arduino LoRaWAN Adaptive Data Rate (ADR) mechanism will be used in this project to optimise the airtime, data rate and power consumption of the LoRaWAN deployment. This will assist in striking the balance between computational load and current draw of the LoRa node.

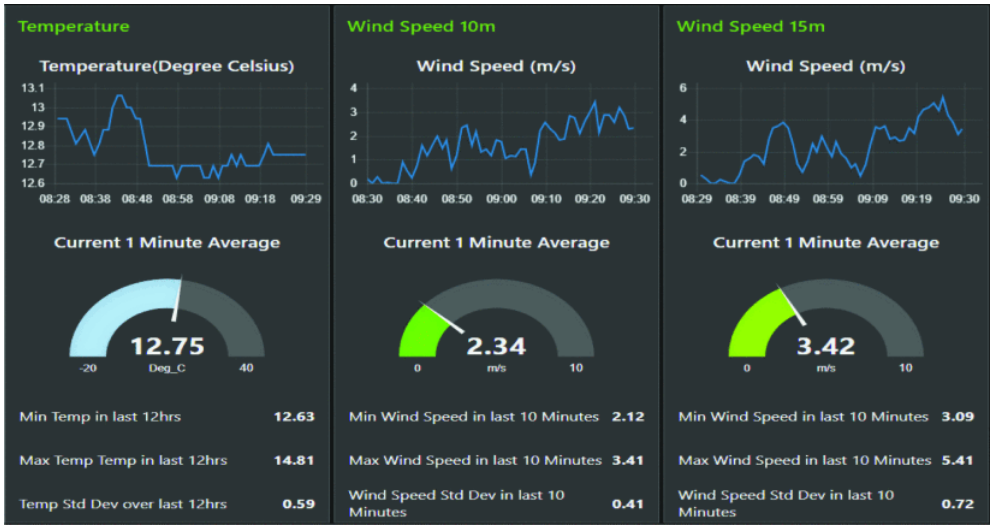


Fig. 2.2. Example of real time data visualization GUI [3]

In another exploration of LoRaWAN capabilities, Wildan et al. (2020) [16] designed and implemented a GUI for smart home systems leveraging LoRa's long-range and low-power benefits. Information such as room temperature is transmitted to a LoRa server through a LoRa client, and is uploaded to a web server. The data is then stored in a database and displayed on a webpage which is accessible from various devices such as a smartphone and laptop. An interface application was created to allow user interaction with the webpage and control over various electronic devices such as lights, TV and air conditioning within the house. The system's performance was tested with a focus on the user experience, web application and delay. This testing was able to address a key limitation in LoRaWAN, the delay in data transmission, noting that while the system performance was generally robust, delays averaged 3.86 seconds. This makes LoRaWAN generally limited for systems that require close to real-time responses. In relation to the smart home, operating the lights and TV with a 3.86 second delay is both significant

and noticeable. In its current state this system is functional, but will inevitably invoke hesitancy for mainstream adoption due to an installation cost for a user experience that is not seamless. The delay time evidenced in this study has resulted in avoiding unnecessary downlink messages for this project. This project will only focus on receiving uplink messages with only necessary downlink acknowledgements to avoid the restriction of a delayed response time.

Maziero et al. (2019) [4] deployed a real-time monitoring system for tracking electrical quantities at the Federal University of Santa Maria campus. Utilizing Grafana software, they created a panel for tracking both electrical and connectivity metrics, demonstrating the effectiveness of combining LoRaWAN technology and application layer monitoring software for real-time data acquisition and visualization. The study highlights the potential scalability of this system in the sense of feeding data stored in the monitoring center to algorithms capable of autonomously handling the distribution network of electrical metrics, for example automatically altering voltage based on state estimations. It was evident in the research that the ability to monitor electrical quantities in the application layer allowed for the capability of better management and informed decision making based on a holistic interpretation of the distribution network. Figure 2.3 displays the design for the electrical monitoring system and the process of receiving data from The Things Network (TNN) applications and transmitting payloads to a web-based user interface via MQTT and HTTP messaging protocols. The system design of this study gives an excellent context for the chosen implementation of TNN for this project. In this project TNN applications will instead communicate with the Arduino IoT cloud using cloud variables which will be used to plot data on the web-based Arduino cloud dashboard.

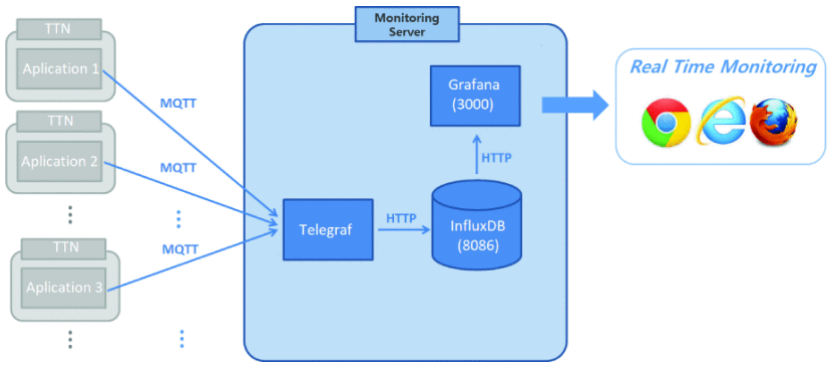


Fig. 2.3. IoT architecture for monitoring electrical quantities [4]

In a study examining the performance of a Smart Gateway network architecture, the authors conducted a series of test focusing on throughput and packet loss [17]. The throughput was found to be inconsistent across a different number of clients where the system was found to have a lower average throughput compared to other LoRaWAN systems due to additional required processing steps. The extra processing steps were attributed to Quality of Service (QoS) parameters, such as measuring packet loss and throughput, processing client data and web server services running on the gateway. The study found a significant packet loss especially when operating with two clients. On average, the sys-

tem exhibited packet loss of 26% across a 1-meter range which is a significantly high drop compared to other LoRaWAN implementations. Despite the range limitations, the study concluded that the gateway could support up to 5 clients registering and requesting data automatically, and run LoRa communication alongside the user interface of the information system simultaneously. In light of these findings, this report intends to use the ADR mechanism to automatically optimize QoS since it dynamically chooses the best spreading factor (SF) to send LoRa packets over. Additionally, the gateway used in this project will act as a base station and use the Arduino IoT Cloud and The Things Network (TNN) to handle the network and application layer, removing unnecessary gateway computations.

Wixted et al. (2016) [18] conducted reliability testing and discovered a successful connection and acknowledgement request between an end device and gateway across all spreading factors was found to be only 42% at a distance of 1.9km. After introducing a second gateway this connection rate rose to 70% and with the introduction of Internet Control Message protocol (ICMP) pings the full connection rate rose to 95.5%. It was also found that connection was completely lost in isolated stairwells indicating that LoRaWAN is sensitive to obstructions. Thanks to the findings of this research, the deployment of the LoRaWAN system in this project will focus on a short-range setup with minimal obstruction, and focus on the holistic deployment of the IoT architecture.

In a paper presenting an online test bed for LoRa development, the correlation between packet RSSI and distance was measured [5]. 5 LoRa nodes were placed at distances 500m, 1000m and 1200m away from the gateway, where the nodes placed 100m and 1200m were blocked by urban and natural obstacles. It was found that a low spreading factor (SF) results in a reduction of the packet reception rate (PRR). SF7 was the lowest spreading factor used and was confirmed to only support packet transmission at very low distance. The study concluded that RSSI decreases with distance and that this relationship is directly influenced by the environment, obstacles and spreading factor used for transmission. Various models can be used to model the relationship between RSSI and distance, but as evidenced in figure 2.4 the Bor model was closest in predicting correlation. To investigate the RSSI and SNR in this project, the values will be extracted from TNN and plotted over time series. This will further assist in gaining an understanding of the fluctuation of signal strength at a fixed distance from the Griffith footbridge.

Jang et al. in 2010 [19] reports the deployment of 70 wireless smart sensor networks (WSSNs) and two base stations on the Jindo Bridge in South Korea. Each sensor facilitates 3-axis acceleration measurement and offers a test bed for structural health monitoring (SHM). Cho et al. in 2010 [6] analysed data collected from these WSSNs, comparing these results to existing acceleration data from a wired monitoring system. The acceleration data collected from the deck and pylons of the bridge revealed a significant acceleration due to vehicle traffic. Vertical and cable vibrations were found to be significant enough for mode extraction and the estimated tension forces for 10 cables were found to lie within a 4% difference of the previous site inspections using wired monitoring systems.

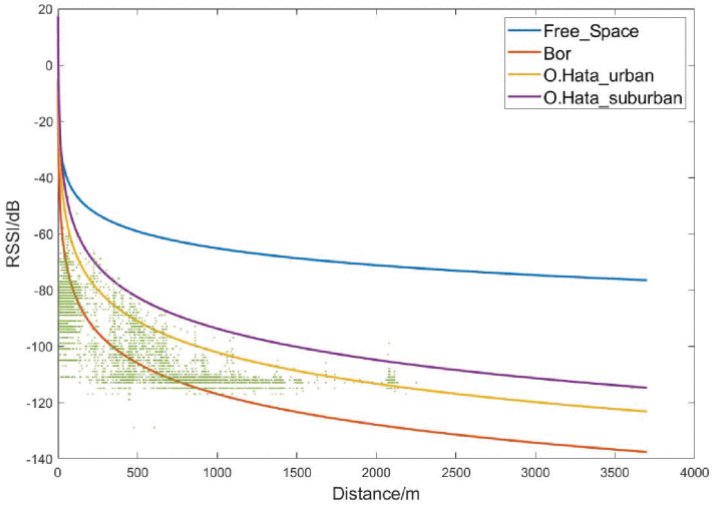


Fig. 2.4. Modelling RSSI-distance correlation. Green represents real data. [5]

The frequencies of the higher modes were found to exceed those predicted in the finite element (FE) model by less than 16%, indicating the benefit of WSSN data collection for verifying and amending simulated models. The PSD of the 2007 site inspection using the wired monitoring system is shown in figure 2.5. This gives an excellent insight for the expected PSD frequency peaks of the Griffith footbridge, indicating that a maximum peak of approximately 2Hz may appear.

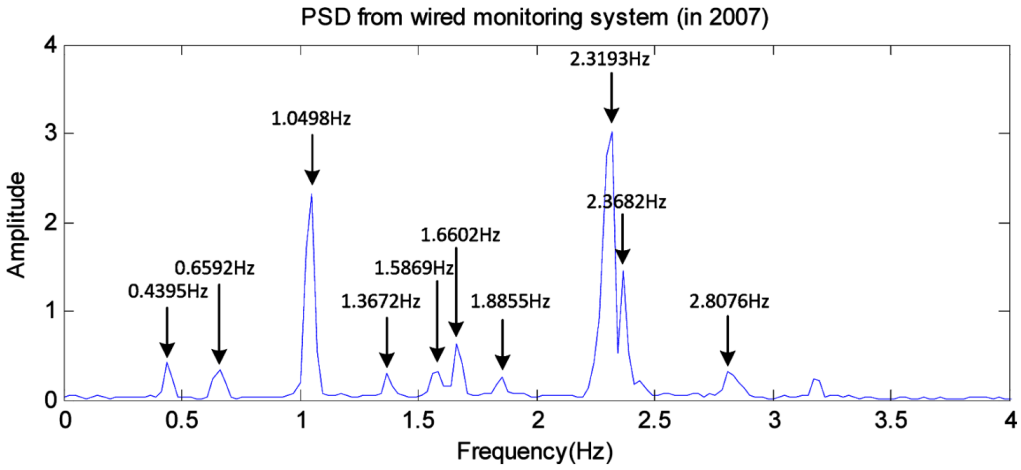


Fig. 2.5. PSD of vertical acceleration with wired monitoring system [6]

The existing literature on LoRaWAN exemplifies the versatility and potential of this technology in various IoT applications, including those demanding long range and low power consumption. Several limitations including the inconsistencies in packet loss, throughput and connectivity, coupled with significant data transmission delay, as outlined in table 2.1, present substantial challenges, especially for real-time data analysis and applications requiring consistent downlink messages. Such findings underscore the need for an in-depth understanding of LoRaWAN's performance to address these limitations in

this project's system design.

Reference	Significant Contribution	Limitation
Gehani et al. (2021)	LoRa Transmitters can function effectively up to a depth of 50 cm in soil.	Only 3 m distance between transmitting and receiving node.
Fox et al. (2019)	Comprehensive IoT cloud architecture deployment.	Limited transmission capacity and end device battery life.
Wildan et al. (2020)	Designed and implemented a GUI for smart home systems leveraging LoRa.	Data transmission delay averaged 3.86 seconds.
Maziero et al. (2019)	Real-time monitoring system for tracking electrical quantities.	No explanation for smart meters with packet delivery less than 95%.
Eridani et al. (2019)	Conducted a series of tests focusing on throughput and packet loss.	Inconsistent throughput and significant packet loss.
Wixted et al. (2016)	Successful connection and acknowledgement request between an end device and gateway.	Connection was completely lost in isolated stairwells and low with only one gateway.
Wang et al. (2019)	Correlation between packet RSSI and distance was measured.	RSSI decreases with distance and is directly influenced by environment, obstacles and spreading factor.
Jang et al. (2010)	Large scale deployment of WSSNs and two base stations on the Jindo Bridge.	Linear relationship between number of requested data points and communication time.
Cho et al. (2020)	Analysed data collected from the WSSNs.	Independent WSSN analysis required due to no synchronization.

TABLE 2.1. SUMMARY OF REVIEWED LITERATURE

In the context of SHM, LoRaWAN presents a compelling alternative to traditional WSNs, such as Wi-Fi, Bluetooth and Zigbee, offering superior range and adaptability whilst satisfying low power requirements and the potential for flexible sensor placement. However, the highlighted limitations from the literature must be carefully addressed in the design of this LoRaWAN-based SHM system. This project will seek to tackle the presented challenges by focusing on the design optimization for the Griffith footbridge monitoring system. In particular, the aim is to strike a balance between computational load, power consumption and data transmission delay whilst ensuring robust and reliable system performance.

3. DESIGN PROCESS

3.1. Methodology

The methodology to achieve the objectives and outcomes for this project were divided into the 18 steps listed in table 3.1.

3.2. Design Specifications

Prototypes 1 and 2 were designed with the following specifications.

3.2.1. System Design

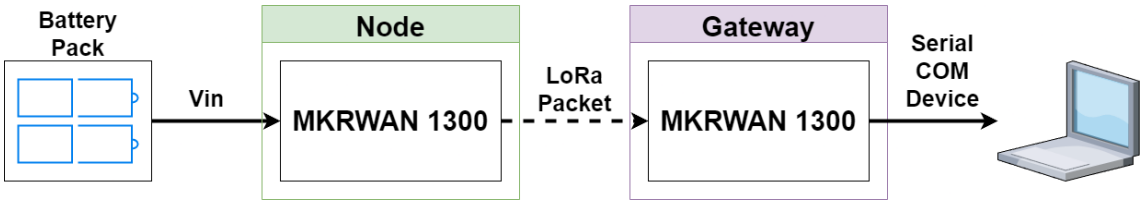


Fig. 3.1. Prototype 1 System Diagram

The system diagram in figure 3.1 was used for initial testing in the beam experiment and has two variations. In the first variation the node is connected via COM device to a laptop and establishes serial connection. In the second variation there is no physical connection between the node and laptop and the node is powered via battery. The initial design with serial connection is so that the raw acceleration can be recorded since this is too much data to send over a LoRa packet. In both variations the node sends the maximum acceleration and peak frequency via LoRa packet to the gateway node.

The system diagram in figure 3.2 forms the system design of prototype 2 and was used in the integrated testing and data collection on the Griffith footbridge. Since the gateway was unable to connect to the Griffith Wi-Fi due to security infrastructure, an Apple MacBook Pro was turned into a router using the inbuilt internet sharing option. A mobile phone hotspot was used to connect the laptop to the internet, and this was shared with the gateway via an Ethernet port. The gateway was configured in Basics Station mode, with TLS certificate and token authentication, and was connected to the Arduino IoT Cloud via the TNN Stack V3 back end integration via the LoRaWAN Network Server (LNS) sub protocol. The LoRa packets and message payloads are managed by TNN and are stored in Arduino Cloud variables. The Arduino Cloud Maker subscription offers 3 months of data retention which was used to plot the acceleration and frequency variables

on two advanced graphs in the Arduino Cloud dashboard. This dashboard can be accessed on the laptop via the Arduino Cloud dashboard web page, or on the mobile phone via the Arduino IoT Remote app. Due to the cloud variable data retention, the variable history log can be downloaded in CSV format and plotted in excel.

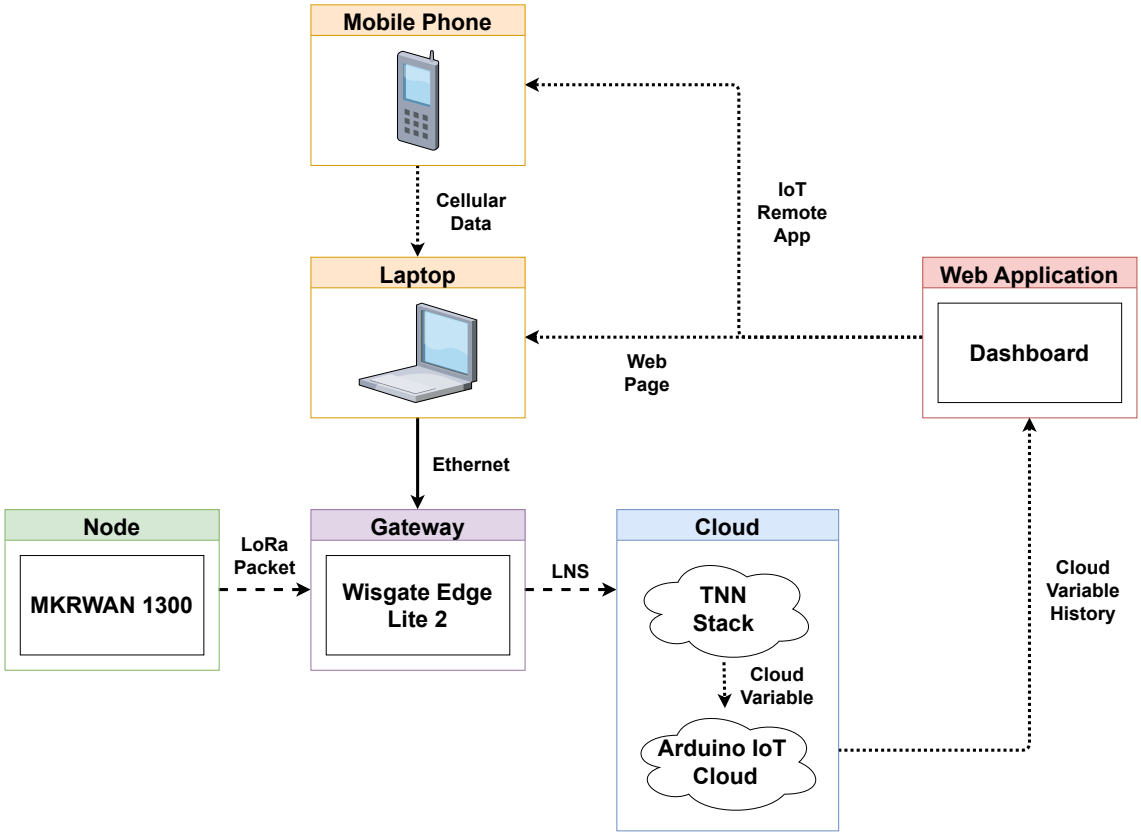


Fig. 3.2. Prototype 2 System Diagram

3.2.2. Data Processing

Sampling

For the beam testing the sampling number (N) was 128, and for the bridge testing N increased to 256. A cut-off frequency (F_c) of 25 Hz was chosen, and according to the Nyquist theorem, the sampling Frequency (Fs) was chosen to be 50 Hz which meets the minimum value of $2 * F_c$ to avoid aliasing. This cut-off and sampling frequency are well above the desired 2 Hz to 3 Hz values and allow for higher frequencies to be shown, allowing for adjustments to the software. The sampling interval is then $\frac{1}{F_s}$, and substituting Fs with 50 Hz gives 0.02 seconds between each sample. This gives a total of 2.56 seconds to sample 128 values and 5.12 seconds to sample 256 values.

Step	Process
1	Perform literature review on LoRaWAN technology. Determine suitable steps to implement the IoT architecture.
2	Perform literature review on structural health monitoring and existing WSN structural implementations.
3	Examine results from FE analysis for beam experiment and review the Griffith footbridge design paper to collect expected first mode flexural frequencies.
4	Determine the correct sampling parameters for the accelerometer and choose a suitable FFT library.
5	Implement the FFT library functions and test with a 2Hz sine wave.
6	Integrate the accelerometer data with the FFT functions and determine the maximum acceleration and maximum frequency. Implement a suitable low pass filter and noise thresholds for data collection.
7	Develop software for the beam test experiment. This includes Python code to log and plot serial data.
8	Conduct beam experiment and review the plotted data. Adjust filter and noise threshold values accordingly.
9	Conduct node to node range test.
10	Design PCB carrier board for microcontroller and accelerometer.
11	Design and print enclosure for PCB and battery pack.
12	Solder connections from microcontroller, accelerometer and battery pack to PCB.
13	Choose suitable LoRaWAN gateway and set up as basics station mode. Add TLS authentication certificate and token to connect to TNN.
14	Register things and cloud variables in Arduino IoT cloud. Add graphs in dashboard for cloud variables.
15	Adapt code to store variables of interest in cloud variables and sync with Arduino Cloud.
16	Install enclosure behind hand rail of Griffith footbridge. Setup gateway and laptop nearby.
17	Conduct bridge testing. Record the packet payload, SNR, RSSI and spreading factor.
18	Determine the best noise thresholds through iterative testing.

TABLE 3.1. METHODOLOGY TO ACHIEVE PROJECT OBJECTIVES

Discretising the Accelerometer Readings

The accelerometer values are input to the program via the Arduino analogRead() function. These raw values are represented as voltage changes ranging from 0-3V. In this scenario, 1.5 V represents the stationary state, 0V maximum acceleration in the negative direction and 3.0 V the maximum acceleration in the positive direction. The x, y and z axis are inputs to three of the MKRWAN1300's analog inputs, A1, A2 and A3. The microcontroller uses the on-board 12-bit analog to digital converter (ADC) to map these voltage readings to a corresponding discrete value between 0 and 4095. The acceleration is then calculated using the accelerometer's reference voltage which is 3.3 V when powered via USB and 3.0 V when powered via battery, and the accelerometer's sensitivity value which is 0.33 mV/g. The zero value is found during the calibration process of the accelerometer which will be outlined in the software design section. These values are multiplied by Earth's gravitational constant g (9.81 m/s/s) to convert the acceleration from G's to m/s/s. This process can be seen in equation 3.1. 1.0 m/s/s is subtracted from the z-axis for scenarios in which the z-axis is facing upwards to remove Earth's downwards directional gravity from this reading. A noise threshold is then applied to the acceleration values depending on the implementation to remove electrical noise from the signal. Equation 3.1 is an example of generating processed acceleration values.

$$a[i] = \frac{\frac{a\_raw[i] * vRef}{adc\_resolution - 1} - \frac{vRef}{2}}{sensitivity - aZero} * g \tag{3.1}$$

Filtering

A low-pass filter was applied to the processed acceleration values to remove high frequencies. The cut-off value for this filter was 5 Hz (LP_F_c) and a smoothing factor of 0.283 was picked after experimental testing. The smoothing factor effectively acts as a weighting for the filter where higher values allow higher frequencies to pass through and lower values filter out high-frequency components. This filter effectively removes noise that was missed by the noise thresholds in the processed acceleration and ensures that no high frequencies due to small and rapid acceleration pass through. The smoothing factor for the low-pass filter was calculated with equation 3.2.

$$Smoothing\_Factor = \frac{1}{1 + (2 * \pi * LP\_F_c * (\frac{1}{F_s}))} \tag{3.2}$$

Fast Fourier Transform

To facilitate the desired maximum frequency response of acceleration over time for both prototypes, the Arduino Fast Fourier Transform (FFT) library was used to convert the

processed acceleration values into the respective frequency response. This frequency response was compared to the Strand7 Finite Elements (FE) simulation of the beam experiment and the Griffith footbridge documentation to validate the first mode flexural frequency (natural vertical frequency) for each of the prototypes. The predicted response of the FFT from the beam experiment is a steady peak of approximately 2.4 Hz, and the documented natural frequency of the bridge was 2.2 Hz.

When performing the FFT, N must be chosen as a power of two as defined in the library code. This is because the FFT is the most efficient when the sample window is to the power of two and so this condition is hard-coded into the library function, hence the values of N, 128 and 256 were chosen. The Hanning window was chosen for the FFT since it offers an adequate balance of frequency resolution and suitable distinction of frequency peaks. This window also minimizes spectral leakage which is helpful to distinguish between the frequency peaks due to acceleration and frequency peaks due to noise in the accelerometer. The FFT function assumed no imaginary values and used a complex to magnitude function to convert from polar to rectangular notation.

3.2.3. Design Resources

The prototypes developed in this project required numerous programming tools, languages, electronics and materials. The microcontroller code was written in the C++ language in the Arduino IDE and the Python language was used for the logging and plotting scripts for prototype one. Altium Designer was used to design the PCB, and OnShape was used for the 3D modelling of the enclosure for prototype 2. The bill of materials for each prototype are displayed in table 6.1 and table 6.2. This bill of materials does not include shipping costs.

3.2.4. Software Design

The software design for this project is outlined in the software diagrams in figures 3.3, 3.4, 3.5, 6.1, 6.2 and 3.6. The software processing steps for each prototype are outlined in tables 3.2 and 3.3.

Import Libraries and Modules

The external libraries and modules specified in steps 1, 10, 14 and 20 of table 3.2 and step 1 of table 3.3 are listed in table 3.4.

Program Parameters

The program parameters specified in steps 1, 10, 14 and 20 of table 3.2 and step 1 of table 3.3 are listed in table 3.5.

Initializing LoRa and Serial Connection

In the prototype 1 setup function for the sender and receiver programs, the serial connection is established with 9600 baud using the Serial.begin() function. This allows the microcontroller to print data serially to the Arduino IDE serial output. This also allows data to be logged via the serial connection within the logging and plotting Python scripts. Since the prototype one beam test involves two LoRa nodes communicating via LoRa packets, the LoRa.begin() function is used to initialized LoRa connection on AU915 using 915E6, representing 915 MHz frequency.

Accelerometer Calibration

In both prototypes, a calibration function is used to initialize the rest values of the accelerometer. This occurs in the setup function and is especially important for testing since the accelerometer must be calibrated with zero values for each axis when the position is changed (table 3.5). Since the calibration function is called within the setup function, the device is calibrated when the reset button is pushed on the microcontroller. The calibration function works by first sampling 256 raw acceleration values, processing these values using the ADC's resolution, voltage reference and the accelerometer's sensitivity, and adding these processed acceleration values to a cumulative offset value for each axis. These offset values are then divided by 256 effectively calculating the average stationary offset and setting these values as the zero point for each axis. When the acceleration is processed in the main function of the sender program, these zero point values are subtracted from acceleration value. In practice, this means that the microcontroller and accelerometer can be placed in any initial starting orientation and the calibration process will correct the data values accordingly.

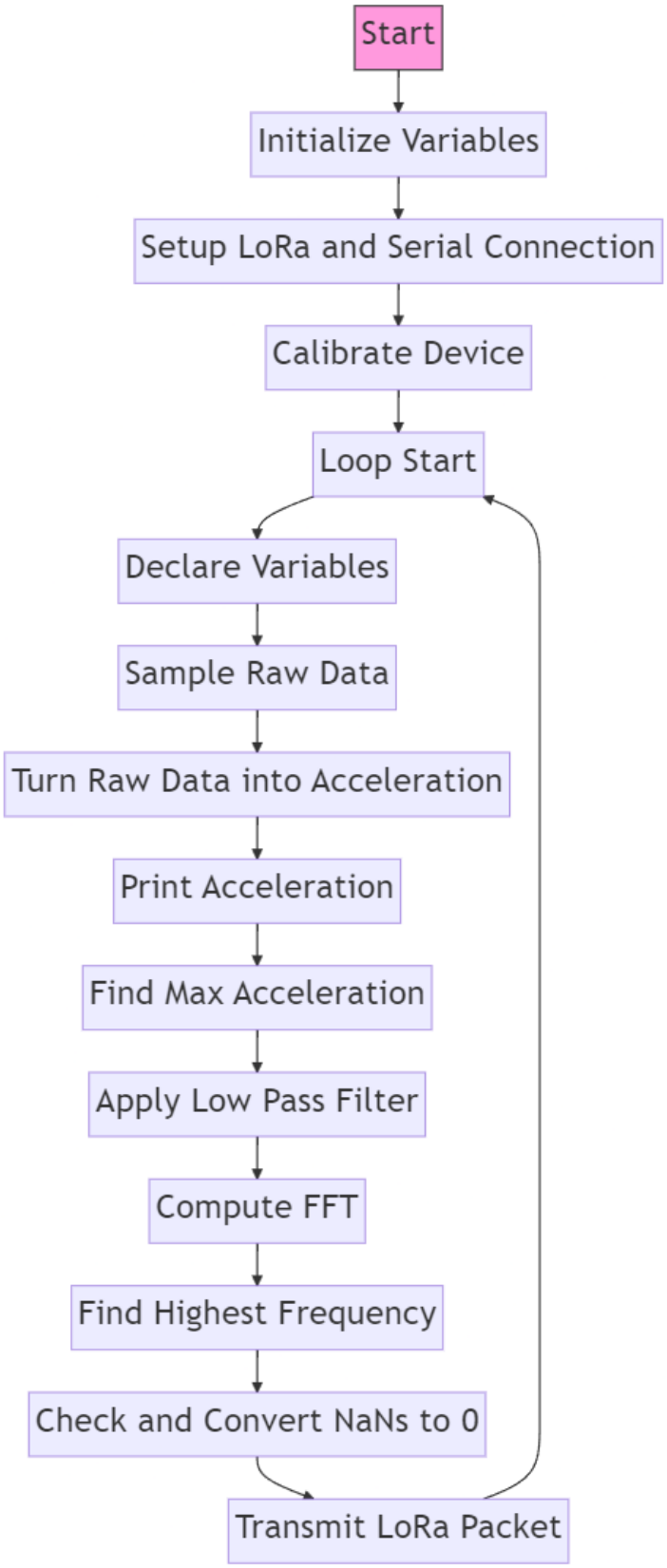


Fig. 3.3. Prototype 1 Software Diagram: Sender

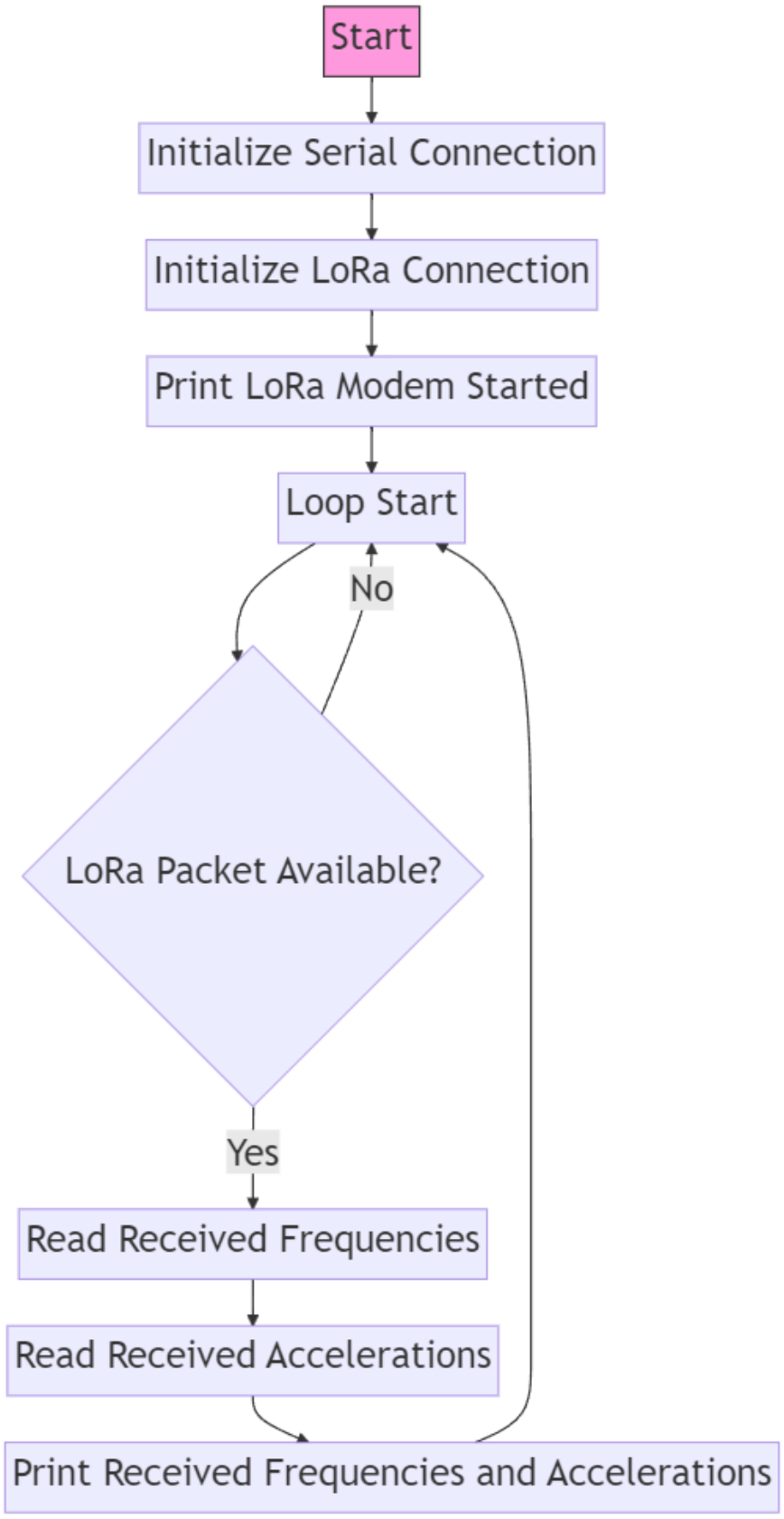


Fig. 3.4. Prototype 1 Software Diagram: Receiver

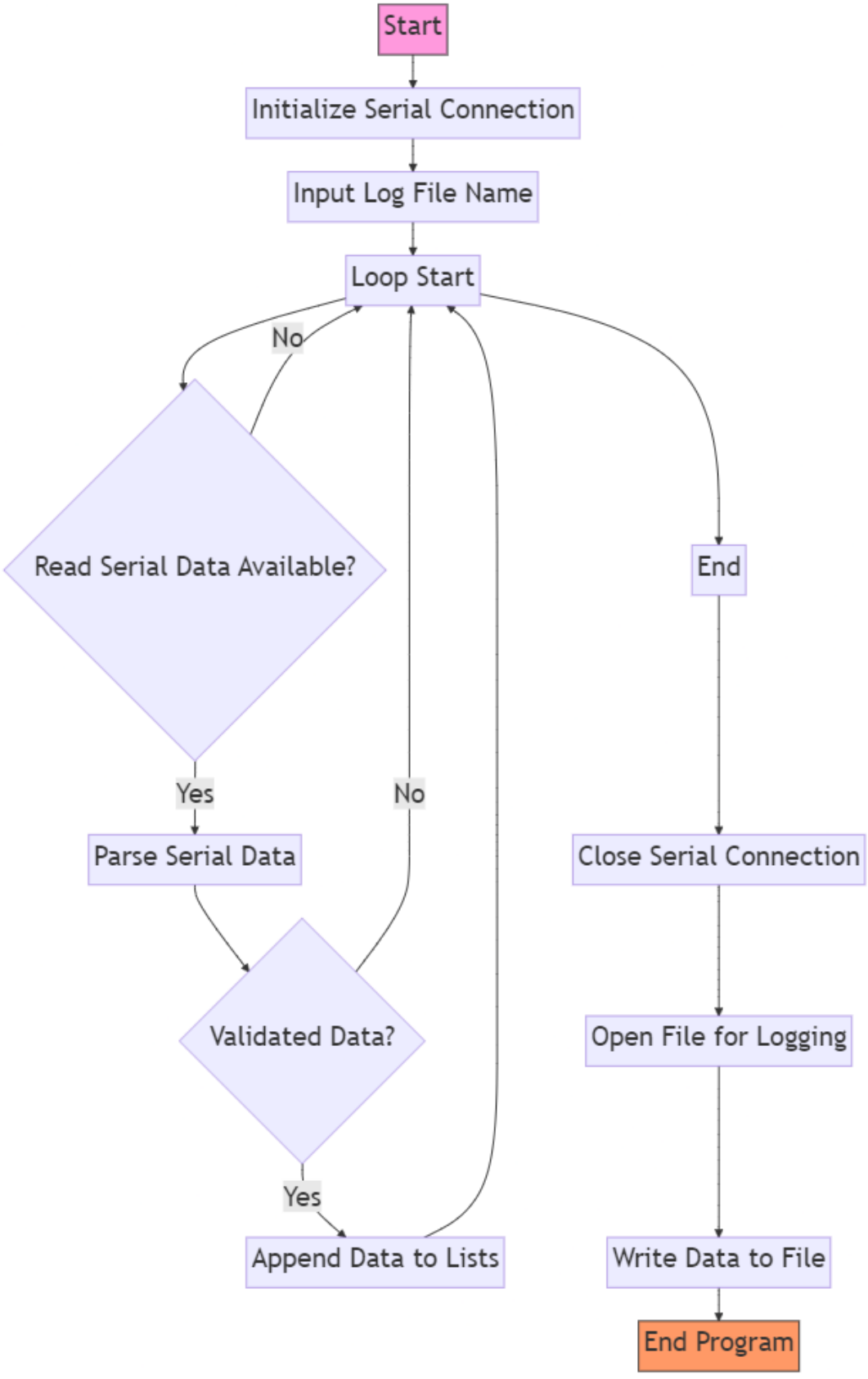


Fig. 3.5. Prototype 1 Software Diagram: Sender & Receiver Serial Logging

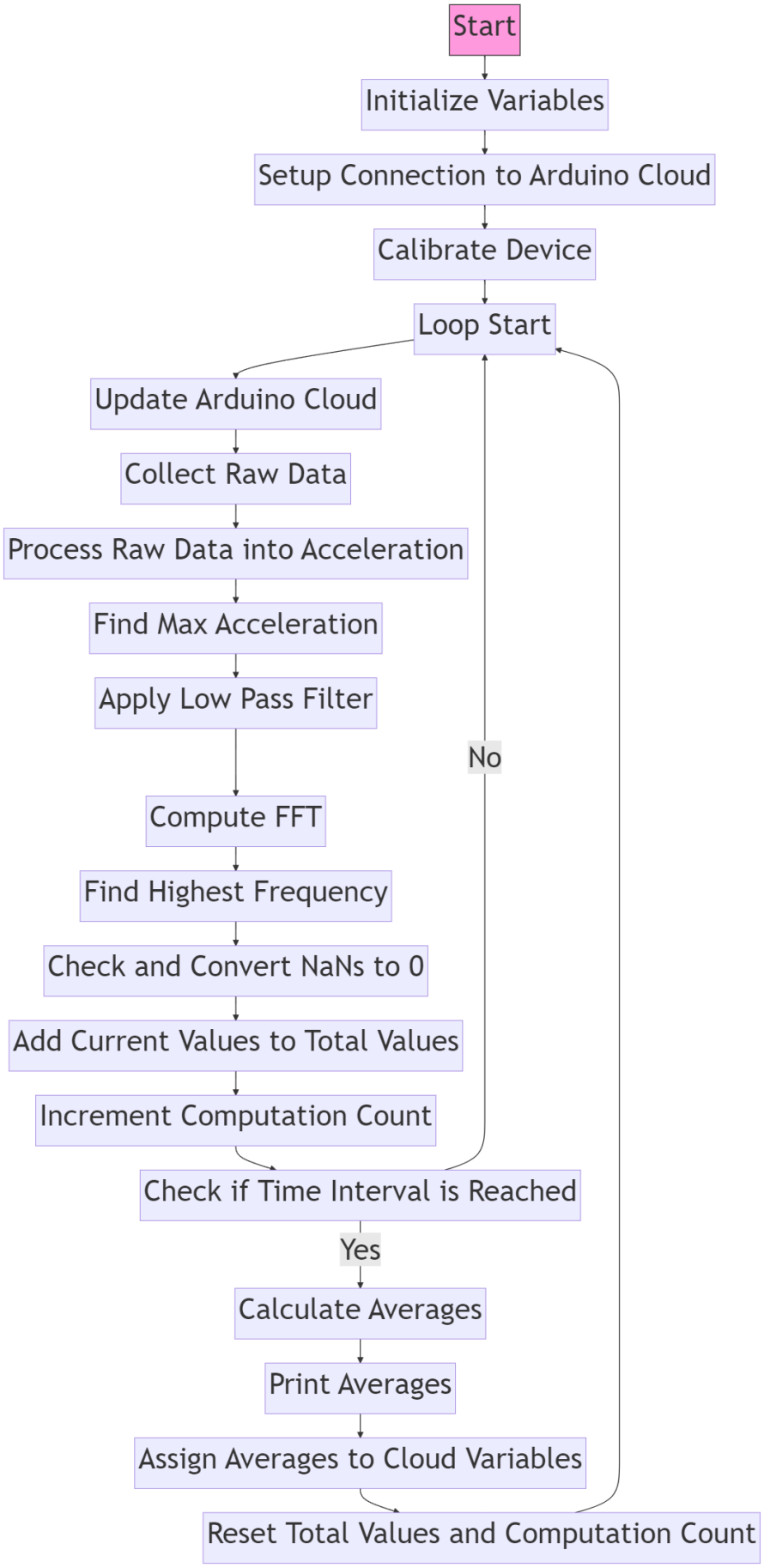


Fig. 3.6. Prototype 2 Software Diagram

Step	Process
Sender	
1	Relevant Arduino and CPP libraries are imported and variables are initialized.
2	LoRa and serial connection is initialized.
3	The accelerometer is calibrated and zero values are set for each axis.
4	The raw acceleration is sampled and processed.
5	The maximum acceleration is found.
6	Low pass filter is applied to the acceleration values.
7	FFT is computed and the peak frequency is found.
8	NaN values are converted to 0.
9	Max acceleration and frequency values for each axis are converted to bytes and sent via LoRa packet.
Receiver	
10	Relevant Arduino and CPP libraries are imported and varaibles are initialized.
11	LoRa and serial connection is initialized.
12	Check whether there is a LoRa packet to read. The code loops until there is an available packet.
13	When there is an available packet the frequency and acceleration values are decoded and printed.
Logging	
14	Relevant Python modules are imported and serial connection is initialized.
15	The user enters the name of the log file.
16	Check whether there is serial data to read. The code loops until there is available data.
17	When there is data available it is parsed and validated.
18	Once validated the data is appended to a list and the serial connection is terminated.
19	The serial data is written to the specified file.
Plotting	
20	Relevant Python modules are imported.
21	The user enters the name of the log file and the plot file.
22	The data from the log file is read and plotted.
23	The plots are saved with the specified file names and displayed to the user.

TABLE 3.2. PROTOTYPE 1: BEAM TEST SOFTWARE PROCESSING STEPS

Step	Process
Sender	
1	Relevant Arduino and CPP libraries are imported and variables are initialized.
2	Connection to Arduino Cloud is established.
3	The accelerometer is calibrated and zero values are set for each axis.
4	The main loop starts and the Arduino Cloud is updated.
5	The raw acceleration is sampled and processed.
6	The maximum acceleration is found.
7	Low pass filter is applied to the acceleration values.
8	FFT is computed and the peak frequency is found.
9	NaN values are converted to 0.
10	Current max acceleration and frequency values are added to a running total.
11	The loop iteration value is incremented.
12	The time passed is checked. If less than timer value the loop resets.
13	If the time passed is equal to or greater than the timer value then the averages of the running total values are found using the loop iteration value.
14	The average values are printed and assigned to their respective cloud variables.
15	Total values and loop iteration values are reset and the loop is restarted.

TABLE 3.3. PROTOTYPE 2: BRIDGE TEST SOFTWARE PROCESSING STEPS

Prototype 1 Main Software Loop

The main software loop for prototype one is displayed in the software diagrams in figure 3.3, figure 3.4, figure 3.5, figure6.1 and figure 6.2.

The sender program is uploaded to the node microcontroller and the receiver program is uploaded to the gateway microcontroller as specified in the system diagram in figure 3.1. The reset button on each device is pushed which causes the node microcontroller to calibrate, and the serial and LoRa connections to be established on both devices. The main program on the sender device then starts reading in raw acceleration values from the accelerometer and converts these into m/s/s values. These processed acceleration values are serially printed. The maximum acceleration value for each axis is found and the low-

Library/Module	Prototype	Program	Function
Arduino	1, 2	Sender	Provides core Arduino functionality such as reading analog input and serial printing.
Math.h	1, 2	Sender	Standard C library for mathematical operations such as finding absolute max values and checking for NaN values.
MKRWAN.h	1	Sender, Receiver	Standard MKRWAN library for basic LoRa communication and channel masking.
LoRa.h	1	Sender, Receiver	Used to control LoRa communication between devices such as writing to a LoRa packet.
ArduinoFFT.h	1, 2	Sender, Receiver	Used to provide FFT functions such as FFT computing, windowing, DC removal and complex to magnitude conversion.
serial	1	Logging	Used for establishing serial connection to COM devices.
matplotlib.pyplot	1	Plotting	Library used for plotting serial log data.
os	1	Logging, Plotting	Used for attaching file paths to user specified files.

TABLE 3.4. ARDUINO, CPP & PYTHON LIBRARIES AND MODULES

pass filter is applied to the processed acceleration values. FFT objects for each axis are created using the processed acceleration values, number of samples and sampling rate. DC bias is removed by subtracting the mean acceleration from each value and a Hanning window is then applied to the data. The FFT is then computed and the result is converted from complex to magnitude form. The Arduino FFT library's MajorPeak() function is called which returns the peak frequency from the computed FFT. Now, the checkNan() function is called for the maximum acceleration and frequency values to convert any NaN values to zero. A LoRa packet is then created using the LoRa beginPacket() function and the maximum acceleration and frequency values are converted to byte format and written to the packet. The packet is then closed using the LoRa endPacket() function and transmitted to the gateway node.

The main program on the receiver device checks if a LoRa packet is available to read using the LoRa parsePacket() function. Once there is an available packet the maximum

Parameter	Prototype	Program	Description	Type
x_axis, y_axis, z_axis	1, 2	Sender	Assign axis values from accelerometer to analog inputs on microcontroller.	static const int
adc_resolution	1, 2	Sender	Resolution of the microcontroller's 12-bit ADC.	static const int
vRef	1, 2	Sender	Input voltage for ADC.	const double
sensitivity	1, 2	Sender	Sensitivity of the accelerometer.	static const double
maxFreq	1, 2	Sender	Cut-off frequency for acceleration sampling.	static const int
sample_n	1, 2	Sender	Set the number of acceleration samples.	static const int
sampling_rate	1, 2	Sender	Sampling frequency for acceleration sampling.	static constexpr double
sample_interval	1, 2	Sender	Delay for acceleration sampling in milliseconds.	static constexpr double
LP_Fc	1, 2	Sender	Low pass cut-off frequency.	constexpr double
LP_alpha	1, 2	Sender	Low pass filter smoothing factor.	constexpr double
window_type	1, 2	Sender	Specify window type for FFT.	define
FFT_dir	1, 2	Sender	Specify FFT direction.	define
xZero, yZero, zZero	1, 2	Sender	Zero values for acceleration calibration.	double
totalMaxAccelX, totalMaxAccelY, totalMaxAccelZ	2	Sender	Running total for finding average max acceleration values.	float
totalMaxFreqX, totalMaxFreqY, totalMaxFreqZ	2	Sender	Running total for finding average max frequency values.	float
computationCount	2	Sender	Counts the number of loop iterations for finding average max values.	int

TABLE 3.5. GLOBAL PROGRAM PARAMETERS

acceleration and frequency values are decoded from bytes to floats and serially printed.

The logging Python script for the sender and receiver devices are executed and write the serial output from each device to the user specified text file. The logging is ended when the user ends the programming by inputting the control c command.

The plotting Python script for both devices is then executed and reads the logged data from the user specified text file. The raw acceleration, maximum acceleration and maximum frequency are plotted using Matplotlib's pyplot function and the figures are stored as PNG images with the user specified file name.

Arduino Cloud Initialization

When cloud variables are created on the Arduino IoT Cloud platform, a thingProperties.h header file is automatically generated with the device's app EUI, app key and thing ID. A network mask is applied to the AU915 frequency band to ensure communication over frequency sub band two which is required for the TNN back-end integration.

Prototype 2 Main Software Loop

The main software loop for prototype two is displayed in the software diagram in figure 3.6.

The main code for the sender node in prototype two is similar to that in prototype one with some major key differences, starting with the initialization of the global variables defined in table 3.5. The calibration process is identical to prototype one but the setup function now initiates a connection to the Arduino Cloud platform using the device EUI and app key defined in the thingProperties.h header file. At the start of the main loop, the Arduino Cloud update() function is called which updates the initial values of the cloud variables to zero. The raw acceleration is processed in the same way as prototype one, and the maximum acceleration is found, FFT computed and maximum frequency peak is calculated. These maximum values are added to a respective running total value and the computation count is incremented which is used to find the average of these maximum running totals over a 60 second period. Once 60 seconds has passed the averaged maximum values are stored in the cloud variables provided by the thingProperties.h header file. The computation count is then reset to zero. The main loop then calls the update() function again and the updated cloud variables are stored on the Arduino IoT Cloud platform.

3.2.5. Hardware Design

PCB Design

The PCB for this project was designed in Altium Designer. The purpose of this PCB was to replace the breadboard in prototype one with a carrier board for prototype 2, facilitating the connection between the microcontroller, accelerometer and battery pack. A footprint library was initially created to match the dimensions and pin locations of the microcontroller and accelerometer. Each device has a pitch of 2.54mm. The accelerometer footprint can be seen in figure 3.7 and the microcontroller in figure 3.8. The pink mesh represents the mechanical keep-out layer which matches the physical dimensions of each device. The physical dimensions of the accelerometer were width 15mm and length 18mm. The physical dimensions of the microcontroller were width 25mm and length 67.6mm.

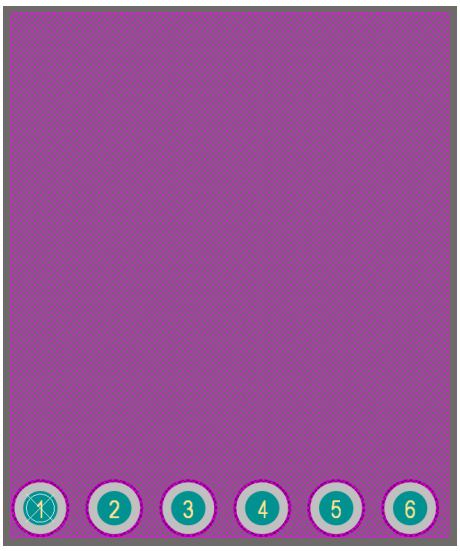


Fig. 3.7. Accelerometer PCB Footprint

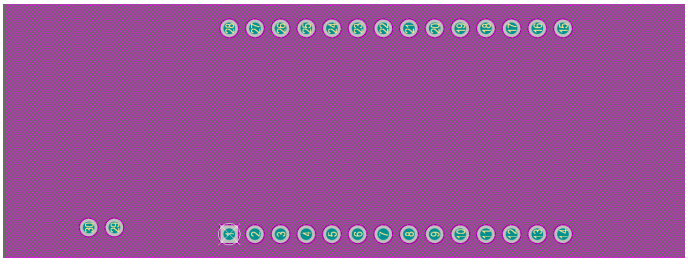


Fig. 3.8. MKRWAN1300 PCB Footprint

A schematic library was then created with symbols for each device. These symbols contain nets for each of the pins on both devices. The symbol for the accelerometer is shown in figure 3.9 and the microcontroller in figure 3.10.

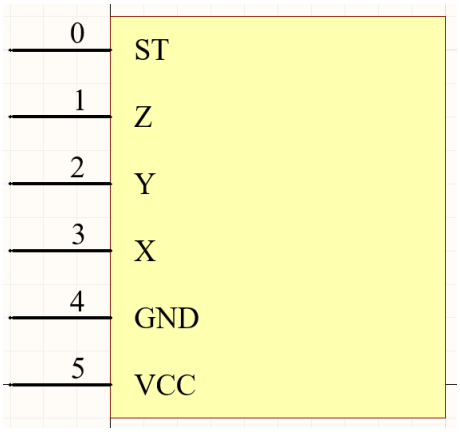


Fig. 3.9. Accelerometer SCH Symbol

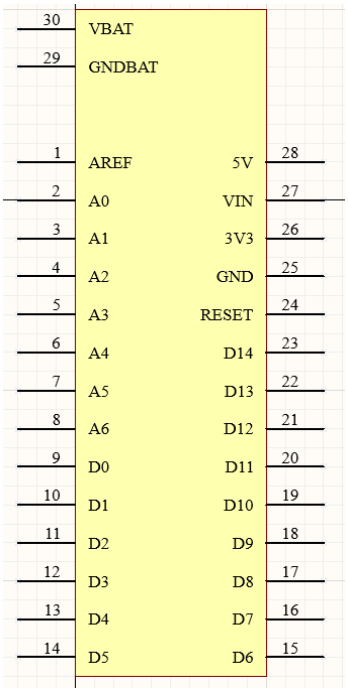


Fig. 3.10. MKRWAN1300 SCH Symbol

The symbols are used to create the schematic of the PCB, and the correct connections between the accelerometer and microcontroller pins are traced. The full schematic can be seen in figure 3.11. The schematic was then imported into a PCB document and the board was created with width 147.6mm, height 31mm and a thickness of 0.4mm. The trace width was set to 0.254mm and tear drops were added to each pin connection. Mounting holes with 2mm diameter were added to each corner of the PCB for enclosure mounting, and labels for the microcontroller and accelerometer were added to the top overlay. The 2D layout is shown in figure 3.12 and the 3D layout is shown in figure 3.13.

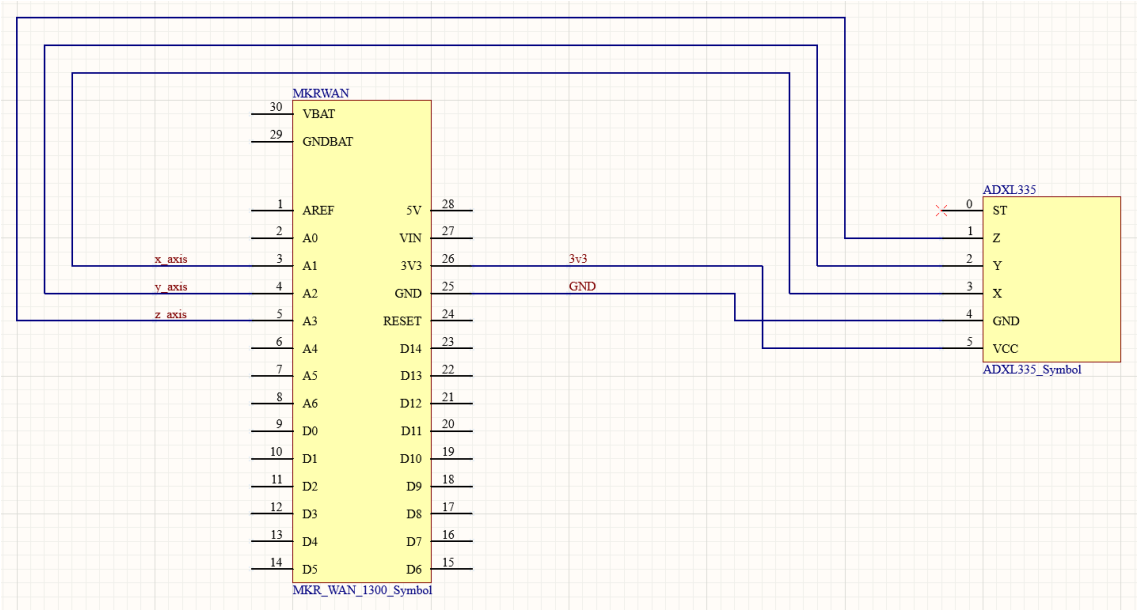


Fig. 3.11. PCB Schematic

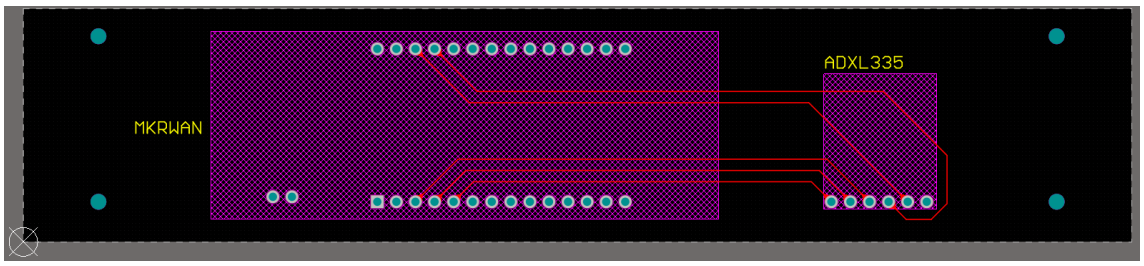


Fig. 3.12. PCB 2D Layout

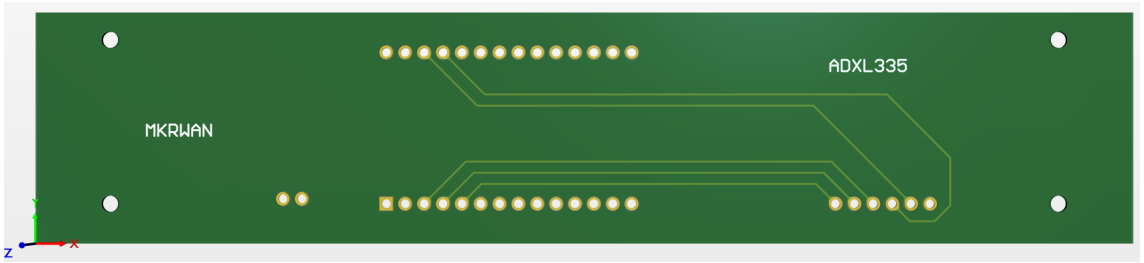


Fig. 3.13. PCB 3D Layout

Enclosure Design

The enclosure for this project was designed in OnShape CAD software. The enclosure was designed to sit perfectly behind the handrail of the Griffith footbridge. The enclosure was 3D printed at Griffith University using hard PETG filament. In order to fit the enclosure onto the 3D printer it had to be designed with a front and back plate which was secured together using six M3 heat set-ins and six M3 20mm machine screws. The lid for this enclosure was designed with an overlap and was secured to the top of the enclosure using four of the same M3 heat set-ins and four M3 20mm machine screws. In order to install the heat set ins, the enclosure was printed with holes of 4mm and the heat set-ins were melted into these holes using a soldering iron. Four M2 30mm screws were used to secure PCB mounts to the back plate of the enclosure, and the PCB was fastened to these mounts using four M2 nuts. The enclosure was printed with a pass-through hole for an M10 bolt that was used to secure the enclosure to the bridge and an M10 nut was used to fasten the bolt to the enclosure. A shelf was created on the right side of the enclosure for the battery pack to sit within. An exploded view of this enclosure can be seen in figure 3.14 and the bridge installation can be seen in figure 3.15. The inside of the enclosure is displayed in figure 3.16a and the installation on the Griffith footbridge can be seen in figure 3.16b.

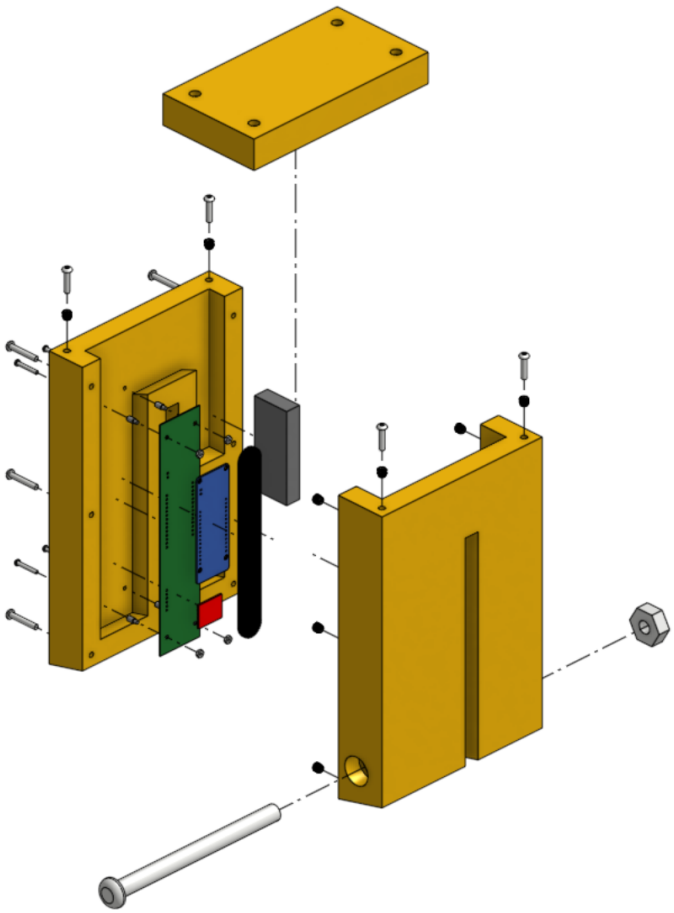


Fig. 3.14. Enclosure Explode View

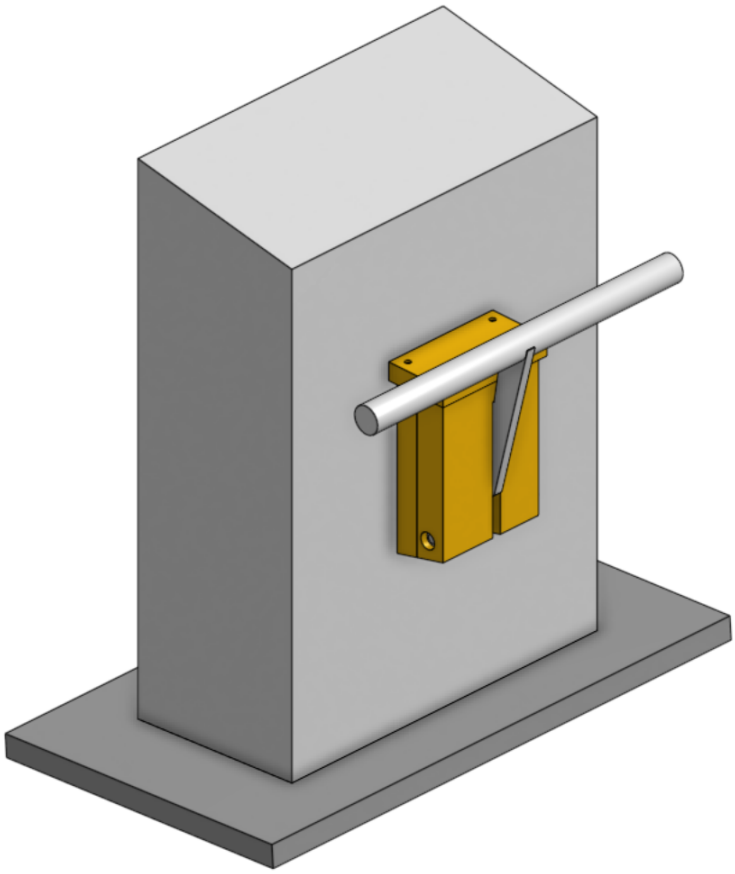
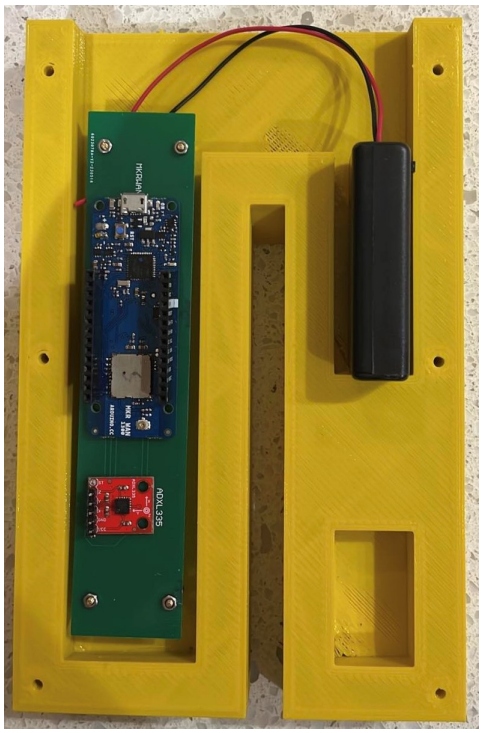
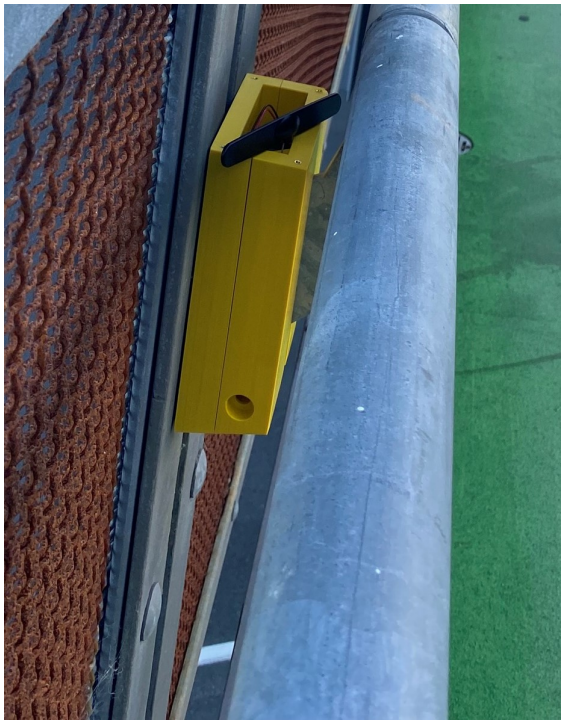


Fig. 3.15. Enclosure Bridge Installation



(a) Inside of Enclosure



(b) Enclosure Griffith Footbridge Installation

Fig. 3.16. Enclosure Integration

3.2.6. Packet Transmission

Data transmission calculations were conducted using the TNN's LoRaWAN airtime calculator [20]. According to Australian communication laws, LoRa packets must be transmitted on the AU915 frequency band between 915-928MHz. As part of this restriction, messages must adhere to a 1% duty cycle. Since the message payload contains six float values, the payload itself is 24 bytes in size. Adding the overhead of 12 bytes, which contains information such as the device address and frame counter, the total packet size is 36 bytes. The Arduino IoT Cloud platform uses Adaptative Data Rate (ADR) to automatically determine the most suitable spreading factor (SF) for transmission. For the case of this project, a spreading factor of 7 is suitable for a connection up to 2km. The SF7BW125 data rate needs a minimum of 77.06ms airtime to transmit a message size of 36 bytes. Accounting for the 1% duty cycle this gives a minimum requirement of 7.71 seconds between subsequent messages on the same sub band allowing for a total of 467 messages to be sent per hour. Adhering to the restrictions of the TNN fair access policy, the data rate is restricted to 30 seconds of air time per device per day, which imposes a limit of 389 messages per day with a 36 byte packet size. Finally, if the device is transmitting all day, a maximum of 16.2 message per hour can be sent.

4. PROTOTYPE TESTING

4.1. Testing Methodology and Criteria

4.1.1. Prototype 1: Beam Test

The purpose of the beam test was to test the functionality of the software on prototype one, and to validate the first mode fundamental frequency from the Strand7 FE simulation. This test involved fastening a large metal beam to a clamp, and securing the node device to the end of the beam using elastic bands. As outlined in the system diagram in figure 3.1, the node and gateway devices were first physically connected to a laptop as serial COM devices. The test setup can be seen in figure 4.1.

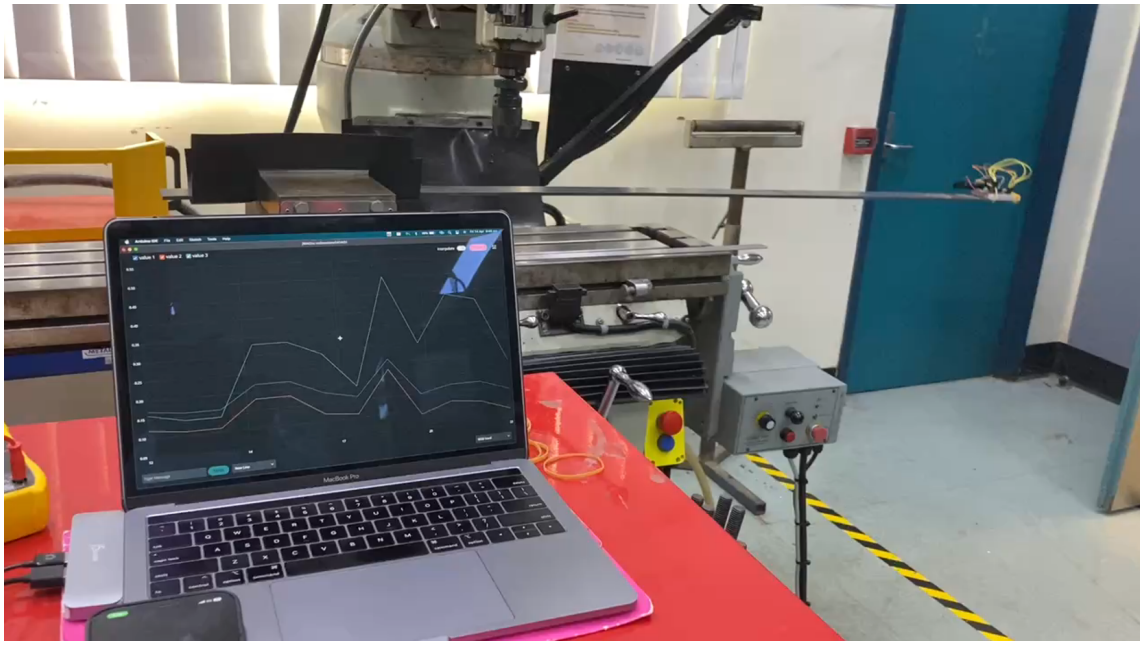
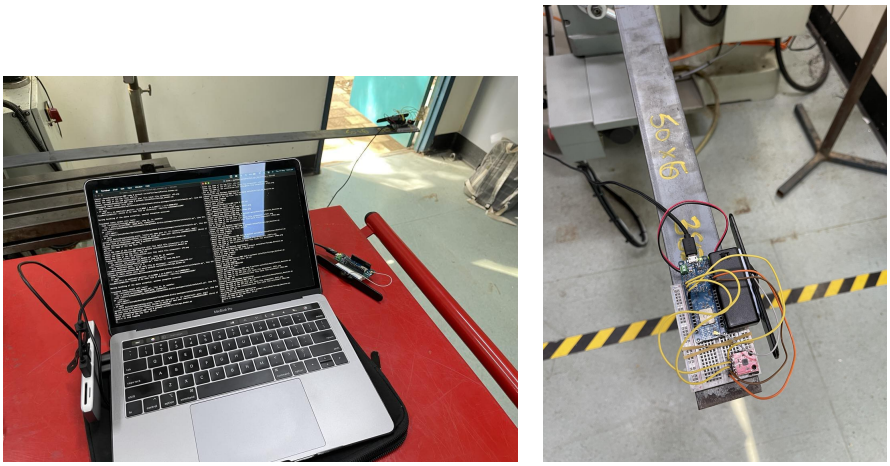


Fig. 4.1. Laboratory Beam Test Setup

Initially, the noise threshold values in the acceleration processing function were tuned through trial and error. The live serial plot in Arduino IDE assisted in tuning these parameters to a final value of 0.7 for the x-axis, 0.9 for the y-axis and 0.1 for the z-axis. The beam was then pushed down 15cm and released. Once the noise thresholds were acceptable, the first test was conducted.

Prototype 1: Test 1

Test one involved two participants, one to monitor the laptop and run the software and another to push the beam down. The first test was conducted with both devices connected to the laptop via USB. The node device was fastened to the beam and the reset button was pushed, calibrating the device. The node device on the beam can be seen in figure 4.2b. The sender and receiver Python logging scripts were executed on the laptop, as seen in figure 4.2a, and the beam was released. The test ran for approximately two minutes until the beam reached a stationary state. At this moment the logging programs were ended and the sender and receiver Python plotting scripts were executed. The names of each logging file was input by the user and the raw acceleration, maximum acceleration and peak frequency were displayed in a pyplot graph. These graphs were named and saved as PNG files.



(a) Logging Python Scripts (b) Node Device on Beam

Fig. 4.2. Close Up of Laptop and Node with Serial Connection in Beam Test

Prototype 1: Test 1 Results

Figure 4.5 shows the raw acceleration from the serial connection between the node device and laptop and figures 4.6 and 4.7 show the maximum acceleration and maximum frequency from the serial connection between the gateway device and the laptop. The raw and maximum acceleration are characteristic of the beam accelerating with maximum displacement and decaying over time to the stationary position. The peak frequency from the FFT is 2.4 Hz which matches the first mode fundamental frequency achieved in the Strand7 FE simulation shown in figures 4.3 and 4.4. This verifies that the FFT computation can accurately identify the maximum peak frequency.

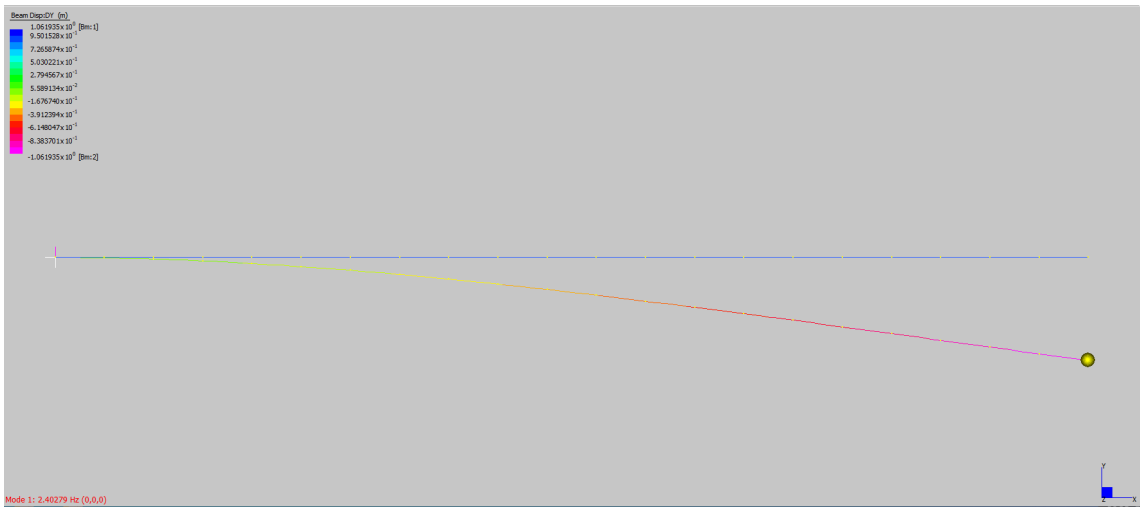


Fig. 4.3. Beam Test FE Simulation

Natural Frequency Log:C:\Users\s5216336\OneDrive - Griffith University\Desktop\EMA steel beam\...

```
THE FIRST 10 EIGENVALUES HAVE CONVERGED

FINAL FREQUENCY RESULTS
Mode        Eigenvalue       Frequency (rad/s)     Frequency (Hz)
   1      2.27925063E+02      1.50971872E+01      2.40279198E+00
   2      8.92651206E+03      9.44802205E+01      1.50369941E+01
   3      1.58281294E+04      1.25809894E+02      2.00232665E+01
   4      6.98265808E+04      2.64247196E+02      4.20562475E+01
   5      2.67572072E+05      5.17273692E+02      8.23266649E+01
   6      6.19896671E+05      7.87335171E+02      1.25308284E+02
   7      6.55906766E+05      8.09880711E+02      1.28896518E+02
   8      7.29770478E+05      8.54266047E+02      1.35960664E+02
   9      1.62550381E+06      1.27495247E+03      2.02914988E+02
  10      3.16498882E+06      1.77904154E+03      2.83143256E+02

TOTAL CPU TIME              : 0.297 Seconds

Solution completed on 07/06/2023 at 17:09:47
Solution time: 1 Second

SUMMARY OF MESSAGES
Number of Notes    : 0
Number of Warnings : 0
Number of Errors   : 0
```

7,985 bytes 176 lines

Fig. 4.4. Beam Test FE Simulation Results

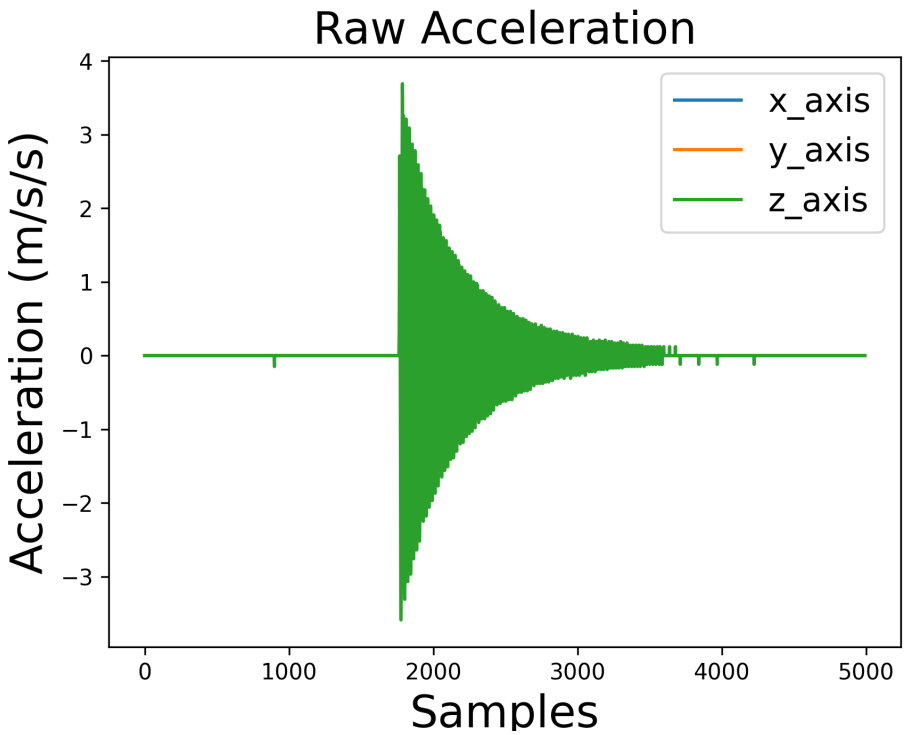


Fig. 4.5. Prototype 1: Test 1 Raw Acceleration

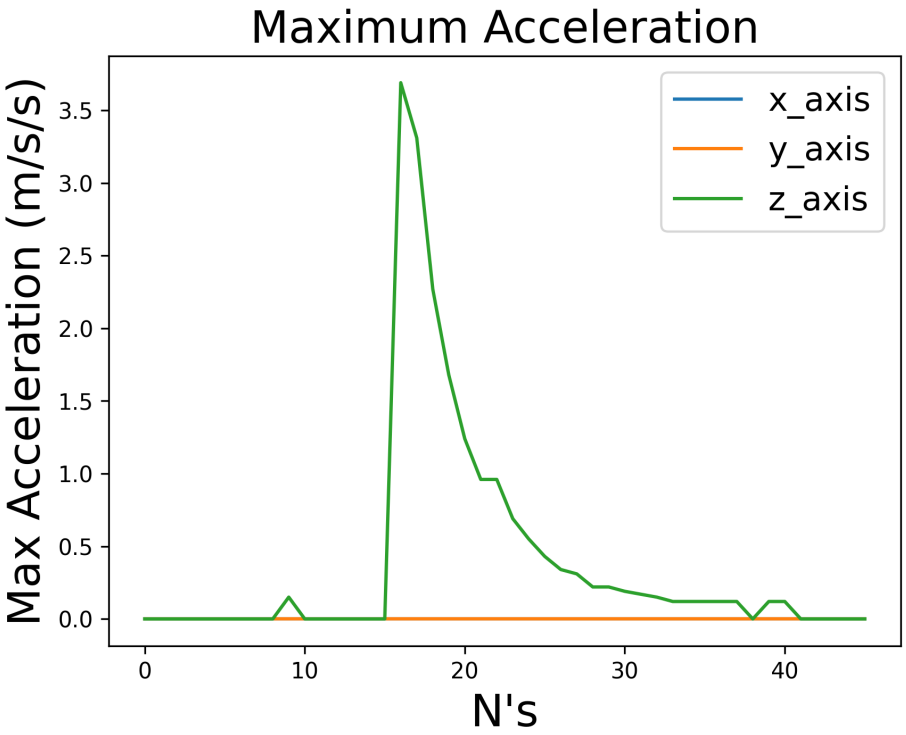


Fig. 4.6. Prototype 1: Test 1 Maximum Acceleration

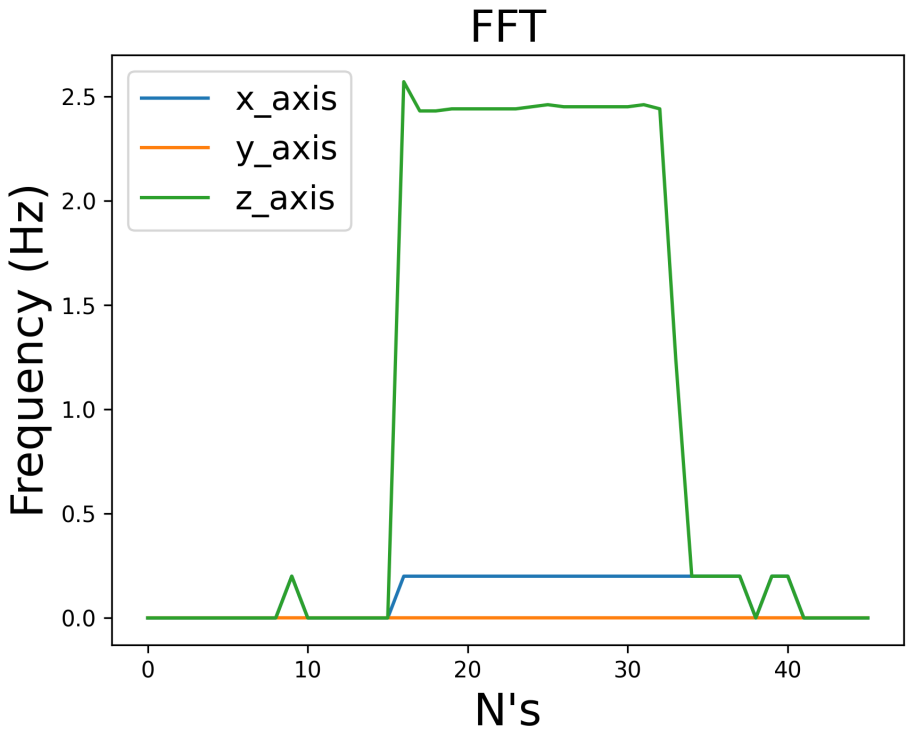


Fig. 4.7. Prototype 1: Test 1 Maximum Frequency

Prototype 1: Test 2

Test two removed the serial connection between the node device and the laptop and used the battery pack to power the device instead. The aim of this test was to achieve the same results as test one with a different input voltage and no serial connection for the node device. To initiate the test, the battery pack was switched on, and the reset button on the device was pushed, calibrating the accelerometer with the new reference voltage. This time only the receiver Python logging script was executed following the release of the beam. Once the beam reached its resting state the logging script was ended and the receiver Python plotting script was executed. The maximum acceleration and maximum frequency were plotted using pyplot and saved as PNG files. Figure 4.8 displays the node device on the beam with no physical connection to the laptop.

Prototype 1: Test 2 Results

Figures 4.9 and 4.10 show the maximum acceleration and maximum frequency from the serial connection between the gateway device and the laptop. These results show the same response as test one which verify the capability of the device running from the battery pack and successful calibration of the accelerometer when using a lower reference voltage of 3.0V.

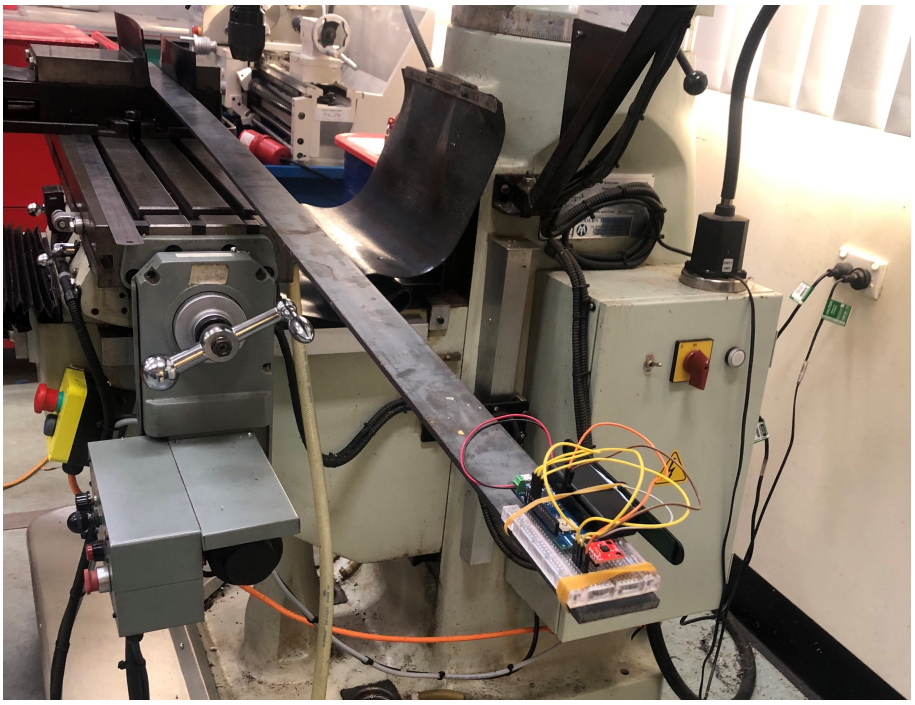


Fig. 4.8. Wireless Node in Beam Test

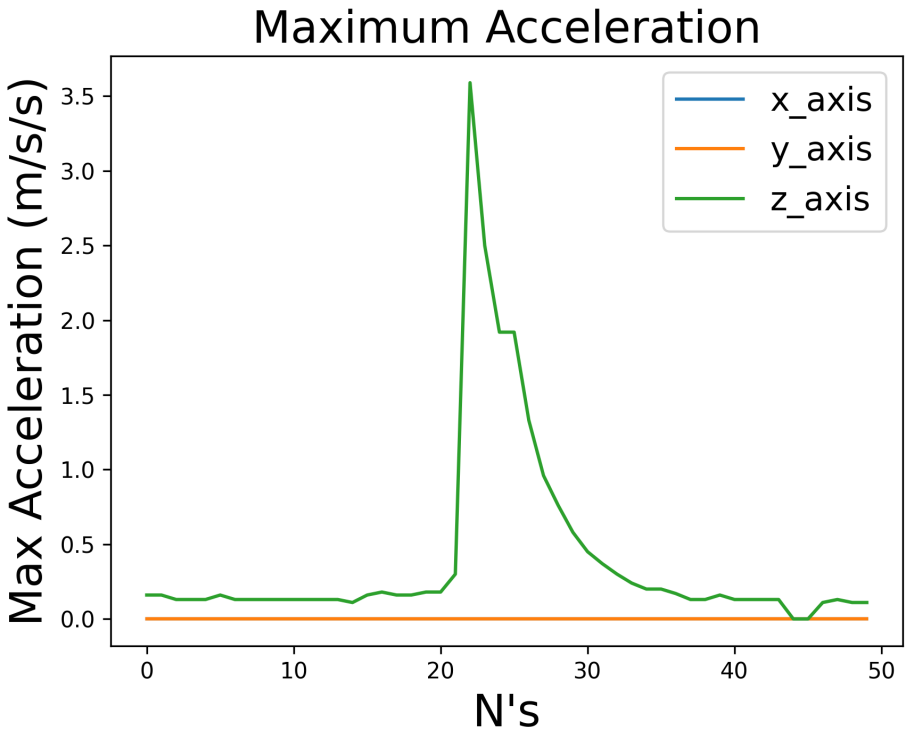


Fig. 4.9. Prototype 1: Test 2 Maximum Acceleration

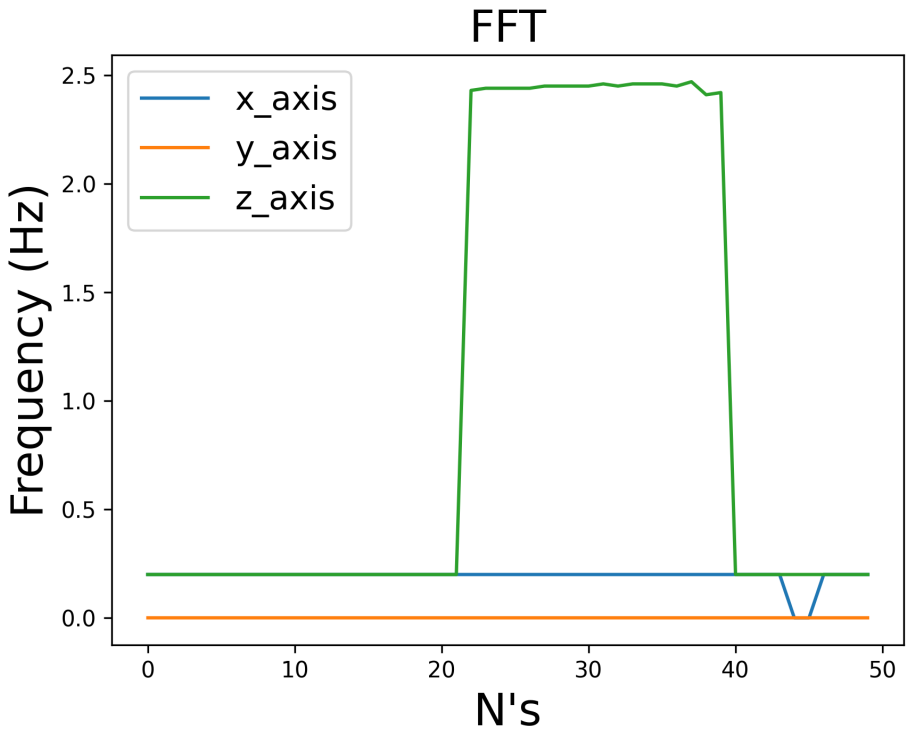


Fig. 4.10. Prototype 1: Test 2 Maximum Frequency

Prototype 1: Test 3

The purpose of test three was to test the range of the communication between the two devices. To do this one participant stayed in the laboratory and the other carried the laptop away with the gateway node which was still connected serially. A Microsoft Teams call was used to communicate between the two participants during this test. The beam was continuously pushed down and the data was observed on the Arduino IDE's live serial plotter. The test was ended once the gateway node was unable to receive LoRa packets and the location was recorded.

Prototype 1: Test 3 Results

Figure 4.11 shows the maximum range of successful communication between the two devices. The range was measured between the mechanical engineering laboratory and the start of the Griffith footbridge, reaching a distance of approximately 200m.

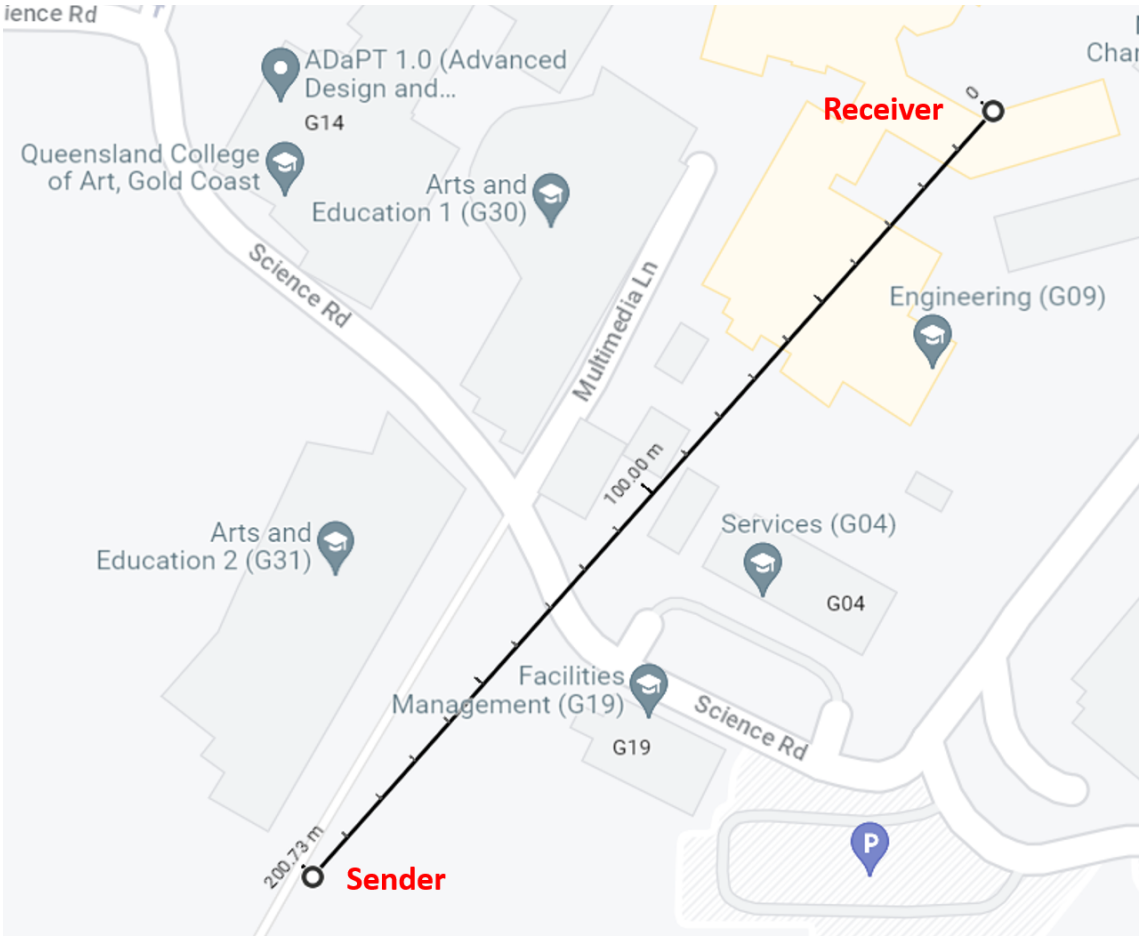


Fig. 4.11. Prototype 1: Test 3 Range [7]

4.1.2. Prototype 2: Bridge Test

The purpose of the bridge test was to test a full deployment of IoT architecture for structural health monitoring and to verify the first mode flexural frequency stated in the Griffith footbridge documentation. This test involved installing the enclosure behind the hand rail of the footbridge and setting up a base station for data collection. The base station was placed 25m away from the node device and consisted of a power generator, laptop and LoRaWAN gateway. Similar to the prototype one testing, the noise threshold values were fine tuned over the course of several experiments. The Arduino IoT Cloud dashboard was used to monitor the data live feed, figure 6.4, and the TNN live data page, figure 6.3, was used to observe incoming LoRa packets. The base station can be seen in figure 4.12 and the enclosure installation can be seen in figure 3.16. The testing was conducted between 4:00 pm and 6:00pm and the bridge was subjected to moderate pedestrian load from the university and a nearby school.

Fig. 4.12. Base Station Setup

Prototype 2: Test 1

Test one was conducted over a thirty minute period with noise threshold values 0.05 for the x-axis, 0.05 for the y-axis and 0.03 for the z-axis. The purpose of this test was to determine if these noise threshold values were suitable for achieving the expected frequency response of the bridge and to observe the signal strength over time. It was discovered in this test that the enclosure lid was too thick and prevented packet transmission so it was removed for testing. Packets were sent from the node device every sixty seconds and these points were plotted in the Arduino IoT Cloud dashboard. Two participants were required for this experiment, one to monitor the incoming packets on the TNN live data page at the base station, and one to safeguard the enclosure on the bridge. Figure 4.13 displays the orientation of each axis with respect to the enclosure placement on the bridge.

Fig. 4.13. Prototype 2: Test 1 Enclosure Bridge Axis

Prototype 2: Test 1 Results

Figures 4.14 and 4.15 show the maximum average acceleration and maximum average frequency over a thirty minute period. Figures 4.16 and 4.17 show the SNR and RSSI of the signal over the testing duration. Unlike the beam experiment, the x-axis was designed to represent the vertical frequency component of the bridge and so the noise threshold value of 0.03 was determined to be too low. The peaks in the acceleration and frequency values are representative of the pedestrian load during testing.

The SNR during this test fluctuated between 10 dB to 15 dB which is characteristic of a good connection. Positive SNR for LoRaWAN indicates that the signal can be successfully demodulated from the noise [21]. LoRa SNR usually lies between -20 dB and 10 dB, with SNR closer to 10 dB characterising a good connection [22]. The RSSI values averaged values between -70 dBm and -85 dBm. RSSI values for LoRa are more resilient than other communication technologies such as 2G and 3G Wi-Fi which list the range of -70 dBm to -85 dBm as a strong signal with good data speeds [23]. The signal drops to -110 dBm on the fourth packet which is very low, however LoRa is resilient enough to transmit packages as low as -120 dBm, so the signal is not fully disconnected at this point.

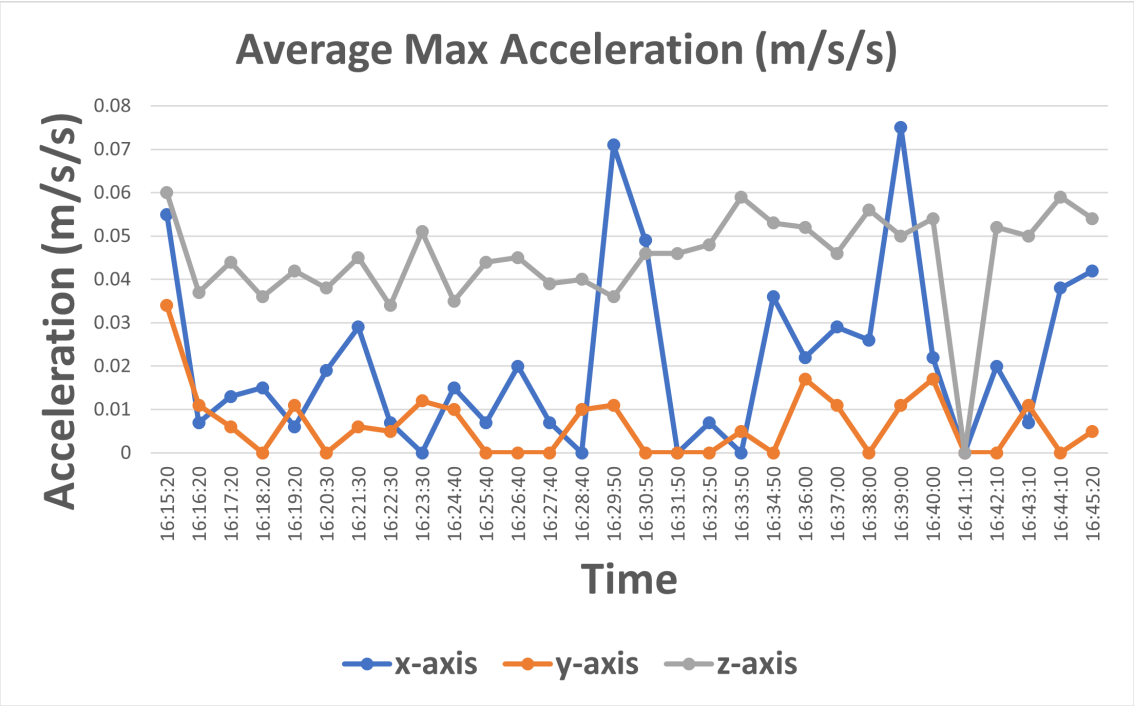


Fig. 4.14. Prototype 2: Test 1 Maximum Average Acceleration

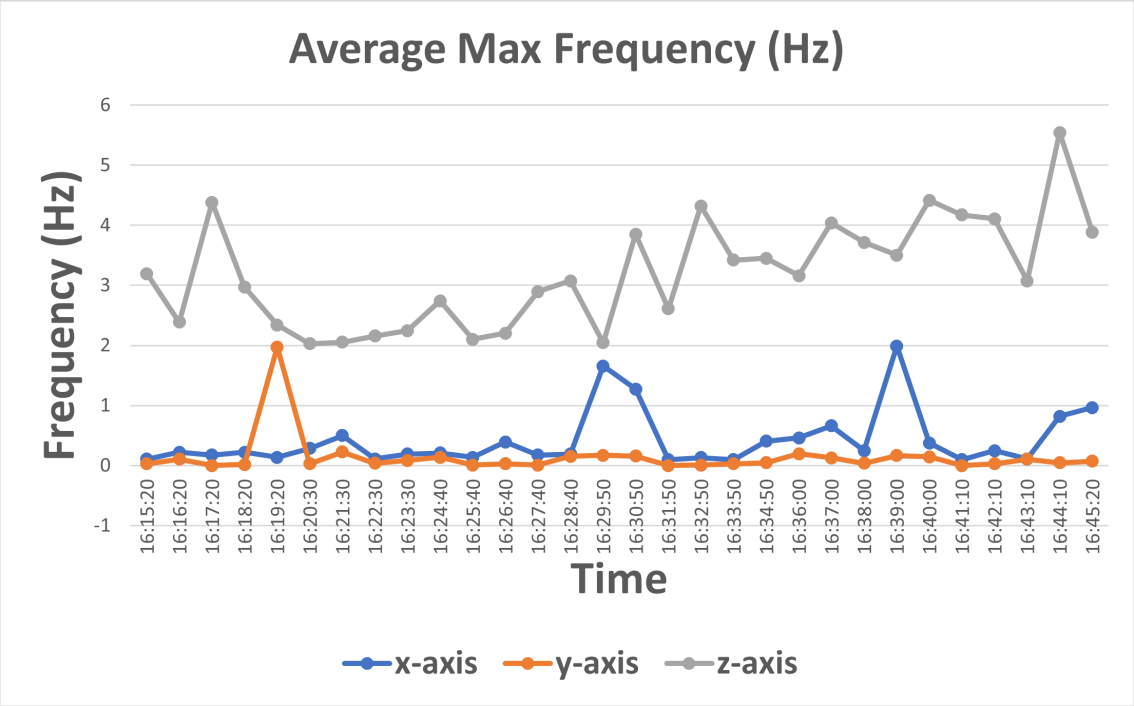


Fig. 4.15. Prototype 2: Test 1 Maximum Average Frequency

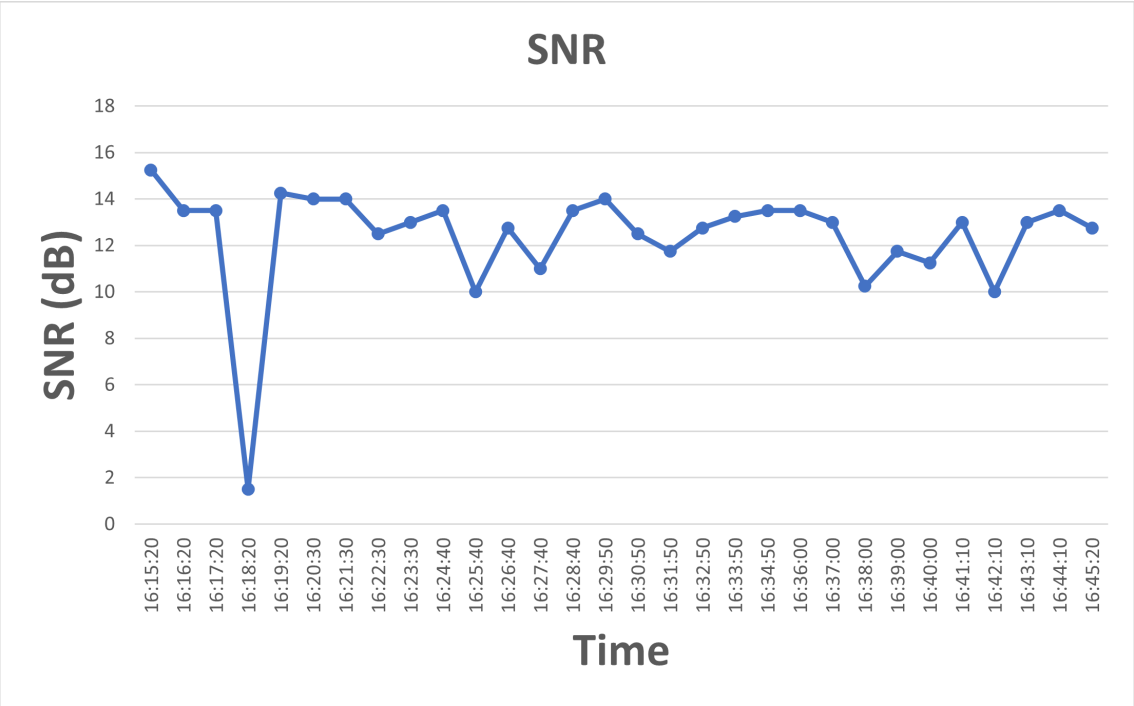


Fig. 4.16. Prototype 2: Test 1 SNR

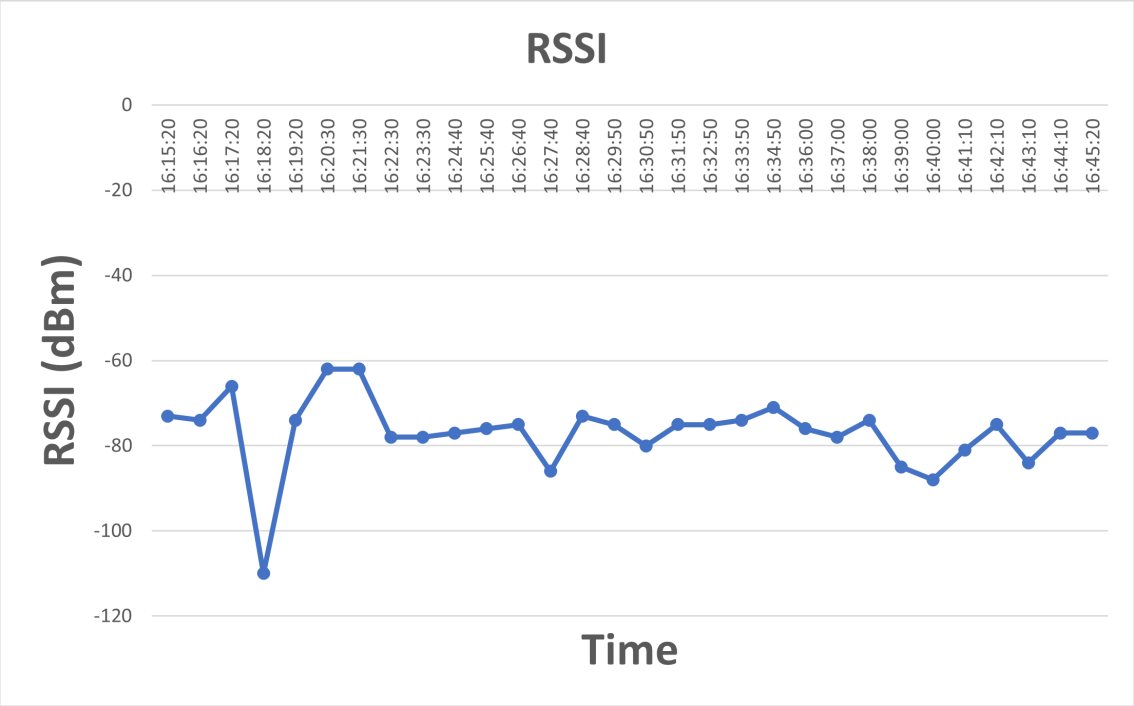


Fig. 4.17. Prototype 2: Test 1 RSSI

Prototype 2: Test 2

Test two was conducted over a thirty minute period with the same noise thresholds as test one. This time, the enclosure was removed from the rail and placed flat on the floor of the bridge. This was done to gain more context about the results from experiment one and insight for the noise threshold values used in test three. Figure 4.18 shows the enclosure placement for this test. Packets were again transmitted every sixty seconds and the values were observed in real time on the Arduino IoT Cloud dashboard. One participant was assigned to monitoring the incoming packets on the TNN live data page at the base station, and one to watch over the enclosure on the bridge. Special attention was required for this test since the enclosure was placed flat on the ground and was at risk of pedestrian interference.

Fig. 4.18. Prototype 2: Test 2 Enclosure Placement

Prototype 2: Test 2 Results

At the conclusion of test one, it was uncertain as to whether the lack of x-axis frequency was truly due to the lower noise threshold or if the z-axis was actually the result of the side of the bridge absorbing frequency components of the x-axis. To resolve this uncertainty the enclosure was placed flat on the bridge, making the z-axis representative of the vertical frequency component (test one x-axis). Since the noise thresholds were unchanged from

test one, test two simultaneously tested the existence of vertical frequency components on the bridge, and the effectiveness of using a lower noise threshold for the x-axis from test one.

The z-axis displayed a frequency that was much more representative of the expected fundamental frequency of 2.2 Hz specified in the Griffith footbridge documentation [24]. The frequency hit maximum average peaks of just over 2.5 Hz and exhibited most peaks within the 2-2.5 Hz range. The readings were also much more reactive to human induced vibrations caused by pedestrian load.

The SNR values again existed between the range of 14 dB to 10 dB which was slightly improved over test one. This can be attributed to the fact that the enclosure was lying flat on the ground with the antenna facing the direction of the base station. The RSSI was similar to test one however there was no drop below -100 dBm which was a slight improvement. Under the LoRaWAN specifications these SNR and RSSI values were characteristic of a typical LoRa signal strength.

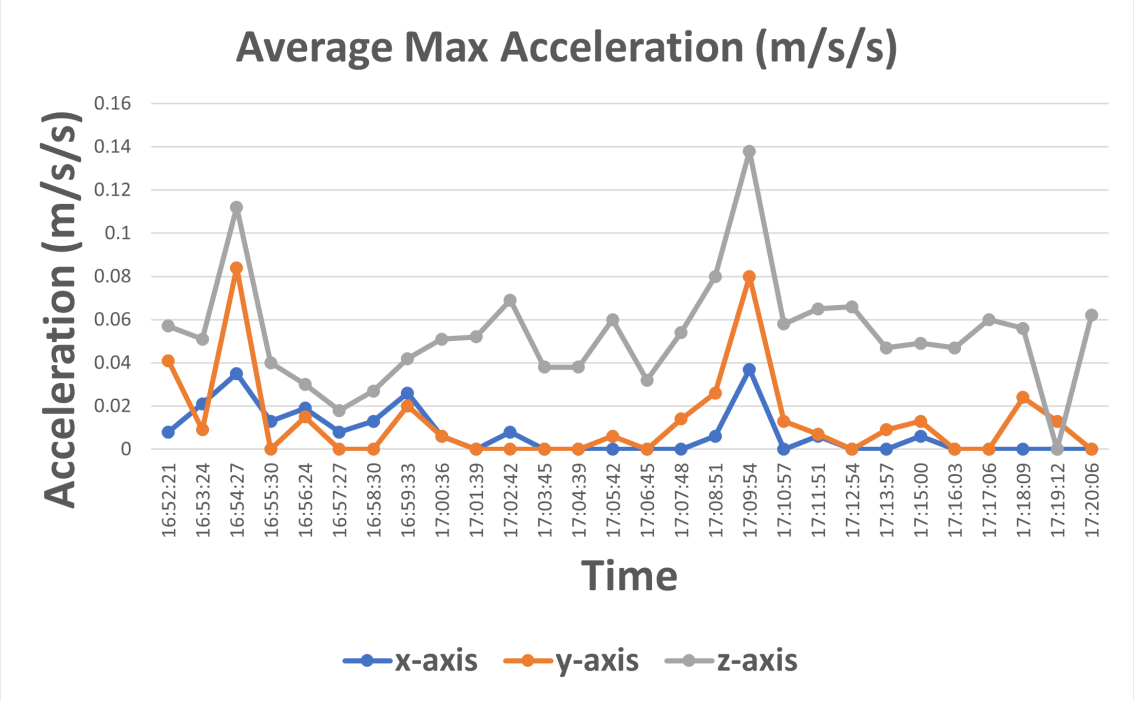


Fig. 4.19. Prototype 2: Test 2 Maximum Average Acceleration

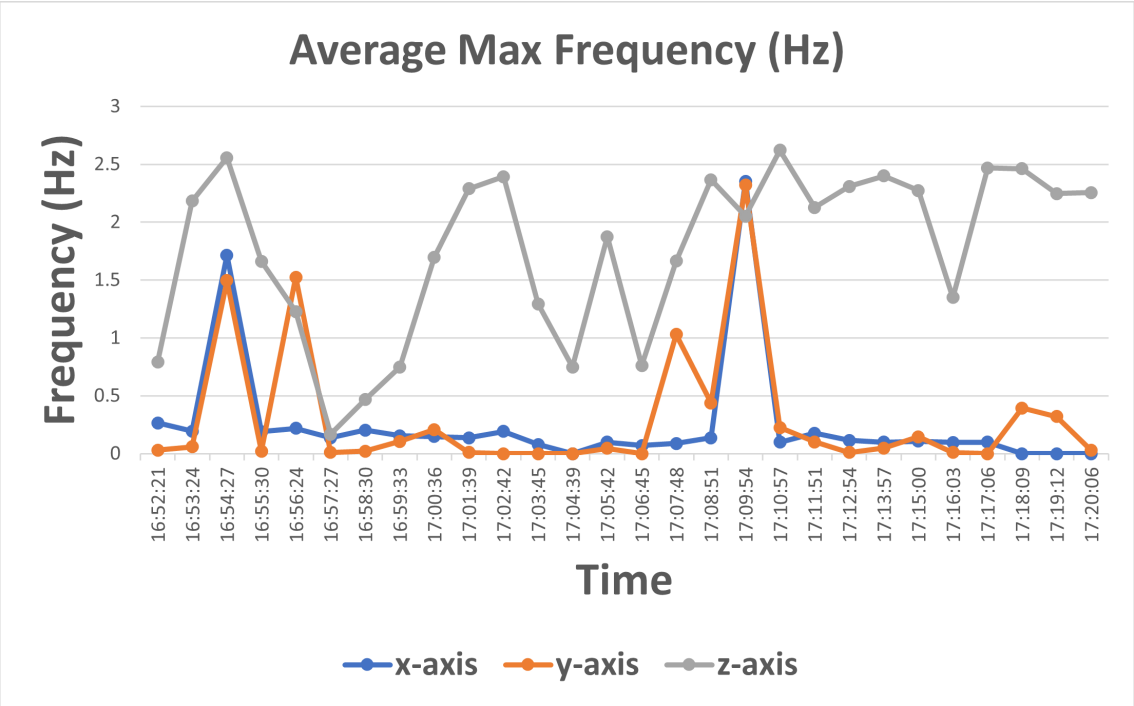


Fig. 4.20. Prototype 2: Test 2 Maximum Average Frequency

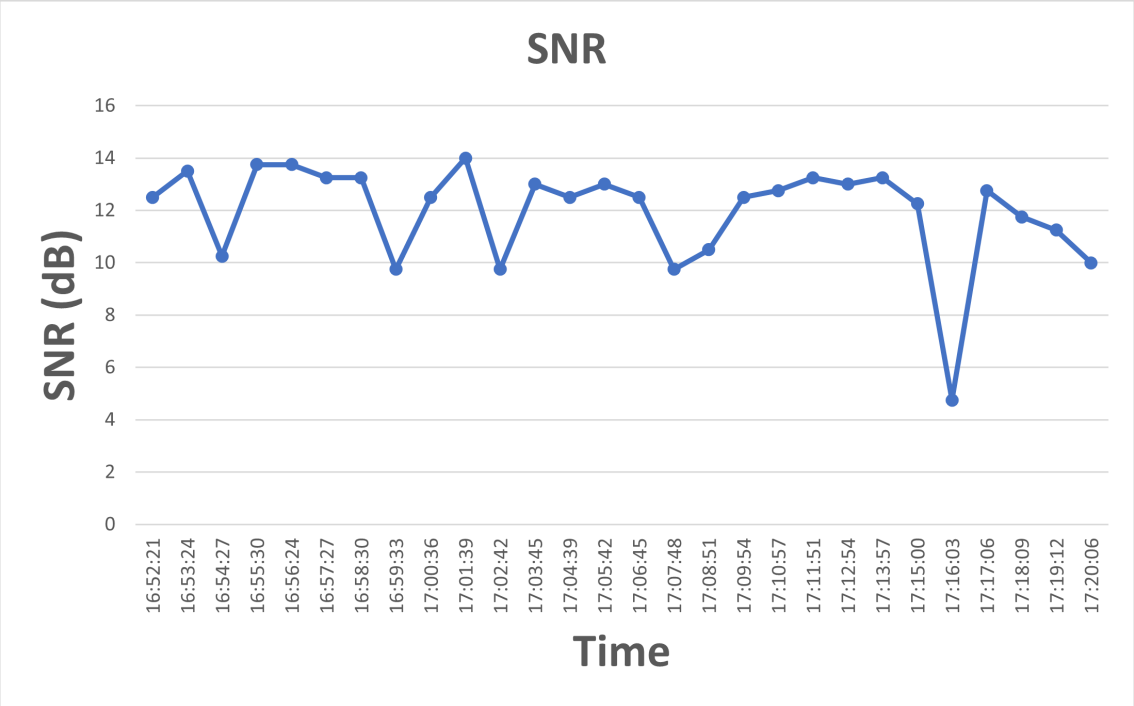


Fig. 4.21. Prototype 2: Test 2 SNR

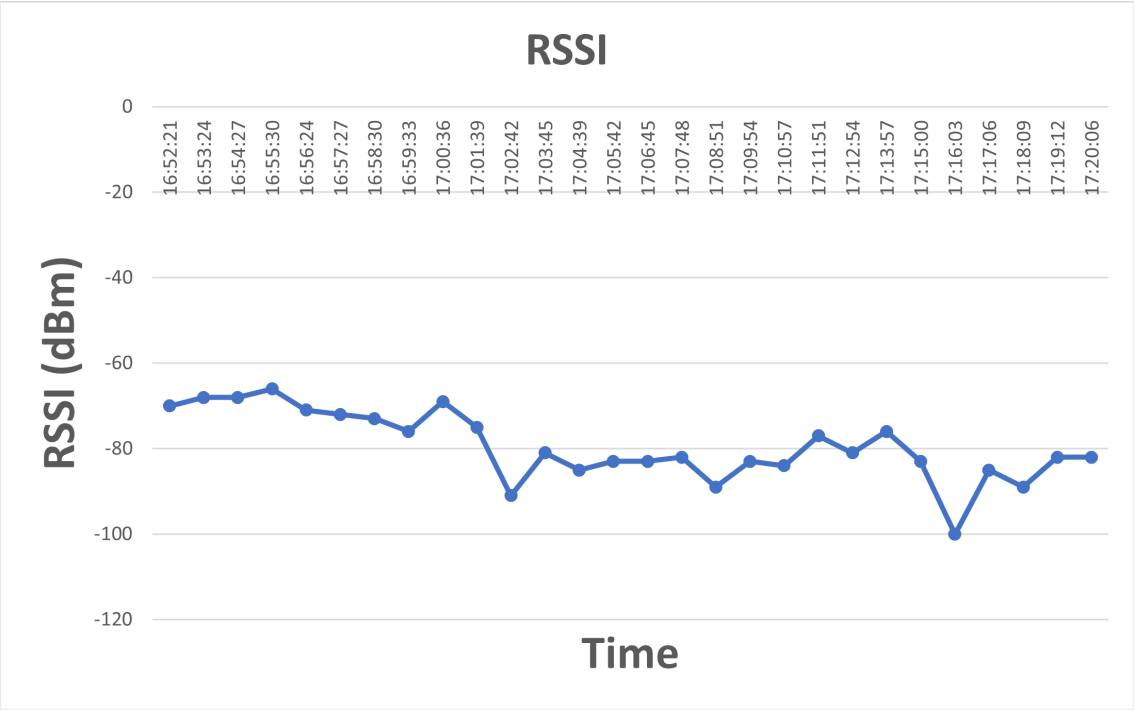


Fig. 4.22. Prototype 2: Test 2 RSSI

Prototype 2: Test 3

Test three was also conducted over a thirty minute period and the enclosure was reinstalled onto the bridge handrail. In this test the noise thresholds were changed to 0.03 for each axis. Packets were transmitted every sixty seconds and each participant fulfilled the same roles specified in test one and two.

Prototype 2: Test 3 Results

Test three was conducted with decreased noise thresholds of 0.03 for each axis. Compared to test one, there was a significantly improved response in the x-axis with an average maximum frequency value of 2.5 Hz. This value is also more stable compared to the previous tests with peak frequencies lying between 2 Hz and 3.5 Hz whilst showing the response characteristics of increased vibration from pedestrian load due to the resonance phenomenon. With the insight from all three experiments, it is clear that a noise threshold value of 0.03 for the x-axis was much more representative of the expected 2.2 Hz for the first mode flexural frequency of the bridge. It is also apparent that 0.03 is too low for the other axis as there are elements of cross talk and noise in the y-axis and z-axis frequency response. The noise thresholds for these axis need to be increased to become more representative of the bridge's y-axis and z-axis frequency components.

In terms of SNR and RSSI, similar values from test one and two were achieved. There was

a a -5 dB drop in SNR and -120 dBm drop in RSSI during this test which was the lowest signal drop in all three experiments. It is clear that the direction of the antenna plays a role in the signal strength as tests one and two involve the antenna facing upwards. Regardless of the drop, the resilient nature of LoRaWAN means that the packet was not completely dropped and the signal was able to resume normal functionality. Notably, this drop occurred at 17:43:40 which corresponds to a strong peak in the average max acceleration. This peak may therefore be attributed to noise or corruption and may not be indicative of the bridge's actual acceleration response.

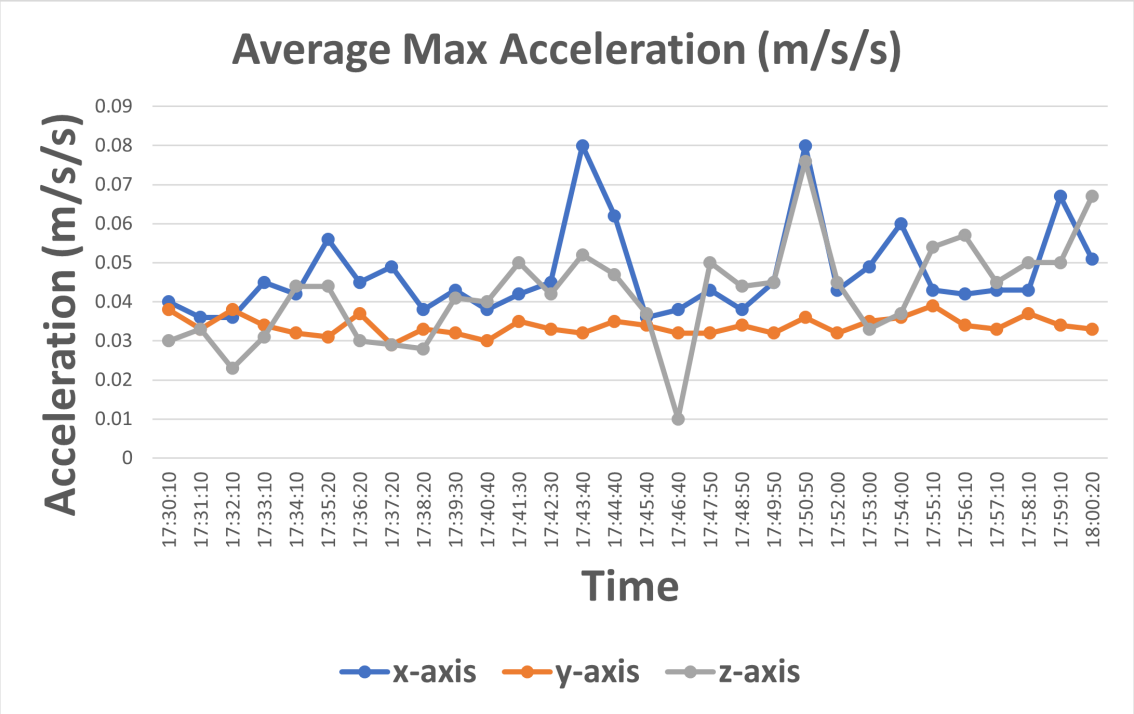


Fig. 4.23. Prototype 2: Test 3 Maximum Average Acceleration

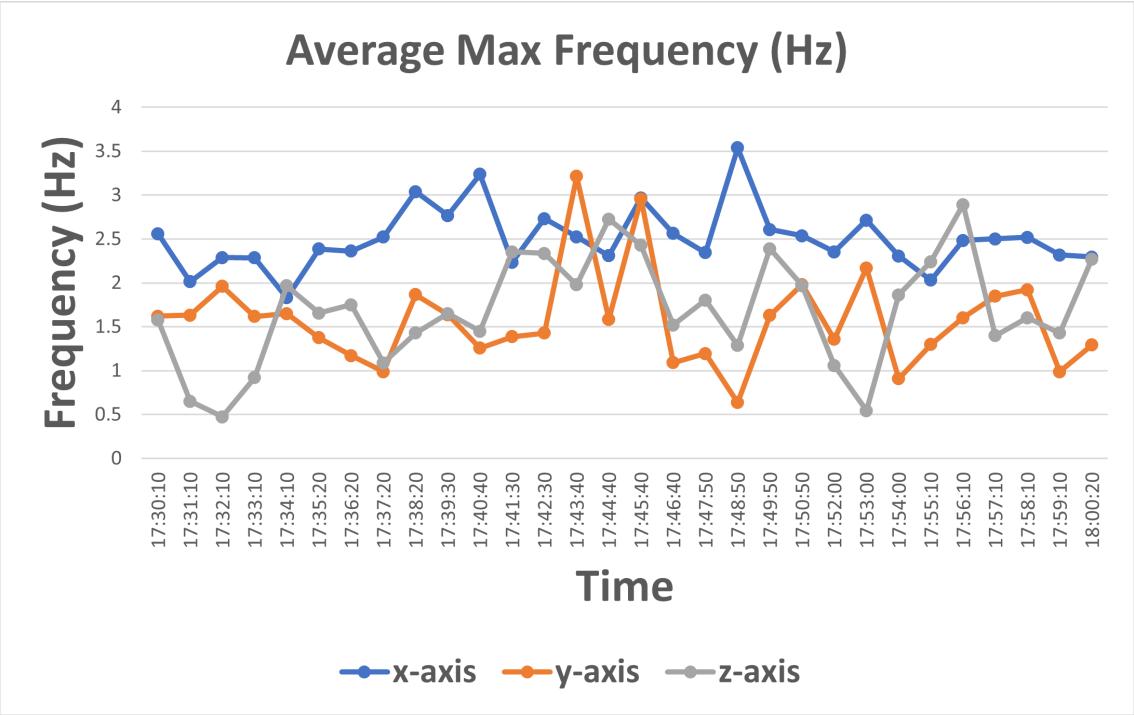


Fig. 4.24. Prototype 2: Test 3 Maximum Average Frequency

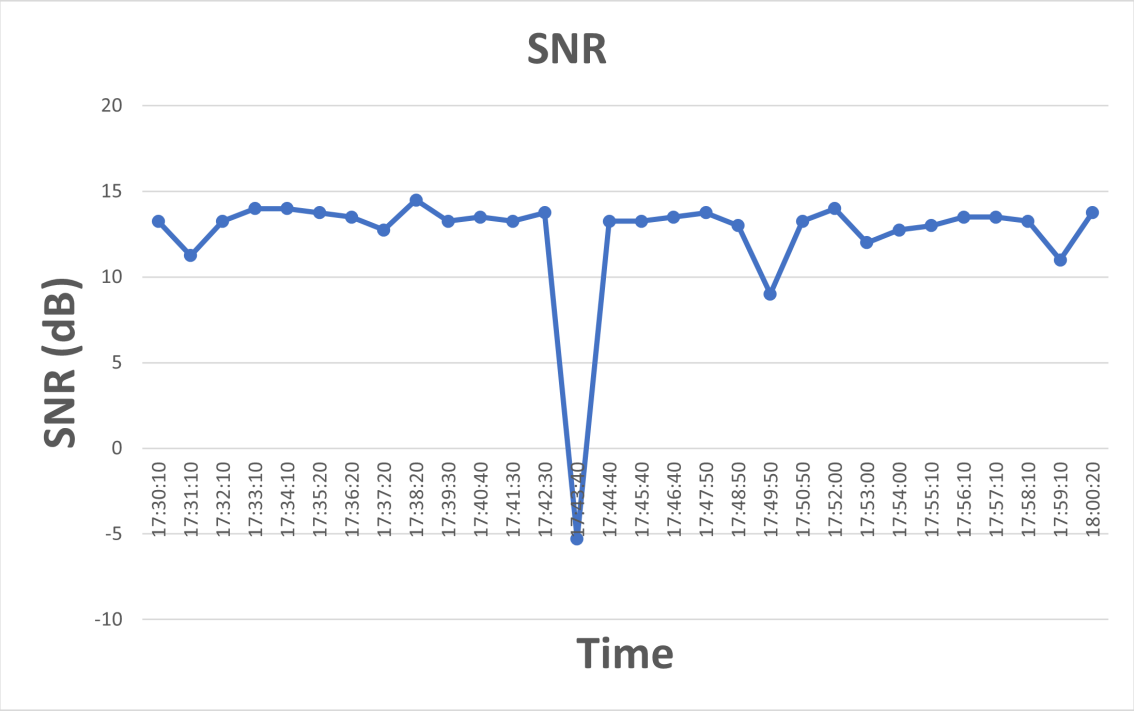


Fig. 4.25. Prototype 2: Test 3 SNR

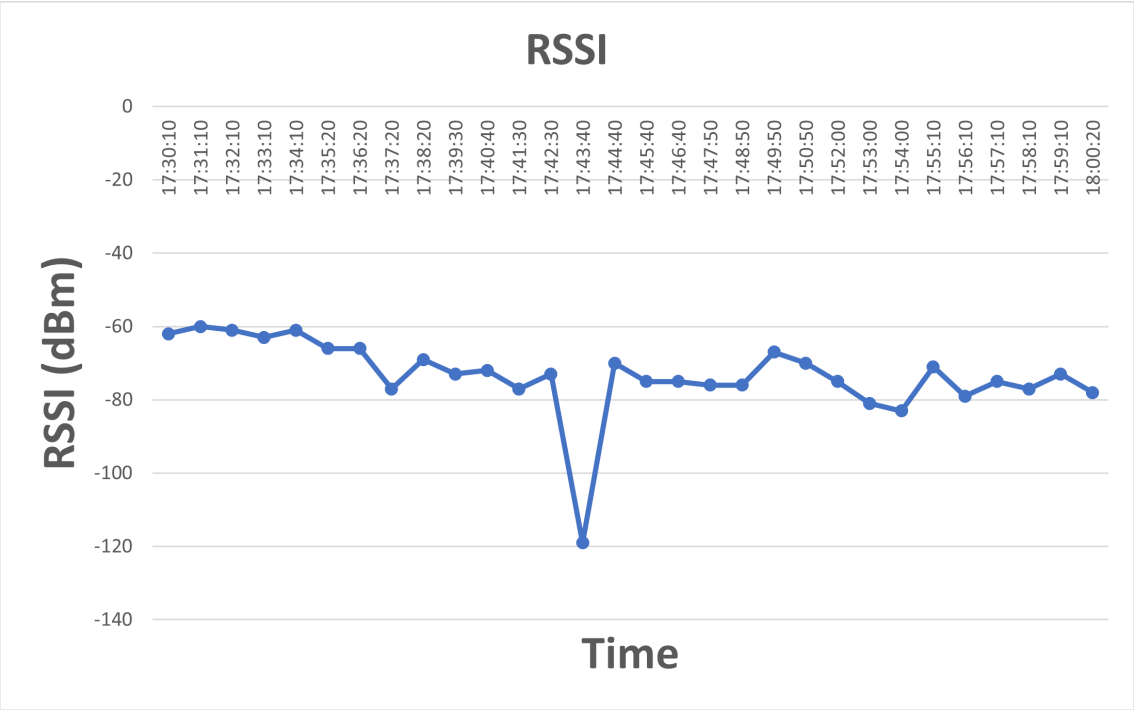


Fig. 4.26. Prototype 2: Test 3 RSSI

5. DISCUSSIONS

5.1. Effectiveness of LoRa Node

The beam laboratory and Griffith footbridge testing demonstrated the effectiveness of LoRa nodes in structural health monitoring as per the following points:

5.1.1. FFT Accuracy and Device Calibration

The laboratory beam testing confirmed the effectiveness of the FFT computation within the software and the ability to accurately identify peak maximum frequencies. This accuracy was verified with comparison to the first mode fundamental frequency achieved in the Strand7 FE simulation as both achieved peak frequency of 2.4 Hz. Additionally, the device's functionality remained consistent when powered by a battery pack and operating under a lower reference voltage. This illustrates the device's adaptability to different power sources and confirms the successful calibration of the accelerometer which is essential for real-world application and deployment.

5.1.2. Low Power

A multimeter was used to measure the current draw of the device during the FFT computation and LoRa packet transmission when using the battery pack. With 3.0 V input voltage from the two triple A batteries, the current draw of the device was approximately 23.8 mA. The batteries were listed at 1250 mAh which means that prototype two was capable of supporting 52.5 hours of deployment with constant computation load. This is a long lifetime for a prototype, highlighting the low-power nature of LoRa nodes, which can be improved through various metrics which are discussed in the recommendations section.

5.1.3. Noise Thresholds and Response

Adjustments to the noise thresholds in the acceleration processing greatly influenced the frequency response of the system. Tailoring these values allowed the system to align closely with the first mode fundamental frequency from the FE simulation in the prototype one laboratory beam test, and within 12.8% of the first mode flexural frequency in the prototype two bridge test. Further trial and error testing can result in a much higher accuracy by further fine tuning the noise threshold parameters.

5.1.4. Impact of Pedestrian Load

The sensitivity of prototype two to variations in pedestrian load indicate its capability to monitor and interpret external influences on the structure. This responsiveness can further the system's ability to analyse structural behaviour under varying conditions. The impact to the structural health of the bridge can be observed by logging the first mode flexural frequency over long periods of time in a future deployment. This can give indication as to how significant the impact of high pedestrian load or an accident has on the bridge's structural integrity.

5.2. LoRaWAN Signal Strength

The main limitations of this project exist in the communication range and signal strength of the prototype two deployment. Some characteristics of the LoRaWAN signal strength do show promising prospects for future design revisions which are discussed in the following points:

5.2.1. Communication Range

Prototype one test three confirmed a successful communication range of 200m, showcasing the system's potential for large-scale structural monitoring. However, there were issues involved in the test setup for prototype two. Initially the base station was deployed in a nearby building but was completely closed off to the outside by thick windows. After three to four successful packets, the connection was dropped. This indicates that the prototype two deployment was highly susceptible to obstacles and environmental factors.

This susceptibility is a result of the antenna, gateway and the environment. The antenna is rated at a gain of -1 dBi in the frequency band of 824 MHz - 960 MHz which restricts the directionality and range of the signal. The gateway was designed for indoor use and so it is likely that the gateway antenna was not designed to penetrate many obstacles such as the thick wall and windows of the enclosed building of the initial base-station deployment. When conducting the range test for prototype one, the door of the laboratory was left open so the test was not isolated from the outside environment. A combination of these factors resulted in a poor communication range for the prototype two implementation, but the exact maximum range for the node to gateway communication in a non-isolated environment is yet to be determined. A antenna with higher gain for the node device and the use of an outdoor gateway would be imperative to achieving a much higher communication range that is more characteristic of LoRaWAN specifications.

5.2.2. Antenna Orientation and Placement

The results from the prototype two testing indicate that the physical placement and orientation of the antenna on the node device influenced the signal strength and frequency response. Improved SNR and RSSI values were noted when the device was flat and the antenna was directed towards the base station. The enclosure was not specifically designed with this directionality in mind which is an aspect to consider for future revisions.

5.2.3. Data Corruption and Signal Resilience

Even in periods of reduced signal strength, the prototype system was capable of resuming normal functionality which highlights the robustness of LoRaWAN technology. Even with these periods of reduced signal strength, there was no packet loss due to the resilience of the protocol and its ability to transmit with RSSI values as low as -120 dBm **??**. The presence of a strong peak in average max acceleration coinciding with a significant signal drop in test three suggests potential data corruption or noise interference which warrants further investigation.

5.3. Recommendations

Moving towards future prototypes, revisions and deployments, there are a multitude of recommendations and strategies discovered over the course of this project that are outlined below.

5.3.1. Hardware Selection

A major limitation faced during the design process was the limited of on board memory for the MKRWAN1300. This is entirely dependent on the intended application of the device but features such as displacement measuring had to be cancelled due to the restricted memory. Besides this, the large amount of GPIO pins on the device and availability of Arduino packages do make the device a great choice as a LoRa node.

It is recommended that a suitable battery system is chosen for future node implementations. Solar powered batteries offer a great way to increase the battery-life of LoRa nodes. Improvements to the software can be made such as smart-sleeping so that the device is not continuously running computations and draining battery. Implementing a system where the default state of the device is sleep, and then waking up after a certain period of time to sample, compute and transmit would vastly increase battery-life. Improving upon these two aspects of battery and software will vastly increase the time between required maintenance for the LoRa nodes.

The major difficulty in the hardware selection came down to choosing a suitable gateway. Two gateways were provided by the university, one Arduino branded and the other Dragino. After attempting to use each of these gateways in the LoRaWAN system, it was discovered that neither of them were capable of connecting to TNN in Australia. The Arduino gateway was restricted to European frequencies and the Dragino gateway was not capable of LoRaWAN communication even though it was branded as a LoRa capable device. Therefore, the gateway must be carefully selected with respect to the region of deployment and interactivity with TNN or other network integrations. There is an Arduino branded version of the WisGate Edge Lite 2 which offers the same hardware as the gateway used in this project but with added Arduino firmware for easier integration with the Arduino IoT Cloud. This simplifies the setup of the gateway and devices, but it is advised that future deployments avoid using Arduino in general due to the restrictive nature of the network infrastructure.

5.3.2. Network Infrastructure

The Arduino IoT Cloud offered an easy to use interface for managing IoT things, cloud variables, dashboards and the TNN back-end integration. However, there are numerous bugs that were found during the implementation. The first bug was the use of an incorrect masking string in the things properties header file that is automatically generated. This is a bug because the TNN integration only works in Australia using the AU915 frequency sub band two which is specified when adding the device to the Arduino IoT cloud. The correct mask is ‘ff000001f000ffff00020000‘.

Another bug found in the Arduno IoT cloud was the inability to download cloud variable history from the dashboard. Since the cloud variables offer up to three months of data retention with the maker subscription, the cloud variable history at any date can be selected and downloaded to a CSV file. However the cloud variables used in this project were unable to be selected. After contacting Arduino they were able to isolate the bug and implement a fix, but not before the data points for the bridge testing were added to a spreadsheet manually.

Besides the bugs, Arduino does have good support channels and are able to resolve issues with a fast turn around time. This is not the case for TNN. TNN is a community based platform that is run by volunteers and is not particularly beginner friendly. This makes it hard to fix issues, ask for help and stress test LoRa transmission. Therefore it is the recommendation that future research is conducted to find a suitable replacement. Services such as LoRa Server offer open-source components for building LoRaWAN networks and offers features such as adding integrations with different cloud providers. Amazon Web Services (AWS) and Message Queuing Telemetry Transport (MQTT) can be deployed to handle network messaging which offers the possibility of adding custom storage such as a Structured Query Language (SQL) database. This gives more control over the im-

plementation of the LoRaWAN system, and allows for more research into LoRaWAN communication without the same data policy restrictions enforced by TNN and Arduino IoT Cloud. Additionally, deploying this applications onto a local server eliminate the monthly subscription required for Arduino IoT Cloud and offers as much data retention as needed.

5.3.3. Data Analysis and Monitoring

In addition to acceleration and frequency response, the incorporation of observational analysis to the system would provide a better understanding of the structural responses of the bridge to pedestrian load. For example, the use of a camera with computer vision and machine learning classifiers could be used to count the number of pedestrians on the bridge at different time stamps to give better context to the vibrational responses. The insights from such an implementation can be further used to improve the predictive capabilities of the system.

The Arduino IoT Cloud dashboard was helpful in monitoring the testing results in real-time, but a more sophisticated network deployment as previously mentioned offers the capability of post-processing and adding a real-time alert system. Post-processing can be combined with observational analysis to remove noise characteristics evident in the frequency response, and the ability to add maximum threshold values in which non-outlier frequency peaks trigger an alert to the Griffith maintenance staff.

5.3.4. Antenna and Gateway Upgrades

As mentioned in the discussion, the limitations of communication range were attributed to the node antenna and gateway. RAK offers a WisGate Edge Pro outdoor gateway that offers a much larger connection range and is capable of handling transmission through more obstacles. An upgraded gateway matched with an upgraded node antenna and an improved network infrastructure can be used to truly unleash the long range capabilities of the LoRaWAN technology. This kind of upgraded infrastructure can support a vast number of LoRa devices which can be used to attain a more accurate vibrational profile of the bridge, and better indicators to the structural health. Upgrading the gateway and network infrastructure allows for channel hopping, which is sending data over multiple channels to handle larger data packets whilst adhering to Australian communication laws. An upgraded gateway can be placed in a strategical location on the roof of a building on the campus, which can be used to facilitate LoRaWAN systems not only on the Griffith footbridge but for numerous applications campus-wide.

5.3.5. Enclosure and PCB Design

A key area for improving the enclosure design is the thickness of the lid. The antenna used in this project was too weak to consistently transmit packets through the enclosure and hence the lid was removed for testing. The lid can also be redesigned to accommodate for a new antenna such as offering an opening for transmission. The enclosure can also be made shorter, but would require a redesign of the PCB with smaller dimensions. There is plenty of empty space on the PCB so this would be an easy fix to implement.

Implementing these recommendations and strategies can result in an advanced campus-wide LoRaWAN IoT deployment that is based on the insights of this project. Moving to more capable hardware for LoRa nodes, antennas and gateways, and more capable network infrastructure can result in a highly beneficial system not only capable of drawing accurate predictive analysis of the structural health of the Griffith footbridge, but also laying the foundation for any other data acquisition needs within the University. By continually understanding the requirements for LoRaWAN deployment and data acquisition, a more reliable, scalable and insightful IoT architecture can be implemented.

5.4. Conclusion

The extensive testing and analysis of LoRa nodes within the structural health monitoring of the Griffith University footbridge has substantiated the significant potential of LoRaWAN technology in structural health monitoring. This is evident through the accuracy of FFT computations, calibration of the accelerometer, substantial battery life of the prototypes, sensitivity of the data to pedestrian load and the adaptability to noise thresholds.

The laboratory beam testing demonstrated the successful implementation of FFT computations within the software to identify peak frequencies, which were found to be consistent with the fundamental frequency achieved in the Strand7 FE simulation. The battery life estimation also highlighted the potential for long-term, low-power monitoring which is critical for real-world applications.

In addition, the bridge testing indicated the sensitivity of the device to changes in pedestrian load, implying that prototype two was effective in monitoring the external influences on the bridge structure, providing insights into the structural behaviour under different load conditions.

However, there were constraints in terms of communication range and signal strength, primarily attributed to the indoor gateway, node antenna and environmental obstructions. The limited gain of the node antenna, combined with the initial confined indoor placement of the gateway resulted in challenges when attempting to maintain connection at even a small range.

With these findings it is clear that future wok needs to focus on specific improvements such as moving away from Arduino and TNN, and moving to more reliable, feature-rich alternative private deployments. This shift would allow for greater control over network messaging, local server deployment, eliminating subscriptions and offer more extensive data retention, promoting in house testing and future LoRa research.

The addition of observational analysis of pedestrian load, improvements in battery technology and software, and upgrades to the enclosure design are recommended for future revisions. Upgrading the node antenna and gateway would also be beneficial to extending the communication range and facilitate channel hopping, allowing for larger data packets and more reliable coverage.

Overall, the work conducted in this project lays the groundwork for future LoRaWAN-based IoT architecture, specifically in the field of structural health monitoring systems. It underscores the potential and versatility of LoRaWAN technology, but also highlights the need for careful consideration in hardware selection, network infrastructure and sophisticated data analysis methods. With the recommended upgrades, improvements and strategies, the existing system can be evolved into a formidable, high-capacity LoRaWAN network. This advanced infrastructure would not only revolutionize the structural health monitoring of the Griffith University footbridge, but could also potentially transform the entire landscape of large-scale, real-time structural health monitoring, serving as the blueprint for future IoT infrastructures in the domain of structural monitoring.

BIBLIOGRAPHY

[1] W. Chanwattanapong, S. Hongdumnuen, B. Kumkhet, S. Junon, and P. Sangmahamad, "Lora network based multi-wireless sensor nodes and lora gateway for agriculture application," in *2021 Research, Invention, and Innovation Congress: Innovation Electricals and Electronics (RI2C)*, 2021, pp. 133–136.

[2] A. Gehani, S. Harsha Shatagopam, R. Raghav, M. Sarkar, and C. Paolini, "Application of 915 mhz band lora for agro-informatics," in *2021 Wireless Telecommunications Symposium (WTS)*, 2021, pp. 1–4.

[3] J. Fox, A. Donnellan, and L. Doumen, "The deployment of an iot network infrastructure, as a localised regional service," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 319–324.

[4] L. Maziero, T. B. Marchesan, C. H. Barriquello, D. P. Bernardon, F. G. Carloto, F. G. Reck, W. D. Vizzotto, and F. V. Garcia, "Monitoring of electric parameters in the federal university of santa maria using lorawan technology," in *2019 IEEE PES Innovative Smart Grid Technologies Conference - Latin America (ISGT Latin America)*, 2019, pp. 1–6.

[5] Z. Wang, Z. Xu, B. Dong, W. Xu, and J. Yang, "Dandelion: An online testbed for lora development," in *2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, 2019, pp. 439–444.

[6] "Structural health monitoring of a cable-stayed bridge using wireless smart sensor technology: data analyses," *Smart Structures and Systems*, vol. 6, no. 5-6, pp. 461–480, 2010. [Online]. Available: https://shm.cs.illinois.edu/Full%20Scale%20Papers/6.%20Cho%20et%20al.%202010%20Data%20Analyses.pdf

[7] G. Maps, "Griffith university," accessed on June 11, 2023. [Online]. Available: https://www.google.com/maps/place/Griffith+Univ+Bridge,+Southport+QLD/@-27.9640738,153.3817182,19z/data=!4m6!3m5!1s0x6b911005c06279cb:0x5c010c795a6b8c17!8m2!3d-27.9651242!4d153.3816698!16s%2Fg%2F11cjg942b1?entry=ttu

[8] M. Pule, A. Yahya, and J. Chuma, "Wireless sensor networks: A survey on monitoring water quality," *Journal of Applied Research and Technology*, vol. 15, no. 6, pp. 562–570, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1665642317301037

[9] H. Zhi-gang and C. Cai-hui, "The application of zigbee based wireless sensor network and gis in the air pollution monitoring," in *2009 International Conference on*

Environmental Science and Information Application Technology, vol. 2, 2009, pp. 546–549.

[10] C.-L. Hsu and S.-Y. Yang, "Active & intelligent energy-saving system designed with wsn modules and efficiency analysis," in *2010 International Computer Symposium (ICS2010)*, 2010, pp. 435–440.

[11] C. Scuro, P. F. Sciammarella, F. Lamonaca, R. S. Olivito, and D. L. Carni, "Iot for structural health monitoring," *IEEE Instrumentation & Measurement Magazine*, vol. 21, no. 6, pp. 4–14, 2018.

[12] C. Arcadius Tokognon, B. Gao, G. Y. Tian, and Y. Yan, "Structural health monitoring framework based on internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 619–635, 2017.

[13] D. K. Singh, H. Jerath, and P. Raja, *Internet of Things – Definition, Architecture, Applications, Requirements and Key Research Challenges*. John Wiley & Sons, Ltd, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119761785.ch16

[14] P. Chaudhari, A. K. Tiwari, S. Pattewar, and S. N. Shelke, "Smart infrastructure monitoring using lorawan technology," in *2021 International Conference on System, Computation, Automation and Networking (ICSCAN)*, 2021, pp. 1–6.

[15] B. Citoni, F. Fioranelli, M. A. Imran, and Q. H. Abbasi, "Internet of things and lorawan-enabled future smart farming," *IEEE Internet of Things Magazine*, vol. 2, no. 4, pp. 14–19, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8982742

[16] F. Mu'amar Wildan, E. A. Z. Hamidi, and T. Juhana, "The design of application for smart home base on lora," in *2020 6th International Conference on Wireless and Telematics (ICWT)*, 2020, pp. 1–6.

[17] D. Eridani, E. D. Widianto, R. D. O. Augustinus, and A. A. Faizal, "Monitoring system in lora network architecture using smart gateway in simple lora protocol," in *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2019, pp. 200–204.

[18] A. J. Wixted, P. Kinnaird, H. Larijani, A. Tait, A. Ahmadinia, and N. Strachan, "Evaluation of lora and lorawan for wireless sensor networks," in *2016 IEEE SENSORS*, 2016, pp. 1–3.

[19] S. Jang, H. Jo, S. Cho, K. Mechitov, J. Bridge, S.-H. Sim, H.-J. Jung, C.-B. Yun, B. Spencer, and G. Agha, "Structural health monitoring of a cable-stayed bridge using smart sensor technology: Deployment and evaluation," *Smart Structures and Systems*, vol. 6, 07 2010.

[20] "Airtime calculator for lorawan," accessed on June 11, 2023. [Online]. Available: https://avbentem.github.io/airtime-calculator/ttn/au915/24,12

[21] TNN, "Rssi and snr," accessed on June 12, 2023. [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/rssi-and-snr/

[22] E. B., "Lora," 2018, accessed on June 12, 2023. [Online]. Available: https://lora.readthedocs.io/en/latest/#:~:text=Typical%20LoRa%20RSSI%20values%20are,%3D%2D120dBm%3A%20signal%20is%20weak.

[23] Teltonika, "Rssi," 2020, accessed on June 12, 2023. [Online]. Available: https://wiki.teltonika-networks.com/view/RSSI

[24] J. C. Steele, L. Chong, and E. Newman, "Griffith university cable stayed pedestrian bridge," 2009.

6. APPENDIX

6.1. Bill of Materials

6.1.1. Prototype 1 Bill of Materials

Item	No.	Total Cost	Supplier	Function
Arduino MKRWAN1300	2	$135.04	Arduino	Microcontroller used as LoRaWAN node. Responsible for acceleration and frequency computations, and payload transmission.
Arduino RF antenna (X000016)	test2	$8.74	DigiKey	RF antenna used for LoRa packet transmission.
Accelerometer (ADXL335)	1	$26.44	DigiKey	Accelerometer used for acceleration sampling.
Breadboard 400 Tie points (PB8820)	1	$8.85	Jaycar	Breadboard used for transmitting node.
Breadboard 170 points (PB8817)	1	$5.95	Jaycar	Breadboard used for receiving node.
AAA battery holder	1	$4.50	Jaycar	Battery holder with switch connected to terminal block of the MKRWAN1300.
AAA lithium battery	2	$16.95	Jaycar	Powers MKRWAN1300 with total 3.0V.
Total Cost:		**$ 206.57**		

TABLE 6.1. PROTOTYPE 1 BILL OF MATERIALS

6.1.2. Prototype 2 Bill of Materials

Item	No.	Total Cost	Supplier	Function
Arduino Maker Subscription	1	$10.77	Arduino	Unlimited cloud variables and 3 month data retention.
M3 machine screw 20mm	25	$4.95	Core Electronics	Used to secure front and back plate of enclosure.
M3 machine screw 14mm	25	$3.35	Core Electronics	Used to secure lid of enclosure.
Heat-set insert soldering iron tip	1	$21.88	Core Electronics	Used to install heat-set inserts.
M3 brass heat-set inserts 4mm	50	$11.95	Core Electronics	Used to secure front and back plate, and lid of enclosure.
M2 screw and nut mounting kit	1	$23.95	Core Electrnoics	Used to mount PCB in enclosure.
M10 bolt and nut	1	$2.68	Bunnings Warehouse	Used to secure enclosure to bridge.
Threadlocker Loctite 10ml	1	$15.02	Bunnings Warehouse	Used to secure all screws.
M2 machine screw 30mm	20	$8.00	Bolt & Nut Australia	Used to mount PCB in enclosure.
RAKWireless WisGate Edge Lite 2 LoRaWAN Gateway (RAK7268)	1	$299.00	The IoT Store	LoRaWAN gateway.
PCB	5	$13.40	JLCPCB	Carrier board for microcontroller and accelerometer.
Total Cost:		**$ 414.95**		

TABLE 6.2. PROTOTYPE 2 BILL OF MATERIALS

6.2. System Diagrams

6.2.1. Python Sender Plotting Script

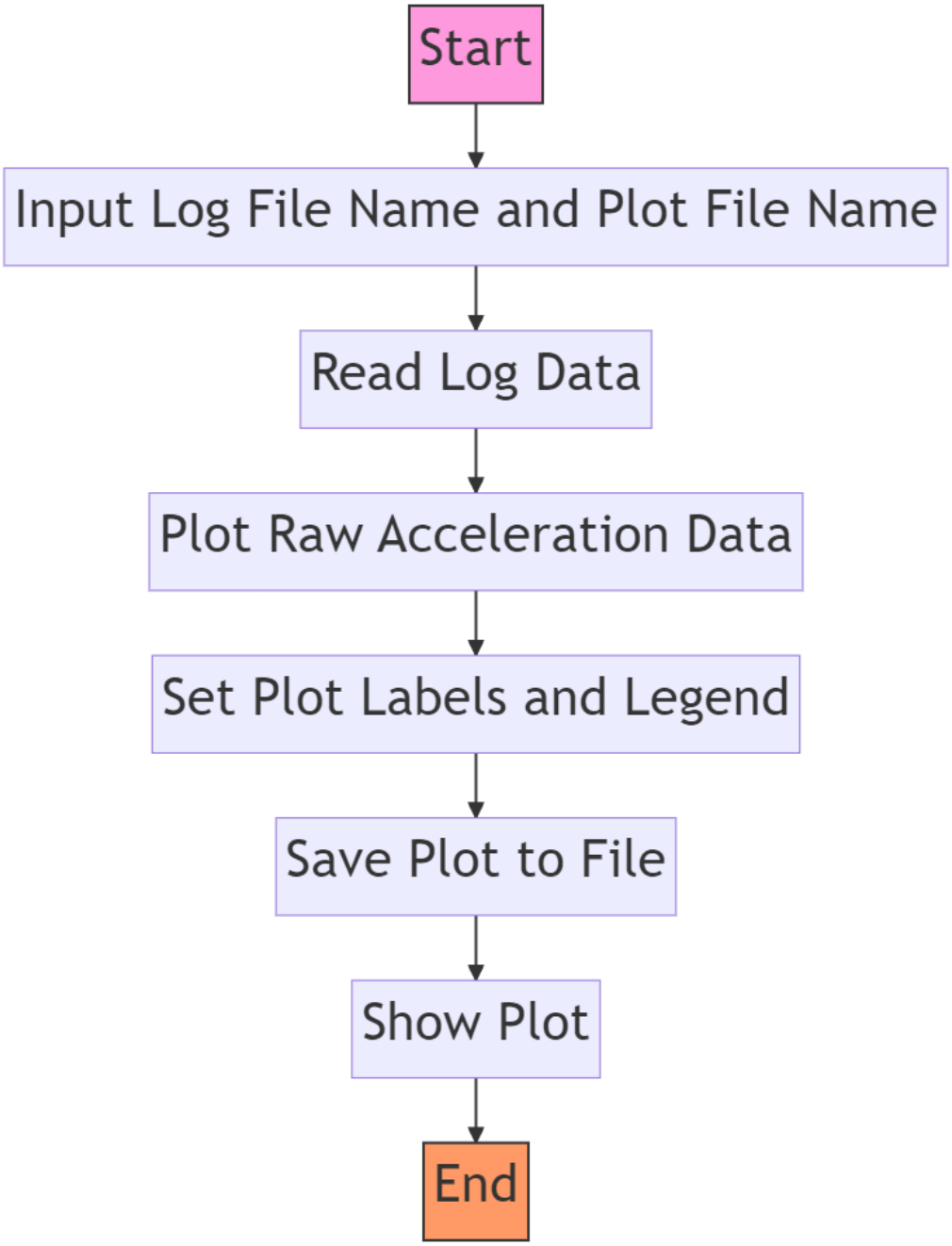


Fig. 6.1. Prototype 1 Software Diagram: Sender Plotting

6.2.2. Python Receiver Plotting Script

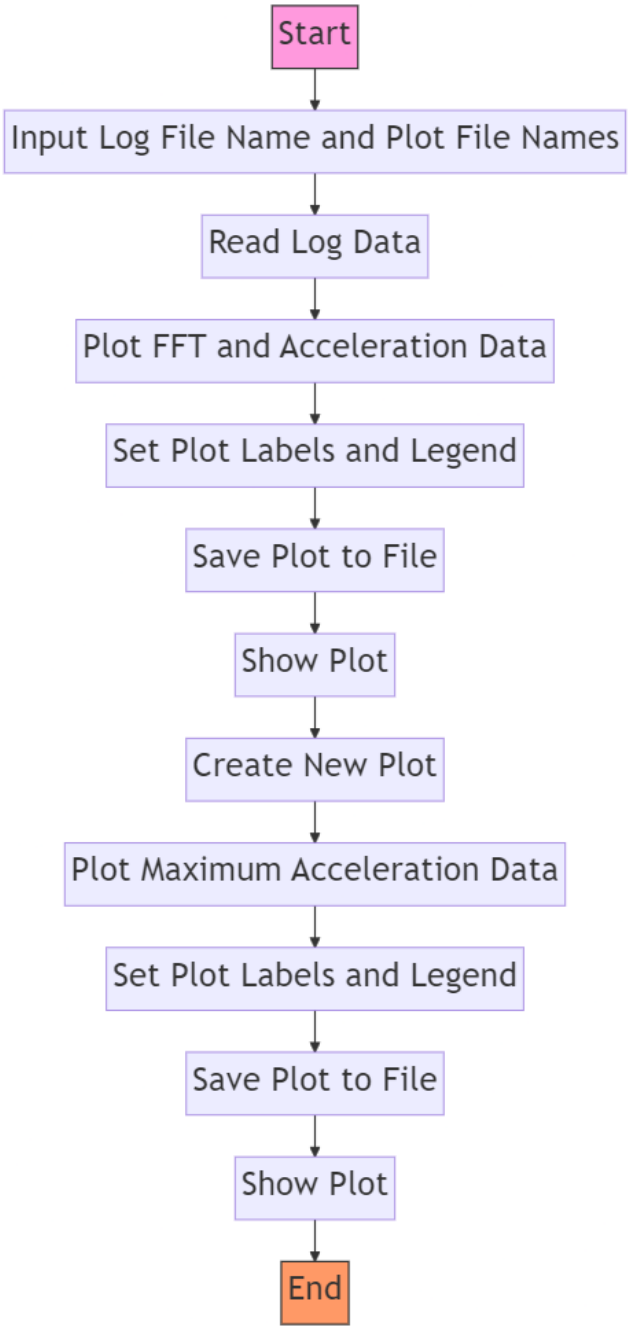


Fig. 6.2. Prototype 1 Software Diagram: Receiver Plotting

6.3. Programs

6.3.1. Prototype 1: Sender Main Program

```
#include "Sender.h"

double xZero = 0.0; double yZero = 0.0; double zZero = 0.0;

void setup() {
  // //Initialise Serial connection
  Serial.begin(9600);
  LoRa.begin(915E6);
  callibrate(256, xZero, yZero, zZero);
}

void loop() {
  // ---------- VARIABLES ----------
  // FFT SAMPLING
  double real_x_axis[sample_n]{}; double imag_x_axis[sample_n]{};
  double real_y_axis[sample_n]{}; double imag_y_axis[sample_n]{};
  double real_z_axis[sample_n]{}; double imag_z_axis[sample_n]{};

  // ACCELERATION SAMPLING
  double xAccel[sample_n]{}; double yAccel[sample_n]{};
  double zAccel[sample_n]{};

  // VELOCITY SAMPLING
  double xVel[sample_n]{}; double yVel[sample_n]{};
  double zVel[sample_n]{};

  // DISPLACEMENT SAMPLING
  double xDisp[sample_n]{}; double yDisp[sample_n]{};
  double zDisp[sample_n]{};

  // MAXIMUM ACCELERATION
  double maxAccelX = 0.0; double maxAccelY = 0.0;
  double maxAccelZ = 0.0;

  // MAXIMUM FREQUENCY
  double xFreq = 0.0; double yFreq = 0.0; double zFreq = 0.0;
  // ---------- VARIABLES ----------
```

```
// ---------- SAMPLE RAW DATA ----------
readRawData(real_x_axis, real_y_axis, real_z_axis);
// ---------- SAMPLE RAW DATA ----------

// ---------- TURN RAW DATA INTO ACCELERATION (m/s/s) ----------
processRawData(xAccel, yAccel, zAccel, real_x_axis, real_y_axis,
                real_z_axis, xZero, yZero, zZero);
// ---------- TURN RAW DATA INTO ACCELERATION (m/s/s) ----------

// ---------- SERIAL PRINT ACCELERATION (m/s/s) ----------
printAxisValues(xAccel, yAccel, zAccel);
// ---------- SERIAL PRINT ACCELERATION (m/s/s) ----------

// ---------- FIND MAX ACCELERATION ----------
maxAccelX = findMaxAbs(xAccel);
maxAccelY = findMaxAbs(yAccel);
maxAccelZ = findMaxAbs(zAccel);
// ---------- FIND MAX ACCELERATION ----------

// ---------- REMOVE HIGH FREQUENCIES ----------
lowPassFilter(xAccel);
lowPassFilter(yAccel);
lowPassFilter(zAccel);
// ---------- REMOVE HIGH FREQUENCIES ----------

// ---------- COMPUTE FFT ----------
arduinoFFT xFFT(xAccel, imag_x_axis, sample_n, sampling_rate);
arduinoFFT yFFT(yAccel, imag_y_axis, sample_n, sampling_rate);
arduinoFFT zFFT(zAccel, imag_z_axis, sample_n, sampling_rate);

xFFT.DCRemoval();
yFFT.DCRemoval();
zFFT.DCRemoval();

xFFT.Windowing(window_type, FFT_dir);
yFFT.Windowing(window_type, FFT_dir);
zFFT.Windowing(window_type, FFT_dir);

xFFT.Compute(FFT_dir);
yFFT.Compute(FFT_dir);
zFFT.Compute(FFT_dir);
```

```
  xFFT.ComplexToMagnitude();
  yFFT.ComplexToMagnitude();
  zFFT.ComplexToMagnitude();
  // ---------- COMPUTE FFT ----------

  // ---------- FIND HIGHEST FREQUENCY ----------
  xFreq = xFFT.MajorPeak();
  yFreq = yFFT.MajorPeak();
  zFreq = zFFT.MajorPeak();
  // ---------- FIND HIGHEST FREQUENCY ----------

  // ---------- CONVERT NAN TO 0 ----------
  checkNan(maxAccelX);
  checkNan(maxAccelY);
  checkNan(maxAccelZ);

  checkNan(xFreq);
  checkNan(yFreq);
  checkNan(zFreq);
  // ---------- CONVERT NAN TO 0 ----------

 // ---------- TRANSMIT LORA PACKET ----------
  LoRa.beginPacket();
  LoRa.write((uint8_t*)&xFreq, sizeof(xFreq));
  LoRa.write((uint8_t*)&yFreq, sizeof(yFreq));
  LoRa.write((uint8_t*)&zFreq, sizeof(zFreq));

  LoRa.write((uint8_t*)&maxAccelX, sizeof(maxAccelX));
  LoRa.write((uint8_t*)&maxAccelY, sizeof(maxAccelY));
  LoRa.write((uint8_t*)&maxAccelZ, sizeof(maxAccelZ));
  LoRa.endPacket();
  // ---------- TRANSMIT LORA PACKET ----------
}
```

6.3.2. Prototype 1: Sender Header File

```
#ifndef Sender_h
#define Sender_h

#include "Arduino.h"
#include <algorithm>
#include <math.h>
#include <MKRWAN.h>
#include <arduinoFFT.h>
#include <LoRa.h>
#include <stdio.h>
#include <iostream>

static constexpr double pi = 3.14159265358979323846;

//---------- Accelerometer Values ----------
static const int x_axis = A1;
static const int y_axis = A2;
static const int z_axis = A3;
static const int adc_resolution = 4096; //12 BIT ADC
static const double g = 9.81;
const double noiseThreshold = 0.15;
// CALIBRATION
// const double vRef = 3.3; //3.3V reference / supply
const double vRef = 3.0; // BATTERY CONNECTION
static const double sensitivity = 0.33; //330 mV/g at 3.3V
//---------- Accelerometer Values ----------

//---------- FFT Values ----------
static const int frequency = 2;
// THIS IS THE MAXIMUM FREQUENCY THAT THE FFT CAN DETECT
static const int maxFreq = 25;
static const uint16_t sample_n = 128; //MUST BE EXP 2
// SAMPLING RATE MUST BE AT LEAST TWICE THE MAX FREQUENCY
static const constexpr double sampling_rate = maxFreq * 2;
 //Gives sample interval in milliseconds
static constexpr double sample_interval = ((1.0/sampling_rate)
                                           * 1000);

// ---------- FILTER VALUES ----------
constexpr double HP_Fc = 0.5; // high pass cut-off frequency
```

```
constexpr double LP_Fc = 5; // low pass cut-off frequency
constexpr double sampling_period = 1.0 / sampling_rate;

constexpr double HP_alpha = (1 - (2 * pi * HP_Fc
                                  * sampling_period))
                            / (1 + (2 * pi * HP_Fc
                                  * sampling_period));
constexpr double LP_alpha = 1 / (1 + (2 * pi * LP_Fc
                                  * sampling_period));

const double alpha = 0.95;
// ---------- FILTER VALUES ----------

//FFT Function Values
// #define window_type FFT_WIN_TYP_HAMMING
#define window_type FFT_WIN_TYP_HANN
#define FFT_dir FFT_FORWARD
//---------- FFT Values ----------

// ---------- FUNCTIONS ----------
double findMaxAbs(const double data[]);
void lowPassFilter(double *data);
void highPassFilter(double *data);
void addGravity(double arr[]);
int findMaxIndex(double arr[]);
double findAvg(double arr[]);
void removeBias(double arr[]);
void integrate(const double arr[], double dt, double integrated[]);
void readRawData(double real_x_axis[], double real_y_axis[],
                 double real_z_axis[]);
void processRawData(double xAccel[], double yAccel[],
                    double zAccel[],
                    const double real_x_axis[],
                    const double real_y_axis[],
                    const double real_z_axis[], double xZero,
                    double yZero,
                    double zZero);
void integrate(const double xIn[], const double yIn[],
               const double zIn[], double xOut[],
               double yOut[], double zOut[]);
// ---------- FUNCTIONS ----------
```

```
// ---------- TEST FUNCTIONS ----------
void genSineWave(double arr[]);
void genZeroWave(double arr[]);
void printAxisValues(double xAccel[], double yAccel[],
                        double zAccel[]);
void printAxisVal(double xFreq, double yFreq, double zFreq);
void printAxisValueSingle(double accel[]);
void callibrate(int callibration_samples, double& xZero,
                double& yZero, double& zZero);
void checkNan(double& freq);
// ---------- TEST FUNCTIONS ----------

#endif
```

6.3.3. Prototype 1: Sender Functions

```
#include "Sender.h"

void callibrate (int callibration_samples, double& xZero,
                 double& yZero, double& zZero) {
    Serial.println("Calibrating Accelerometer...");

    double offsetX = 0;
    double offsetY = 0;
    double offsetZ = 0;

    for (int i = 0; i < callibration_samples; i++) {
        double rawX = analogRead(x_axis);
        double rawY = analogRead(y_axis);
        double rawZ = analogRead(z_axis);

        double voltageX = (rawX * vRef) / (adc_resolution - 1);
        double voltageY = (rawY * vRef) / (adc_resolution - 1);
        double voltageZ = (rawZ * vRef) / (adc_resolution - 1);

        double accelerationX = (voltageX - vRef / 2) / sensitivity;
        double accelerationY = (voltageY - vRef / 2) / sensitivity;
        double accelerationZ = (voltageZ - vRef / 2) / sensitivity - 1;

        offsetX += accelerationX;
        offsetY += accelerationY;
        offsetZ += accelerationZ;
```

```
        delay(10);
    }

    offsetX /= callibration_samples;
    offsetY /= callibration_samples;
    offsetZ /= callibration_samples;

    // ---------- MUST BE TURNED OFF IF PWR IS BATTERY ----------
    // Serial.print("Offset X: ");
    // Serial.println(offsetX, 6);
    // Serial.print("Offset Y: ");
    // Serial.println(offsetY, 6);
    // Serial.print("Offset ccoffsetZ: ");
    // Serial.println(offsetZ, 6);
    // ---------- MUST BE TURNED OFF IF PWR IS BATTERY ----------

    xZero = offsetX;
    yZero = offsetY;
    zZero = offsetZ;
}

double findMaxAbs(const double data[]) {
  double maxVal = 0;
  for (int i = 0; i < sample_n; ++i) {
    double absVal = abs(data[i]);
    if (absVal > maxVal) {
      maxVal = absVal;
    }
  }
  return maxVal;
}

double findMax(const float data[]) {
  double maxVal = 0;
  for (int i = 0; i < sample_n; ++i) {
    double absVal = data[i];
    if (absVal > maxVal) {
      maxVal = absVal;
    }
  }
  return maxVal;
}
```

```
void lowPassFilter(double *data) {
  // Initialize with the first value of the data array
  double filteredData = data[0];

  for (uint16_t i = 1; i < sample_n; i++) {
    filteredData = LP_alpha * data[i] + (1.0 - LP_alpha)
                   * filteredData;
    data[i] = filteredData;
  }
}

double findAvg(double arr[]) {
  double sum = 0.0;
  for (int i = 0; i < sample_n; i++) {
    sum += arr[i];
  }
  return sum / sample_n;
}

void removeBias(double arr[]) {
  double bias = findAvg(arr);
  for (int i = 0; i < sample_n; i++) {
    arr[i] -= bias;
  }
}

void readRawData(double real_x_axis[], double real_y_axis[],
                 double real_z_axis[]) {
  for (int i = 0; i < sample_n; i++) {
    // READ RAW DATA FROM ANALOG PINS
    real_x_axis[i] = analogRead(x_axis);
    real_y_axis[i] = analogRead(y_axis);
    real_z_axis[i] = analogRead(z_axis);
    // SAMPLING INTERVAL
    delay(sample_interval);
  }
}

void processRawData(double xAccel[], double yAccel[],
                    double zAccel[], const double real_x_axis[],
                    const double real_y_axis[],
```

```
                    const double real_z_axis[],
                    double xZero, double yZero, double zZero) {
  for (int i = 0; i < sample_n; i++) {
    // CONVERT ADC LEVEL TO A VOLTAGE
    double voltageX = ((real_x_axis[i] * vRef) / (adc_resolution - 1));
    double voltageY = ((real_y_axis[i] * vRef) / (adc_resolution - 1));
    double voltageZ = ((real_z_axis[i] * vRef) / (adc_resolution - 1));

    xAccel[i] = ((voltageX - vRef / 2) / sensitivity - xZero) * g;
    yAccel[i] = ((voltageY - vRef / 2) / sensitivity - yZero) * g;
    // Subtract gravity
    zAccel[i] = ((voltageZ - vRef / 2) / sensitivity - zZero - 1) * g;

    // // Apply threshold to remove noise
    xAccel[i] = (abs(xAccel[i]) <= 0.7) ? 0 : xAccel[i];
    yAccel[i] = (abs(yAccel[i]) <= 0.9) ? 0 : yAccel[i];
    zAccel[i] = (abs(zAccel[i]) <= 0.1) ? 0 : zAccel[i];
  }
}

void checkNan(float& freq) {
 if (isnan(freq)) {
  freq = 0;
 }
}
```

6.3.4. Prototype 1: Recevier Main Program

```
#include "Receiver.h"

void setup() {
  //Initialise Serial connection
  Serial.begin(9600);
  //Initialise LoRa Connection
  if (!LoRa.begin(915E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("LoRa Modem Started");
}

void loop() {
```

```
    if (LoRa.parsePacket()) {
    float receivedFreqX;
    float receivedFreqY;
    float receivedFreqZ;

    float receviedAccelX;
    float receviedAccelY;
    float receviedAccelZ;

    LoRa.readBytes((uint8_t*)&receivedFreqX, sizeof(receivedFreqX));
    LoRa.readBytes((uint8_t*)&receivedFreqY, sizeof(receivedFreqY));
    LoRa.readBytes((uint8_t*)&receivedFreqZ, sizeof(receivedFreqZ));

    LoRa.readBytes((uint8_t*)&receviedAccelX, sizeof(receviedAccelX));
    LoRa.readBytes((uint8_t*)&receviedAccelY, sizeof(receviedAccelY));
    LoRa.readBytes((uint8_t*)&receviedAccelZ, sizeof(receviedAccelZ));

    Serial.print(receivedFreqX);
    Serial.print(" ");
    Serial.print(receivedFreqY);
    Serial.print(" ");
    Serial.print(receivedFreqZ);

    Serial.print(" ");
    Serial.print(receviedAccelX);
    Serial.print(" ");
    Serial.print(receviedAccelY);
    Serial.print(" ");
    Serial.println(receviedAccelZ);
  }
}
```

6.3.5. Prototype 1: Sender Logger Program

```
import matplotlib.pyplot as plt
import serial
import time
import os

# MAC ADDRESS
# PORT = '/dev/tty.usbmodem1413301'
# PC ADDERSS
```

```
PORT = 'COM3'
BAUD_RATE = 9600
ser = serial.Serial(PORT, BAUD_RATE, timeout=1)

raw_xAccel = []
raw_yAccel = []
raw_zAccel = []

fname = input("Enter the name of the log file: ")

def read_serial_data():
    data = ser.readline().decode('utf-8').strip()
    if data:
        variables = data.split(' ')
        if len(variables) == 3 and all(variables):
            try:
                var1, var2, var3 = map(float, variables)
                return var1, var2, var3
            except ValueError:
                print("ERROR: INVALID DATA RECEIVED")
    return None, None, None

print("Ctrl-C to stop logging")

try:
    while True:
        var1, var2, var3 = read_serial_data()
        if var1 is not None and var2 is not None and var3 is not None:
            raw_xAccel.append(var1)
            raw_yAccel.append(var2)
            raw_zAccel.append(var3)
        time.sleep(0.01)

except KeyboardInterrupt:
    print("Interrupted by user.")

finally:
    # Close the serial port
    ser.close()
    print("Serial port closed.")

# Open and overwrite senderlog
```

```
dir_path = "serialPlot/logs/"
full_path = os.path.join(dir_path, fname)

f = open(full_path, "w")
for i in range(len(raw_xAccel)):
    f.writelines([str(raw_xAccel[i]), " ", str(raw_yAccel[i]), \
                  " ", str(raw_zAccel[i]), "\n"])
```

6.3.6. Prototype 1: Sender Plotter Program

```
import matplotlib.pyplot as plt
import time
import os

plt.close('all')

def read_log_data():
    raw_xAccel = []
    raw_yAccel = []
    raw_zAccel = []

    with open(full_path, "r") as f:
        for line in f:
            var1, var2, var3 = map(float, line.split())
            raw_xAccel.append(var1)
            raw_yAccel.append(var2)
            raw_zAccel.append(var3)
    return raw_xAccel, raw_yAccel, raw_zAccel

fname = input("Enter the name of the log file: ")
fname2 = input("Enter the name of the raw acceleration \
               plot file (with file extension): ")

dir_path = "serialPlot/logs/"
full_path = os.path.join(dir_path, fname)

dir_path2 = "serialPlot/plots/"
full_path2 = os.path.join(dir_path2, fname2)

raw_xAccel, raw_yAccel, raw_zAccel = read_log_data()

# Create the plot
```

```
plt.plot(raw_xAccel, label="x_axis")
plt.plot(raw_yAccel, label="y_axis")
plt.plot(raw_zAccel, label="z_axis")

# Set plot labels and legend
plt.title("Raw Acceleration")
plt.xlabel("Samples")
plt.ylabel("Acceleration (m/s/s)")
plt.legend()

# Save the figure as a high-resolution PNG file
plt.savefig(full_path2, dpi=300)

# Display the plot
plt.show()
```

6.3.7. Prototype 1: Receiver Logger Program

```
import matplotlib.pyplot as plt
import serial
import time
import os

# MAC ADDRESS
# PORT = '/dev/tty.usbmodem141401'
# PC ADDERSS
PORT = 'COM5'
BAUD_RATE = 9600
ser = serial.Serial(PORT, BAUD_RATE, timeout=1)

xFreq = []
yFreq = []
zFreq = []

xAccel = []
yAccel = []
zAccel = []

fname = input("Enter the name of the log file: ")

def read_serial_data():
    data = ser.readline().decode('utf-8').strip()
```

```
    if data:
        variables = data.split(' ')
        if len(variables) == 6 and all(variables):
            try:
                var1, var2, var3, var4, var5, var6 = map(float, variables)
                return var1, var2, var3, var4, var5, var6
            except ValueError:
                print("ERROR: INVALID DATA RECEIVED")
    return None, None, None, None, None, None

print("Ctrl-C to stop logging")

try:
    while True:
        var1, var2, var3, var4, var5, var6 = read_serial_data()
        if var1 is not None and var2 is not None and var3 is not None\
            and var4 is not None and var5 is not None and var6 is not None:
            xFreq.append(var1)
            yFreq.append(var2)
            zFreq.append(var3)

            xAccel.append(var4)
            yAccel.append(var5)
            zAccel.append(var6)
        time.sleep(0.01)

except KeyboardInterrupt:
    print("Interrupted by user.")

finally:
    # Close the serial port
    ser.close()
    print("Serial port closed.")

# Open and overwrite senderlog
dir_path = "serialPlot/logs/"
full_path = os.path.join(dir_path, fname)

f = open(full_path, "w")
for i in range(len(xFreq)):
    f.writelines([str(xFreq[i]), " ", str(yFreq[i]), " ", str(zFreq[i]), \
                  " ", str(xAccel[i]), " ", str(yAccel[i]), \
```

```
                    " ", str(zAccel[i]), "\n"])
```

6.3.8. Prototype 1: Receiver Plotter Program

```
import matplotlib.pyplot as plt
import serial
import time
import os

plt.close('all')

def read_log_data():
    xVar = []
    yVar = []
    zVar = []

    xAccel = []
    yAccel = []
    zAccel = []

    with open(full_path, "r") as f:
        for line in f:
            var1, var2, var3, var4, var5, var6 = map(float, line.split())
            xVar.append(var1)
            yVar.append(var2)
            zVar.append(var3)

            xAccel.append(var4)
            yAccel.append(var5)
            zAccel.append(var6)

    return xVar, yVar, zVar, xAccel, yAccel, zAccel

# Open and overwrite senderlog
fname = input("Enter the name of the log file: ")
fname2 = input("Enter the name of the FFT plot file \
               (with file extension): ")
fname3 = input("Enter the name of the accleration \
               plot file (with file extension): ")

dir_path = "serialPlot/logs/"
full_path = os.path.join(dir_path, fname)
```

```
dir_path2 = "serialPlot/plots/"
full_path2 = os.path.join(dir_path2, fname2)

dir_path3 = "serialPlot/plots/"
full_path3 = os.path.join(dir_path3, fname3)

xVar, yVar, zVar, xAccel, yAccel, zAccel = read_log_data()

print("Plotting FFT and Maximum Acceleration")

# Create the plot
plt.plot(xVar, label="x_axis")
plt.plot(yVar, label="y_axis")
plt.plot(zVar, label="z_axis")

# Set plot labels and legend
plt.title("FFT")
plt.xlabel("N's")
plt.ylabel("Frequency (Hz)")
plt.legend()

# Save the figure as a high-resolution PNG file
plt.savefig(full_path2, dpi=300)

# Display the plot
plt.show()

plt.figure()  # Create a new figure for the second plot
plt.plot(xAccel, label="x_axis")
plt.plot(yAccel, label="y_axis")
plt.plot(zAccel, label="z_axis")

# Set plot labels and legend
plt.title("Maximum Acceleration")
plt.xlabel("N's")
plt.ylabel("Max Acceleration (m/s/s)")
plt.legend()

# Save the second figure as a high-resolution PNG files
plt.savefig(full_path3, dpi=300)
```

```
# Display the second plot
plt.show()
```

6.3.9. Prototype 2: Sender Main Program

```
#include "arduino_secrets.h"

#include "thingProperties.h"
#include "Sender.h"

double xZero = 0.0; double yZero = 0.0; double zZero = 0.0;
//Store total max values for averaging later
float totalMaxAccelX = 0.0; float totalMaxAccelY = 0.0;
float totalMaxAccelZ = 0.0;
float totalMaxFreqX = 0.0; float totalMaxFreqY = 0.0;
float totalMaxFreqZ = 0.0;
//Record the number of computations within the time limit
int computationCount = 0;
//Timing variables
unsigned long previousMillis = 0;
const long interval = 60000; //test scenario 60 seconds

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor
  //without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
  delay(1500);

  callibrate(256, xZero, yZero, zZero);
  delay(1500);
}
```

```
void loop() {
  ArduinoCloud.update();

  // ---------- VARIABLES ----------
  double real_x_axis[sample_n] {};
  double imag_x_axis[sample_n] {};
  double real_y_axis[sample_n] {};
  double imag_y_axis[sample_n] {};
  double real_z_axis[sample_n] {};
  double imag_z_axis[sample_n] {};

  // ACCELERATION SAMPLING
  double xAccel[sample_n] {}; double yAccel[sample_n] {};
  double zAccel[sample_n] {};

  // MAXIMUM ACCELERATION
  float maxAccelX = 0.0; float maxAccelY = 0.0;
  float maxAccelZ = 0.0;

  // MAXIMUM FREQUENCY
  float xFreq = 0.0; float yFreq = 0.0; float zFreq = 0.0;
  // ---------- VARIABLES ----------

  // ---------- SAMPLE RAW DATA ----------
  readRawData(real_x_axis, real_y_axis, real_z_axis);
  // ---------- SAMPLE RAW DATA ----------

  // ---------- TURN RAW DATA INTO ACCELERATION (m/s/s) ----------
  processRawData(xAccel, yAccel, zAccel, real_x_axis,
                 real_y_axis, real_z_axis,
                 xZero, yZero, zZero);
  // ---------- TURN RAW DATA INTO ACCELERATION (m/s/s) ----------

  // ---------- SERIAL PRINT ACCELERATION (m/s/s) ----------
  // printAxisValues(xAccel, yAccel, zAccel);
  // ---------- SERIAL PRINT ACCELERATION (m/s/s) ----------

  // ---------- FIND MAX ACCELERATION ----------
  maxAccelX = float(findMaxAbs(xAccel));
  maxAccelY = float(findMaxAbs(yAccel));
  maxAccelZ = float(findMaxAbs(zAccel));
```

```
// ---------- FIND MAX ACCELERATION ----------

// ---------- REMOVE HIGH FREQUENCIES ----------
lowPassFilter(xAccel);
lowPassFilter(yAccel);
lowPassFilter(zAccel);
// ---------- REMOVE HIGH FREQUENCIES ----------

// ---------- COMPUTE FFT ----------
arduinoFFT xFFT(xAccel, imag_x_axis, sample_n, sampling_rate);
arduinoFFT yFFT(yAccel, imag_y_axis, sample_n, sampling_rate);
arduinoFFT zFFT(zAccel, imag_z_axis, sample_n, sampling_rate);

xFFT.DCRemoval();
yFFT.DCRemoval();
zFFT.DCRemoval();

xFFT.Windowing(window_type, FFT_dir);
yFFT.Windowing(window_type, FFT_dir);
zFFT.Windowing(window_type, FFT_dir);

xFFT.Compute(FFT_dir);
yFFT.Compute(FFT_dir);
zFFT.Compute(FFT_dir);

xFFT.ComplexToMagnitude();
yFFT.ComplexToMagnitude();
zFFT.ComplexToMagnitude();
// ---------- COMPUTE FFT ----------

// ---------- FIND HIGHEST FREQUENCY ----------
xFreq = float(xFFT.MajorPeak());
yFreq = float(yFFT.MajorPeak());
zFreq = float(zFFT.MajorPeak());
// ---------- FIND HIGHEST FREQUENCY ----------

// ---------- CONVERT NAN TO 0 ----------
checkNan(maxAccelX);
checkNan(maxAccelY);
checkNan(maxAccelZ);

checkNan(xFreq);
```

```
checkNan(yFreq);
checkNan(zFreq);
// ---------- CONVERT NAN TO 0 ----------

totalMaxAccelX += maxAccelX;
totalMaxAccelY += maxAccelY;
totalMaxAccelZ += maxAccelZ;

totalMaxFreqX += xFreq;
totalMaxFreqY += yFreq;
totalMaxFreqZ += zFreq;

computationCount++;

unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;

  float averageMaxAccelX = totalMaxAccelX / computationCount;
  float averageMaxAccelY = totalMaxAccelY / computationCount;
  float averageMaxAccelZ = totalMaxAccelZ / computationCount;

  float averageMaxFreqX = totalMaxFreqX / computationCount;
  float averageMaxFreqY = totalMaxFreqY / computationCount;
  float averageMaxFreqZ = totalMaxFreqZ / computationCount;

  Serial.print(averageMaxAccelX);
  Serial.print(" ");
  Serial.print(averageMaxAccelY);
  Serial.print(" ");
  Serial.print(averageMaxAccelZ);
  Serial.print(" ");
  Serial.print(averageMaxFreqX);
  Serial.print(" ");
  Serial.print(averageMaxFreqY);
  Serial.print(" ");
  Serial.println(averageMaxFreqZ);

  dev1_cloud_avg_max_accel_x = averageMaxAccelX;
  dev1_cloud_avg_max_accel_y = averageMaxAccelY;
  dev1_cloud_avg_max_accel_z = averageMaxAccelZ;
```

```
    dev1_cloud_avg_max_freq_x = averageMaxFreqX;
    dev1_cloud_avg_max_freq_y = averageMaxFreqY;
    dev1_cloud_avg_max_freq_z = averageMaxFreqZ;

    totalMaxAccelX = 0.0;
    totalMaxAccelY = 0.0;
    totalMaxAccelZ = 0.0;

    totalMaxFreqX = 0.0;
    totalMaxFreqY = 0.0;
    totalMaxFreqZ = 0.0;

    computationCount = 0;
  }
}
```

6.3.10. Prototype 2: Sender Thing Properties Header File

```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>


const char THING_ID[] = "09f4b4cd-c539-49df-9701-d4c915b536ae";

const char APPEUI[]   = SECRET_APP_EUI;
const char APPKEY[]   = SECRET_APP_KEY;


float dev1_cloud_avg_max_accel_x;
float dev1_cloud_avg_max_accel_y;
float dev1_cloud_avg_max_accel_z;
float dev1_cloud_avg_max_freq_x;
float dev1_cloud_avg_max_freq_y;
float dev1_cloud_avg_max_freq_z;

void initProperties(){
  ArduinoCloud.setThingId(THING_ID);
  ArduinoCloud.addProperty(dev1_cloud_avg_max_accel_x, 1,
                           READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(dev1_cloud_avg_max_accel_y, 2,
```

```
                              READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(dev1_cloud_avg_max_accel_z, 4,
                              READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(dev1_cloud_avg_max_freq_x, 5,
                              READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(dev1_cloud_avg_max_freq_y, 7,
                              READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(dev1_cloud_avg_max_freq_z, 8,
                              READ, ON_CHANGE, NULL);

}

LoRaConnectionHandler ArduinoIoTPreferredConnection(APPEUI,
      APPKEY, _lora_band::AU915, "ff000001f000ffff00020000");
```

6.4. Network Layer

6.4.1. TNN Live Data

Fig. 6.3. TNN Live Data View

6.4.2. Arduino Iot Cloud Dashboard

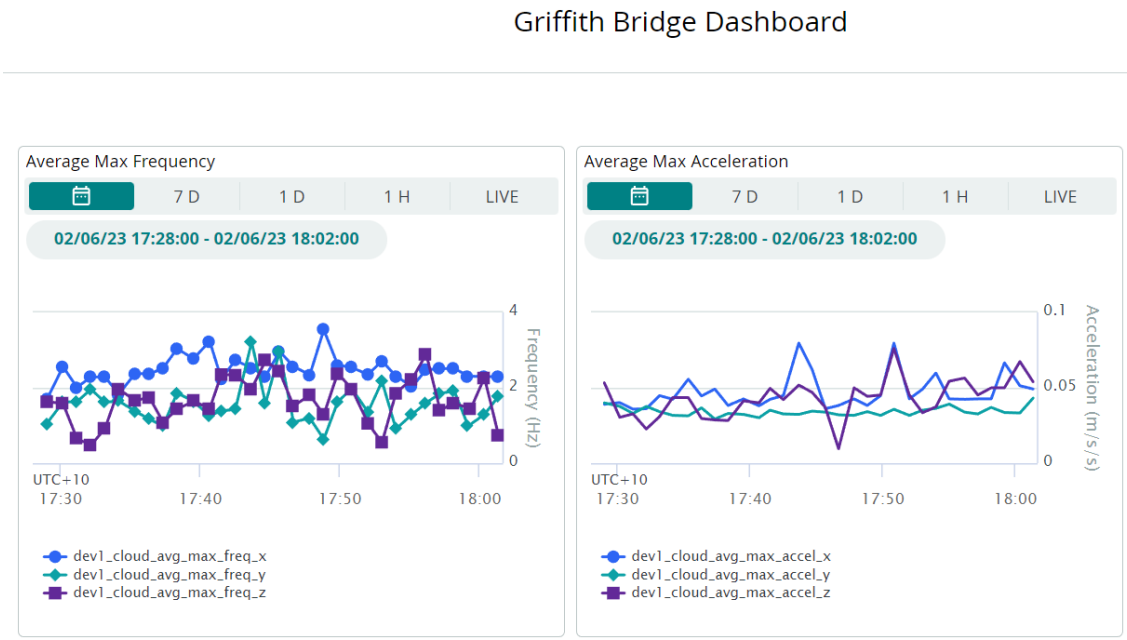


Fig. 6.4. Arduino IoT Cloud Dashboard Desktop View

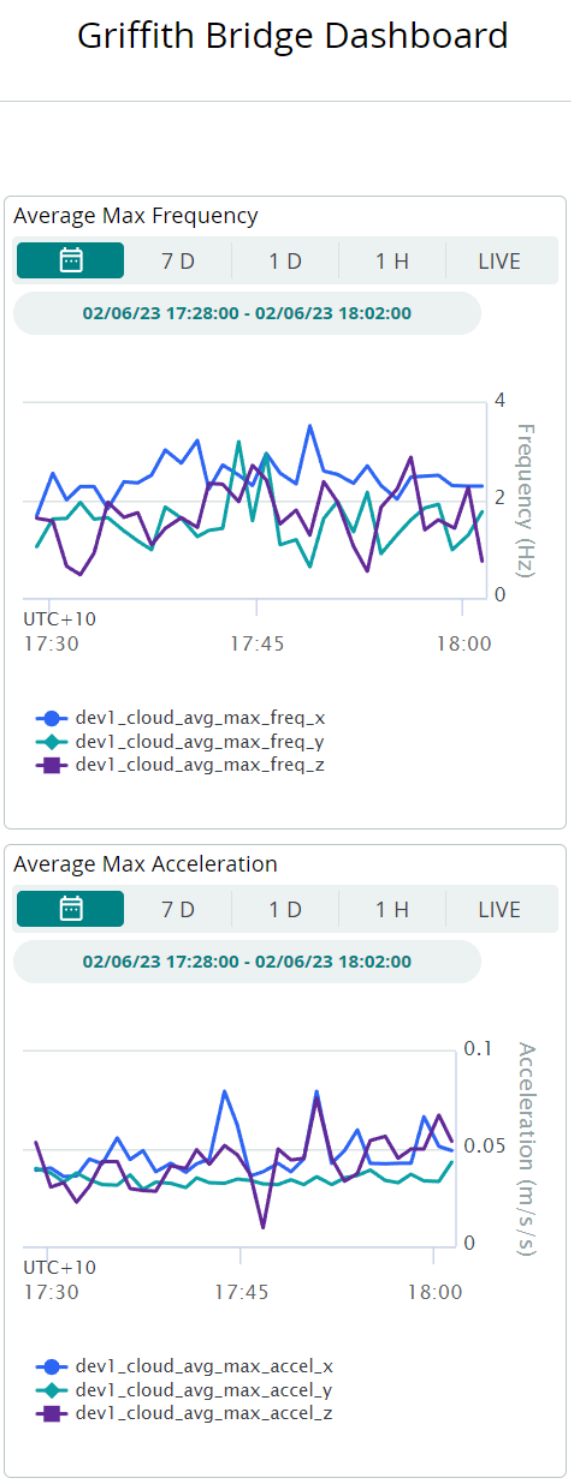


Fig. 6.5. Arduino IoT Cloud Dashboard Phone View