|  |
| --- |
| Open Source Security |
| Security System |
|  |

|  |
| --- |
|  |

Contents

[Introduction 2](#_Toc152064062)

[Background Research 2](#_Toc152064063)

[What is a distributed system? 2](#_Toc152064064)

[Mechanisms of a Distributed System 2](#_Toc152064065)

[Zookeeper 2](#_Toc152064066)

[Mesh 2](#_Toc152064067)

[Dynamo 2](#_Toc152064068)

[Communication Methods 2](#_Toc152064069)

[I2C 2](#_Toc152064070)

[Wi-Fi 2](#_Toc152064071)

[UART 2](#_Toc152064072)

[GPIO 2](#_Toc152064073)

[Project Proposal 3](#_Toc152064074)

[Control Node – Keypad 3](#_Toc152064075)

[Function Overview 3](#_Toc152064076)

[Unlocking 3](#_Toc152064077)

[Locking 5](#_Toc152064078)

[Alarm Test 5](#_Toc152064079)

[Support 6](#_Toc152064080)

[Bugs 6](#_Toc152064081)

[Testing 7](#_Toc152064082)

[Limitations 7](#_Toc152064083)

[Application 7](#_Toc152064084)

[Dashboard 8](#_Toc152064085)

[New Nodes 8](#_Toc152064086)

[Communication System 9](#_Toc152064087)

[Jewelry Box Node 9](#_Toc152064088)

[Safe Node 9](#_Toc152064089)

[Bug Testing and Critical Reflections during integrations 9](#_Toc152064090)

# Introduction

The distribueted system is comprised of four nodes. The nodes wirelessly communicate via a Mesh, with an ESP using UART to communicate to the Arduino Nano, and another ESP communicates with a computer using UART as a stationary node.

# Background Research

## What is a distributed system?

A distributed system “is one which component located at networked computers communicate and coordinates their actions only by passing messages” \*, this suggests the idea that distinct computing systems will need to communicate with one another in order to perceive what actions they need to do, and thus creating what is known as a distributed system.

Distributed systems are created from nodes, which are the fundamental building blocks of the distributed system, each node contains a constituent part of the overall system and contributes towards the goal of the system. Each node will contain primary code, which is specific to itself and then secondary code, which may also be referred to as middleware, this code is common across the entire system and is what allows the distributed system to communicate with one another.

Each node within a distributed system communicates with one another through the use of messages. Messages are segments of information that is passed between each node that instructs specific nodes or all nodes to complete a specific task, messages can also contain information which allows for the distributed system to keep all of its nodes up to date with one another.

## Formats of a Distributed System

There are many different formats in which a distributed system can occur in, in this section only two formats have been discussed, as they were the proposed formats for this project.

### Zookeeper

Zookeeper, as the name suggests, has a master node, ‘the zookeeper’, and slave nodes, ‘the animals’. The master node is in place to ensure that all the information within the distributed system is kept up to date, and that each node knows what it is doing. It has strong consistency; this is because the master node will ensure that all the nodes are constantly up to date.

The master node will control everything, but as previously stated, every slave node will have a copy of the master node, for the purpose of portioning, if it occurs. Partitioning will occur in the case of the project if the master node or any of the slave nodes go down, if it is the master node that goes down, then the slave nodes need to select a new master node, and to be able to do this each node must have a copy of everything on itself, which is again known as the middleware.

### Mesh

A mesh is a leaderless system, in which every node communicates with every other node within the system. It is highly effective as it does not require a master, therefore if any of the nodes go down then the other nodes will be able to continue as if nothing has happened. An advantage to a leaderless system, is that every node does not need to be connected to the master, and there can be chains of nodes within a mesh system, which allows for greater range of the distributed system.

Meshes fully support partitioning as well, as every node will contain middleware which allows it to dynamically communicate with other nodes in the network. Finally, there is eventual consistency between each node within the mesh distributed system, as it will take a couple of clock cycles for all of the information to be replicated across the entire network.

## Communication Methods

To be able to send information from each node within a distributed system to one another, there has to be standardized methods of communication, for the purposes of our project, this comes in the form of I2C, Wi-Fi, UART and GPIO pins. Each of the communication methods used are discussed below and the reasonings to why they have been used.

### I2C

Inter-Interconnected Communication is a communication standard, which is commonly used in electronics to communicate between peripherals and the microcontroller. It is effective as it is bidirectional, meaning that the master can receive and transmit information between slave devices. This communication protocol also only contains 2 connections, with being the System Clock and System Data connections. This method is synchronous, meaning that the data has to be times correctly otherwise it may not be read or transmitted correctly.

### Wi-Fi

Wireless Fidelity or Wi-Fi for short, is used to interconnect electronics between one another without a wired connection, it is primarily used to interface with the internet, however, can also be used to create a local network, or mesh, as seen within this project.

### UART

Universal Asynchronous Receive Transmit, is another older communication standard which is commonly used in electronics to upload code to boards. It is asynchronous meaning that there is no need for clocks to synchronize the information, instead this protocol employs buffers to contain the information until it is suitable for the microcontroller to receive it, e.g. on a clock edge. This means that information can be received at any point in time and it wont be affected by not being received on a clock edge.

### GPIO

This is the most basic form of communication method. GPIO stands for General Purpose Input Output, and is used to send and receive information, e.g., PWM, analogue and Digital On/Off signals. It is used in this project to notify the 5V logic level system of on off signals that cannot be send back through the voltage divider from the 3.3V logic level system within node 1. It is also within various nodes to receive information from sensors that do not employ any other communication protocol.

# Project Proposal

The requirement for this project was to develop a distributed system consisting of several nodes and communication methods. The purpose and methods were free to choose, and a box of basic components were provided.

The project decided on was a security system. A number of nodes with differing sensors would be designed, wired and coded, and communication between each independent node would allow for a general overview of the security of the system’s location.

The distributed system would be comprised of a control node, a safe node, a jewelry node, and a communication system. The control node would comprise of a keypad, LCD and an alarm, this could be considered the “main” node as it was the only one with a human-interface. The safe/jewelry node would determine if a safe/jewelry box had been opened/moved and would send an alert if needed. The communication system would allow information to be passed between these nodes for uses such as alarm activation and node connection/disconnection.

# Control Node – Keypad

## Function Overview

A close-up of a computer

Description automatically generatedNode 1 is one of the main interfaces between the end user and the system. The main function of this node is to provide a terminal that the user can input a code to disarm the system, so it doesn’t go off in the day when you’re just walking around the area. This node has a 16 key keypad and a display screen for inputs and visual prompts, these prompts are the unlocking function to disarm, a locking function when unlocked to rearm the system, an alarm sound test to make sure the systems alarm is working, a network function to test if it is connected to the other nodes, and a help function that shows a website and a phone number to contact for help.

### Unlocking

A computer screen with wires and a keypad

Description automatically generatedWhen the system is wanted to be unlocked the user can go to the terminal as seen in the photo below. The system will show text asking for a password to be inputted, this password can be a mix of numbers 0-9 and letters A-D, for the testing the password was ‘123A’ but the password can be any number of characters long.

A small electronic device with a blue screen

Description automatically generatedA small electronic device with a blue screen

Description automatically generatedA close up of a circuit board

Description automatically generatedIf the password is inputted correctly then a welcome screen is shown followed by the menu for further actions, shown below. These actions allow for further control and testing of the system.

A finger on a finger pressing a blue screen

Description automatically generatedA finger pressing a keypad on a small rectangular device

Description automatically generatedA finger touching a small rectangular device

Description automatically generatedOn the other hand, if the password is inputted incorrectly, the system will tell you this and tell you then there are 2 more attempts allowed. If the password is correct on any of these other attempts, the system carries on as normal. But if the password is inputted in wrong 3 times in a row, the system will go into warning mode. As seen in the image below, this means there is a countdown that gives the chance for one more attempt of the password, but if the password is wrong again, or the timer runs out, then the alarm sounds, a code is sent to the cloud to alert it something is wrong, and the keypad system locks up.

### Locking

A small green electronic device with a blue screen

Description automatically generatedOnce unlocked the user has the option to relock the system with one button press. As seen in the image below, the option on the menu shows up as ‘Lock’ press it and the system goes back to asking or a password. This function allowed the user to easily arm the space that its installed in, protecting the building and themselves. The system is built to be more problematic to unlock than lock, for security reasons.

### Alarm Test

A finger on a computer screen

Description automatically generatedIncluded in the menu is an alarm test, much like a smoke alarm, every now and then you’ll want to test if the alarm is working. It sounds the alarm for a few seconds, enough to know it works, then turns off.

### Support

A close up of a computer

Description automatically generatedThe last option on the menu list is simply a help page with the website link on it and a phone number to contact a help line for the system.

## Bugs

For Bugs for integration, see Bug Testing and Critical Reflections section.

For Node 1 there were many bugs that needed to be fixed, as show in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Node | Problem | Description | Solution |
|  | A-B Password Ending | If the password ended in either A or B, the corresponding option would be selected in the proceeding menu. | Clear the key pressed queue after unlocking. |
|  | Debounce | A single press of a key would result in ~30 inputs of the pressed key. | A delay was added after every button press to avoid multiple inputs being received. |
|  | Infinite locking loop | Locking the system would result in an infinite loop if the password ended in A. | Solved when the previous bug-fixes were introduced, as the combination of them resulted in the first menu option (Lock) always being selected. |
| Control Pad (1) | Alarm spam | Pressing enough keys would eventually stop the alarm. | The inputs would act as another attempt on password, when 3 attempts were made, the timer reset. |
|  | Fake timer (display) | During/after the lockdown timer, password attempts could still be seen on the LCD. | Introducing a full lockdown mechanism disabled the keypad, not allowing any further inputs during alarm phase. |
|  | Fake timer (function) | Entering the correct password after the alarm started would unlock the system. | After Lockdown has initiated, the only method to stop is using an external device (cloud). |
|  | Alarm crash | Introducing a second timer during the alarm phase would crash the system. | While not confirmed, it is believed either the testing board only had one timer, or the library only allowed for one timer.  Code was re-developed to only require one timer. |

## Testing

MAKE A TESTING TABLE

Vigorous testing was performed on the node to make sure all functioned as well as expected.

We tested getting the password wrong many times, to get the security alarm to go off.

We tested getting the password wrong, but getting it correct at each step-in order. Once the wrong password section was done, we tested getting the password right, then locking the system and then getting it wrong again to check that the system did work as intended and still alarmed once locked.

We tested the menu system, to make sure each option worked as intended, that the lock option locked and the help option showed the information.

We tested that the C and D keys moved the menu around.

We tested the alarm test.

## Limitations

# ~~Application~~

~~This section of the documentation describes the purpose of Application. The Application (“App”), is a fundamental part of the distributed system, providing a crucial way of monitoring and controlling the security system from a remote location. The App is split into three main sections: Dashboard, New Nodes, and Information.~~

## ~~Dashboard~~

~~A screenshot of a computer

Description automatically generated~~

~~<Figure Caption>~~

~~The above figure demonstrates the overview that is seen on the dashboard. The main component of this page is the table which displays if the node is active, the Node Name, addresses and if the Node Alarm is active. The Update button allows the consumer to effectively update the node list to check if there are any changes to the node, the list will only update automatically when the Node alarm changes from “Neutral” to “Active”.~~

## ~~New Nodes~~

~~A screenshot of a computer

Description automatically generated~~

~~<Figure Caption>~~

~~This section works by entering the details of the node, and then submitting them, to create the node, deleting them, to delete the node, or Update node, to update the node. How this works is explained in more detail in the communication system section of this documentation.~~

# Communication System

This section of the project is not easily seen when viewing the security system, it is effectively the middleware of the project, this is because it is found on all the devices but is not easily seen when looked at.

As with every node within this project, the solution did not come easy, the original proposal was to have multiple nodes communicating with the cloud and communicating with one another, via a zookeeper system. The cloud system was able to be connected to but took up a lot of the processing power and storage space on the ESP boards.

The zookeeper system would have used the Wi-Fi to also communicate with one another, however this bought on problems as well, such as not being able to know the IP addresses of each node before they connect, meaning that they did not know what the IP address of the master would be. The problem was attempted to be solved using nRF chips, which would connect to one another without the need for a known IP address, which is only dynamically created on connection, instead it was a static address, which could be hard coded in beforehand, theoretically allowing for the communication between each of the nodes. However, the nRF chips were difficult to communicate with for unknown reasons. Therefore, it was chosen to abandon the zookeeper system for the time being and instead use a mesh network, which gave a few advantages of:

* Not needing an IP address and instead using the Wi-Fi chipset’s static MAC address.
* Not needing additional hardware as the ESP’s already had Wi-Fi chips on them.
* Being a format of distributed system, meaning that the zookeeper was additional none needed work.

Finally, it was also decided to not connect the network to the cloud, as the cloud expired and would lead to security risks if it was to be used in a real-world environment. Once the project had got the wireless communication mesh working, through the use of libraries such as PainlessMesh. The next major problem was communicating between the boards, as PainlessMesh was an ESP only library at the time, it meant that the project could not use the Arduino Uno R4 WiFi as a communication point, instead the project had to employ another route, which included connecting the Arduino Uno to an ESP board. This gave problems due to the fact that they do not use the same logic level, this being the Uno is on 5V, and the ESP being on 3.3V. The initial idea was to use a Logic Level shifter, which did not work at the time, therefore it was instead decided to use a voltage divider with a 2k2 and 1k resistor. The communication, however, did not work still. This was later found to be due to using Serial.write() instead of Serial.println() and trying to use Serial.swap() on the ESP board. After changing both sides to SoftwareSerial and changing the Serial.write() to Serial.println() the boards started communicating with one another. To communicate back to the 5V Arduino from the 3.3V ESP, the project had to employ a clever route of using the digital Out pins on the ESP and reading the Digital signal through the Arduino’s Analogue pins, as they would be able to read voltages less than 5V. Once a level had been set the boards were able to communicate with one another and generate pulses to turn on and off the alarm system.

A mesh network has different topologies, depending on the arrangement in space and additional factors such as traffic patterns. In the case of the project, the topology of the mesh network will be a ‘Full mesh topology’, in this format, all of the nodes are connected to every other node, which increases stability and if one of the nodes go down, it does not affect the other nodes, this would be more likely to occur in a ‘partial mesh topology’ due to the fact that every node is not connected to every other nodes, and instead chains are created, this would occur if the network was spread out in a line and all of the nodes were not close to one another. This has the risk of cutting off entire sections of the node network, if one or more nodes that connected a section the rest of the node network went down.

# Jewelry Box Node

## Overview

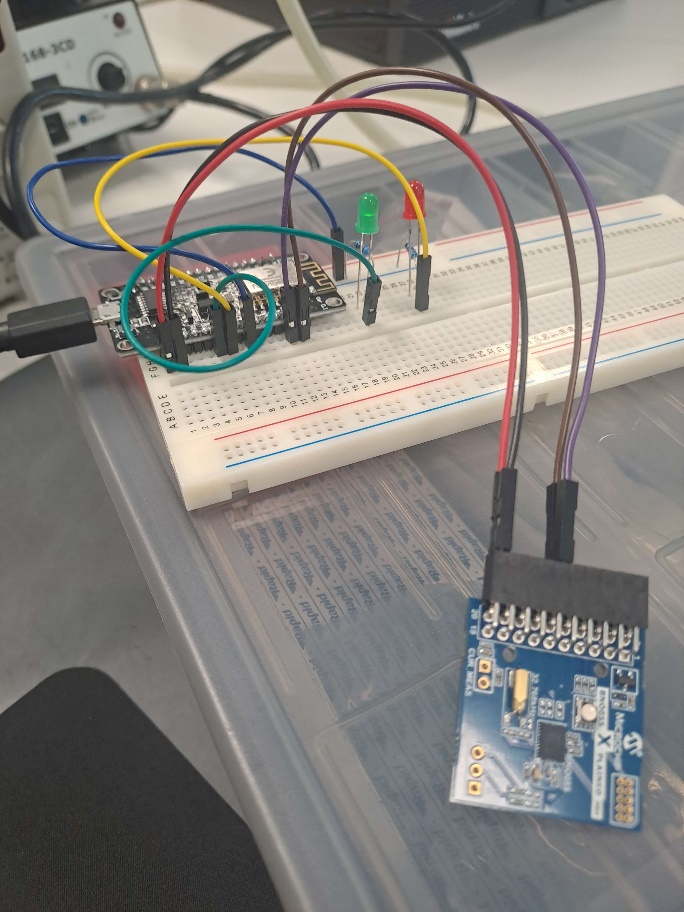
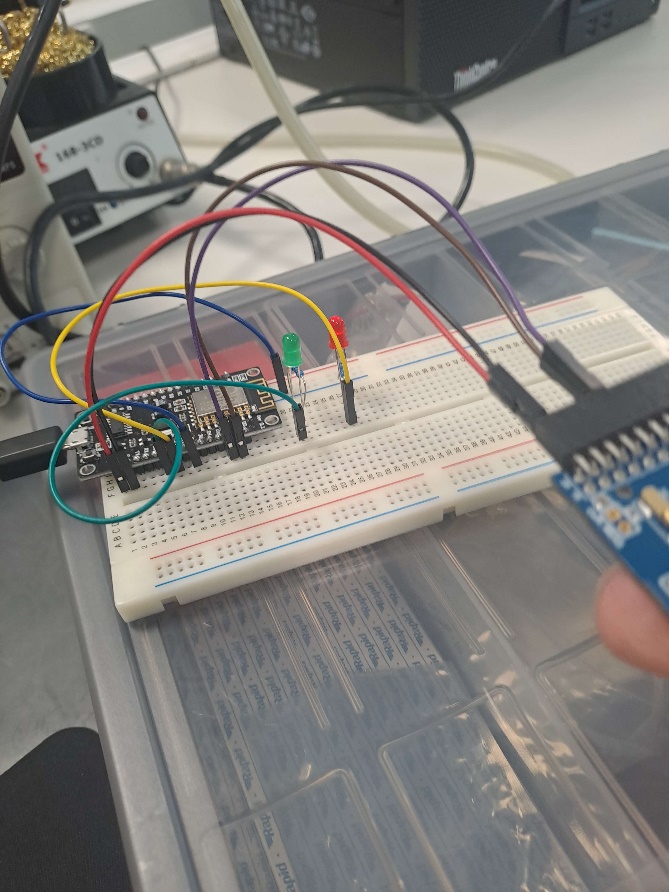
The Jewellery Box Motion Detection Module is a small device designed to be placed in a jewellery box to produce an alert if the box is moved. It works by using a BNO055 inertial measurement unit to report the orientation of the device. If the device’s orientation changes too rapidly, such as if the box were picked up, it generates an alert signal that is sent to the rest of the network, informing it of a potential theft.

The BNO055 is run using Adafruit’s sensor libraries. An I2C communication line receives the measurements from the IMU, and the values are read in the form of a Quaternion, a vector type that combines the rotation values around the X, Y, and Z axes to produce a single value representing the rotation around an arbitrary axis known as the Euler axis. By having a single value that represents all rotation, it makes it easier to compare the data against a rolling average of previous readings, allowing the device to not produce false positives from small bumps to the surface the device is on, while also making it harder to move the device without setting off the alarm.

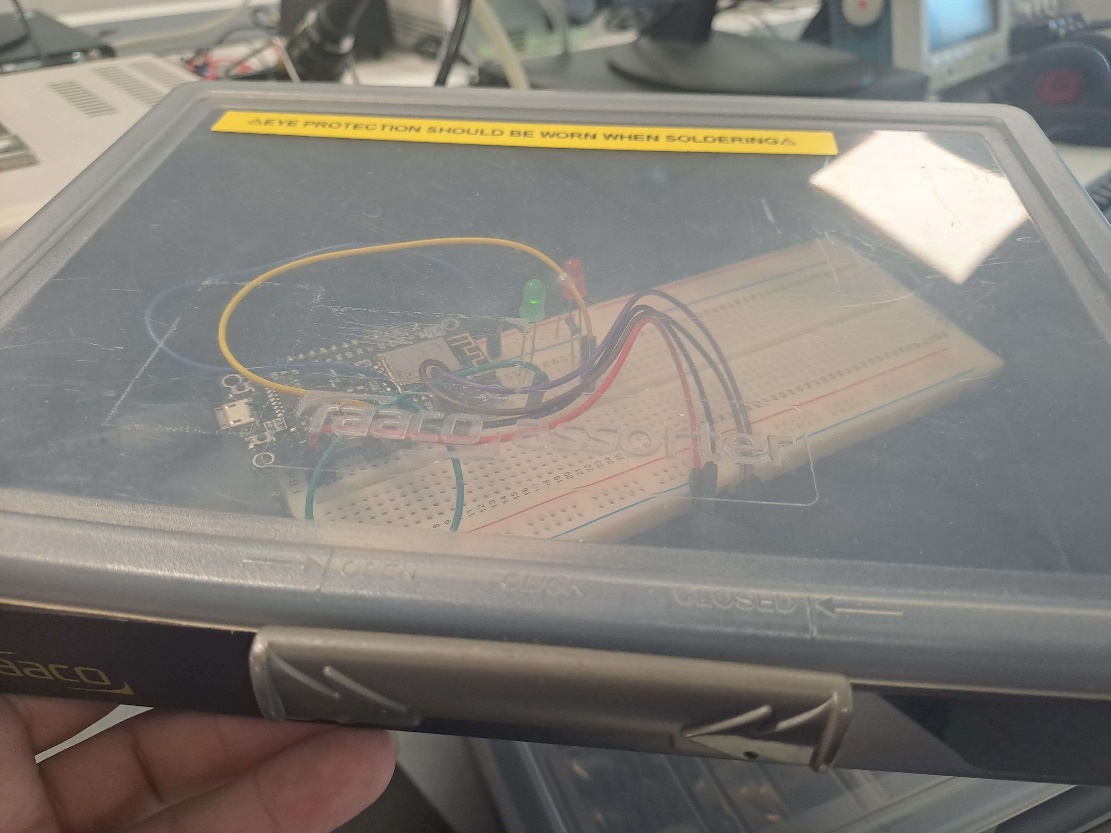
## Functions

The module only has one function: to send an alert to the rest of the network if the module is moved. It is constantly comparing its current state to a rolling average of previous readings. If the new reading varies too much from that rolling average, an alert is generated, which is then sent out. During development, visual indicators were added to be able to test the system independently.

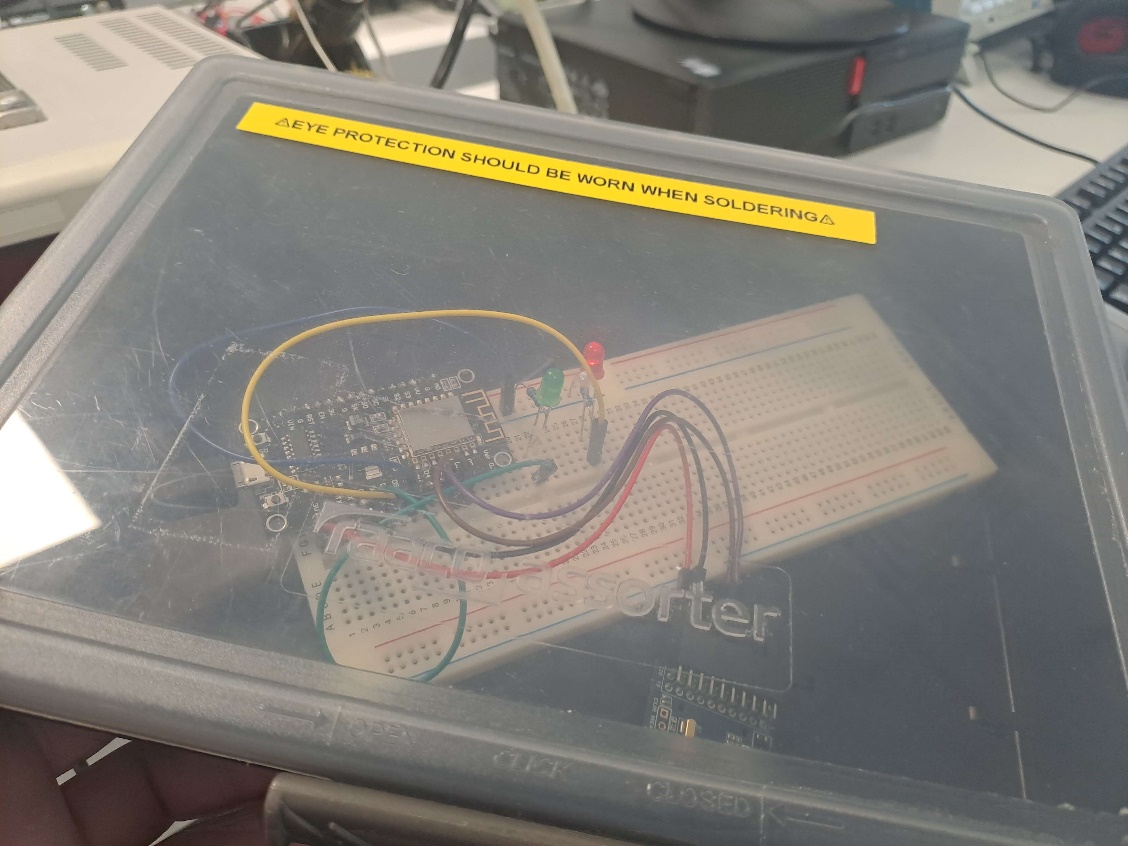
Running the function to read the IMU on a scheduled task introduced a delay in how often it was called, which had the unexpected side effect of increasing the device’s ability to tell the difference between negligible movement from bumping the table and significant movement that would justify generating an alert, as in the case of bumping the table, the device would return to its original position before the next call of the reading function.

Device while stationary Device while IMU is being moved



Device inside a stationary box



A close up of a text

Description automatically generatedDevice inside a moving box

Sample of serial terminal output while device is stationary

A white paper with black text

Description automatically generatedSample of serial terminal output while device is producing alerts

A grey rectangular object on graph paper

Description automatically generated

Model of a box to place the device into

A grey box with a blue and black rectangular object

Description automatically generated

Model with the lid removed. Black box represents the microcontroller, blue box represents the IMU, and the silver cylinder represents a rechargeable battery

## Challenges

The program required many iterations to reach its final state. First, the initial code relied on an outdated version of the sensor libraries, as it was recycled from an older project, and newer versions of the libraries had changed the formatting of some of the functions. The program also needed slight adjustments to account for the fact that it was originally written for use on Arduino boards, rather than an ESP8266.

## Bugs

Placing the device in a box and attempting to lift it without producing an alert revealed that a steady hand could move the device vertically and horizontally without setting it off, though any attempt to rotate it would immediately generate an alert. This was because the device originally only considered its angular position when deciding whether it had been moved. By changing the way the device decides whether it has been moved to incorporate the accelerometers that the BNO055 has, a new threshold was able to be established that means it would be prohibitively slow for any would-be thief to avoid setting it off.

Occasionally, on initial booting of the system, the IMU would be unresponsive, only giving values of zero on all data points. This is caused by the voltage on the VCC pin not meeting certain thresholds during power up, which causes the IMU to not properly reset for configuration. To fix this, a system was added that would wait for a number of cycles of the main loop, then check if the data values were blank. If they were, then the program would automatically reset itself. While implementing this, a toggle was added to prevent the system from generating movement alerts while it was still calibrating its position.

## Testing

Numerous tests were run, involving attempts to move the device (both inside and outside of a box) as well as bumping the table near it, to establish a suitable threshold for deciding whether changes in position were significant enough to trigger the alert. For ease of use during testing, a pair of LEDs and a serial communication port were connected to the device. By doing so, the device had visible ways of declaring an alert that did not rely on the presence of other nodes in the network.

## Limitations

As the system is very simple at its core, the main limitations are its size and how long it can remain powered if it has to run off of a battery.

The device could be improved by mounting it onto a permanent circuit board, rather than just a temporary breadboard. It also currently relies on being plugged into a usb port to receive power, which ideally should be replaced with a rechargeable battery.

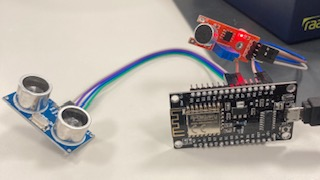
## Intergration

Integrating the systems to communicate with the rest of the network went smoother than expected. Other than the new functions needed for it, there was only two changes that needed to be made. First, the alert function needed to call the new function that sends messages to the network. Second, the function calls in the main loop had to be moved to a scheduled task, as the mesh system requires the main loop to only call the mesh’s update function. As much of the code that was previously in the main loop only served as a method to delay the device from producing alerts while the rolling average of its position calibrated, it could be blended into the setup function to remove an if statement that became unnecessary after the first second of the device running.

# Safe Node

## Overview

The purpose of this node was to make a hypothetical safe more secure by adding additional features. An Ultrasonic Sensor would be used to determine if the safe door was opened (while the system was armed), and a sound trigger would suggest if the safe was being drilled (or otherwise broken). Initially, an Arduino Uno was used to test the sensors and ensure they functioned, then the code would be modified and transferred onto an ESP8266 board to allow for easy communication with other nodes.



## Function

This node has a variable for the distance threshold for an alarm, and a noise threshold for an alarm. When the system is armed, the node will send an alert to the central node (depending which variable has been reached). The ESP8266 would wirelessly communicate with the central node via the mesh, not directly activating an alarm due to false-positives (for example, TV noise being picked up by the noise sensor) but sending a warning alert that the safe should be checked on.

ALARMED

ALARMED

SECURE



## Testing

The testing for this node was relatively simple. The serial monitor on the testing computer would be used to read the results of each sensor and output a string if the threshold had been surpassed. An object in-front of the Ultrasonic would be placed at a predetermined distance and moved away in a controlled manner with the sensor’s output compared to the actual distances. The sound sensor was tested by generating different volumed noises next to the sensor and adjusting the sensitivity potentiometer to match the threshold to the desired noise level.

## Bugs

One of the largest issues faced within the creation of this node was getting any form of accurate reading from the Ultrasonic Sensor when switching from an Arduino Nano to an ESP8266. The potential bugs explored included definitions and setup in code, missing/faulty libraries for Arduino to ESP, damaged wires, faulty pins, incorrect pin connections and incorrect logic levels. All of these proved not to be the issue, as all would result in the same reading of 0 no matter what.

Using an oscilloscope to measure the analogue voltages, the Ultrasonic was performing an output, but it was magnitudes smaller than it should have been. The issue turned out to be a safety diode integrated onto the ESP board. For a reason, it was not allowing the current at 3.3V to flow to the sensor, the solution was to bypass this diode using solder. The result of this was a fully functioning Ultrasonic Sensor on the ESP8266 with no negative repercussions.

The only other significant bug with this node was with the sound trigger. No amount of sensitivity adjustments actually allowed for significant changes in readings during testing. The reason for this was reading the sensor as an Analogue when it output in the order of mV. To solve this, the definition in the code was changed to a Digital input, and a threshold was made, as opposed to an exact Analogue value.

# Bug Testing and Critical Reflections during integrations