Flood Watch

Technical Report

Daniel Beauchamp - dcb34@kent.ac.uk
Dharius Robinson - dr296@kent.ac.uk
Natalie Mclaren - nlm21@kent.ac.uk

School of Computing
University of Kent
United Kingdom

Word Count: 4076

March 29th, 2018

Abstract

Flood Watch aims to forecast and mitigate the effects of flooding in the UK. To achieve this, it makes use of devices equipped with ultrasonic sensors to measure river levels. The system sends these measurements via a wireless network (LoRaWAN) to a custom-built API; enabling users to query the data in a meaningful way. In this report, we will explain our design process and thoughts from start to end. Aside from technical features, we will also present the project's requirements, challenges and final achievements.

1. Introduction

The UK is experiencing more floods than ever with January 2016 seeing the most severe and extreme floods in the past 100 years [1]. These incidents, amongst others, have cost the country approximately £1bn each year in damages [2].

The Oxford Flood Network is an exemplar example of how the Internet of Things (IoT) can enable new ways of monitoring the world around us; it is a community driven project, affording localised monitoring. Despite its success, what it fails to do is allow for the integration of any new devices in new areas or allow end-users to query the data.

To address these limitations we devised our own system. It aims to readily allow new devices to be added and for coverage to be increased. To achieve this, we have used a community-based network called "The Things Network". This community initiative allows users to create their own low-power and long-range network, to allow sensors to communicate with the Internet. In order to handle these varying locations, we also built an API containing a range of endpoints that can easily be queried for statistics and measurements; -grouped by time period and geo-location. The API provides a way of presenting the data in a meaningful way. Since our choice of network allows for connections to multiple devices, as long as the device is registered in the network, it is very simple to integrate them and their data.

2. Design Process

2.1 Systems Engineering

We decided to create our system following a Systems Engineering approach. "Systems engineering is the management technology that controls a total system life-cycle process. It involves and results in the definition, development, and deployment of a system that is of high quality, is trustworthy, and is cost-effective in meeting user needs" [3]. There are eight steps in the System Engineering Life Cycle process; the very first being the creation of Concept of Operations (hereafter: ConOps). ConOps serve to outline the manner in which a system will be used. The rest of the SE Life Cycle consists of; Requirements, Design, Implementation, Integration and Testing, System Verification, Operations & Maintenance and Assessment. However, we will not be focusing on the maintenance side of the cycle as it is not within our scope for a one off project, though we have kept device maintenance for the user in mind.

2.1.2 User requirements

Our ConOps included user scenarios and questions such as why the system is needed and any boundaries we can foresee. From this, we were able to identify four major user groups that might interact with our system; installers, maintainers, data analysts and anyone in the local community with an interest in monitoring rivers in their area. When assessing the different skills that each of these users would need, we decided that; installers would only need limited technical knowledge beyond installation instructions as opposed to data analysts who would need technical knowledge and skills in managing and maintaining data.

The remaining steps of the System's Engineering Life Cycle, and how we followed them through, will be explained in detail in the remaining sections.

3. Design

Our design stage was broken down into three further stages; System Design, Architecture Design and Module Design.

3.1 System Design

See Appendix 1 for a diagram of our system's required components.

Our system design as an overview of the components that we would need to develop for our system. At this stage, components were delegated between group members; Dharius was responsible for gathering information on desirable power supplies, Daniel was responsible for gathering information on communication components and Natalie was responsible for gathering information on desirable storage components and on cases which would encase our device for deployment. All group members were responsible for gathering information on a processor and measurement device.

3.2 Architecture Design

Communication

See Appendix 2 for an illustration of the advantages and disadvantages between the different types of communication that we considered.

Our initial thoughts for network access was to use 3G for sending water level readings. However, after carrying out some signal tests in Canterbury, we found that we were unable to get good signal in the majority of the areas that we wanted to deploy the sensors. We decided against WiFi due to the unfeasible nature of getting access points setup. After looking into what other similar projects have used, and ways to transmit data, we found LoRaWAN and The Things Network. We chose The Things Network after testing out the local connectivity in the area and found courage along the river. Another benefit of LoRaWAN, is that the network infrastructure can be added too, to allow greater coverage or reliability.

Measurement Device

See Appendix 3 for an illustration of the advantages and disadvantages between the different types of measurement devices that we considered.

Due to limitations of other solutions, such as making physical contact with the water over long periods of time and varying weather conditions, we chose to use an ultrasonic sensor to measure river levels. Our device also detects only the largest encountered objects up to a maximum 5-metre range, which is useful to avoid unexpected objects and animals floating in the river. We also focused on choosing components that prioritise functionality and reliability over cost. Therefore, even though ultrasonic sensors were some of the most expensive, they also offered the most benefits including ease of setup and little to no maintenance, which is very important for our intended users to create a network of devices for use over a long period.

Processor

Initially, we planned to use an ATmega328P microcontroller on a custom designed PCB. Our development board, the Adafruit Feather M0 Adalogger (ATSAMD microcontroller), however, fulfilled all of our needs. We decided to keep this as our processor and integrate the entire board onto our custom PCB to extend its functionality. The reason that the Adalogger's processor worked so well for our system, is that it provides a balance between being powerful for a small IoT device, and guaranteeing low consumption.

Power supply

See Appendix 4 for an illustration of the advantages and disadvantages between the different types of power supply devices that we considered.

Originally we had decided on using AA batteries due their low cost and wide availability when in need of a quick battery replacement. However, lacking the ability to recharge meant that over long periods of time we would accumulate a lot of waste from battery usage. Due to this, we eventually settled on using a 3.7v Lithium Polymer rechargeable battery as our power supply, due to Li-Po batteries' robustness and stability. We needed a battery strong enough to withstand all weather conditions outdoors - Li-Po batteries could ensure this as they generally have reasonable operating temperatures and can last for long periods of time without losing much charge or suffer from aging.

Including this, we also looked at the possibility of using solar or wind power to keep our device charged to reduce the need for maintenance. However, we ultimately decided against it, as there would be considerable wear and tear to wind turbines, and would require large and expensive panels for solar to work effectively for each device in our often cloudy climate (in the UK).

Storage

See Appendix 5 for an illustration of the advantages and disadvantages between different types of storage devices that we considered.

An SD card seemed best over EEPROM due to providing a lower limit on write cycles, unlike the latter, and a larger amount of storage space. Additionally, it proved to be optimal at a later stage when working with our processor which included an SD Card slot.

Case

See Appendix 6 which shows a bird's eye view of our modelled 3D case design, with exact PCB measurements.

A showerproof watertight case was crucial for our device to remain protected outdoors. Initially, we wanted to experiment with 3D printing therefore, we modelled a design of our case. Our supervisor then made us aware that "off-the-shelf" cases were readily available to use and are certified for harsh environments. This helped save time working on creating a good quality case that does not really further our progress on the device. Therefore, the choice to use one that already has these quality guarantees, suited our needs and time constraints best.

3.3 Module Design

Now knowing what components we will be working with; this stage served as a low-level design in which we specified the functionalities of each with pseudo code. We designed functions for; the reading of river levels using the sensor, communication setup and data transmission using LoRaWAN and for saving data into an SD Card. As time went on, some of our functions changed, especially after working with our sensor and learning that it would automatically calculate distances between objects. Initially, as we were unaware that the sensor would be able to provide us with such functionality, we wrote a function that measured the time between sending an ultrasonic soundwave and receiving it back after hitting the top of the river. We then wrote another function that would take this time and convert it into distance (millimetres), which served as the distance from the sensor to the water.

4. Implementation

4.1 Aim

For this project, our main focus was gathering data and organising it for use by other professionals. Therefore, we thought that building an API that would provide a resource-based interface for our readings and devices would be a crucial part of achieving this.

See appendix 7 for a diagram on the flow of our system.

4.2 Device software

We implemented our device software in C++, due to popularity amongst embedded programming projects for its powerful efficiency possibilities in memory and processing. Additionally, Daniel had some experience with it and Dharius and Natalie had recently started to learn the language. Originally in one sketch file, we separated our source code into separate class and header files to maintain organisation and interact with objects. Our software consisted of five classes; Engineering Menu, Sensor, SD Card, Processor and LoRaWAN.

Whilst implementing our device software, we encountered some issues such as insufficient memory on our processor due to global and string variables. In order to fix this, we decided to store the string variables in the processor's flash memory instead, therefore changing their type to "PROGMEM" instead.

Initially when unplugging our device from a laptop without a battery connected to keep it powered on, all set values were lost - i.e. last measurement sent and delay period value. To change this, we decided to store these default values as flash variables in the device's EEPROM instead and then applied a check on the variable flag. If the flag is true, then the variable has been set in a previous setup Therefore, there is no need to take the user through the initial setup process again. If the variable flag is false, it is assumed that the device has been restarted or has not yet gone through a previous setup. Therefore, the user is then taken through the setup process.

4.2.1 Engineering Menu

For debugging purposes, we implemented an engineering menu with options to test component functionality or gain information such as the current battery percentage. Our menu also allowed for a customised configuration of the device with options allowing the user to change various settings, including; measurement period (delay between sampling), the Accelerated Readings Mode threshold and the LoRaWAN spread factor.

4.2.2 LoRaWAN and The Things Network library

To quickly and efficiently connect LoRaWan, we ended up using The Things Network Arduino library. It gave us a great starting point with error handling, but it ended up causing issues when trying to expand features, such as trying all spread factors to prevent the manual changing of spread factor. To resolve this we had to fork The Things Network library from GitHub to gain access to the protected and private functions needed.

4.3 The Things Network & Node.js client

The Things Network processing

The Things Network processing occurs in three steps: the Decoder, where the bytes from the uplink are transformed into a JavaScript array; the Converter, where any of the raw data that was decoded is processed into better information - we use this to add a display name for the uplink type - and finally, the Validator which ensures that the uplink is valid - here, we test that the type exists. If the tests fail, then The Things Network will discard the uplink.

Whitelisting

Using The Things Network's Node library, we connect to the Flood Watch application and get a list of all the setup devices. We then put the device IDs into an array which we check against when receiving incoming uplinks. If the uplink's device ID isn't in the whitelist, we discard it.

4.4 Databases

Once having received an uplink from The Things Network, our Node.js client then stores the data points into our database. If the schema to store the data points is not set up, then the schema is created on the first run.

Since the data that we gather is heavily periodic, timestamping each measurement is of great importance. Therefore, we chose to use a time series database called InfluxDB to store the incoming data. This includes device ID and battery power and the measurement with its timestamp. Depending on the type of data, it will get stored in either the 'Reading', 'Still Here' or 'Error' Measurement (known as a 'Table' in relational databases).

Furthermore, the relational structure of a MySQL database is used to allow us to have IDs for devices and apply them as tags on the time series data points. It also allows us to provide a relation of many devices to one area to provide more structure by grouping the devices. The areas can then have a parent area to provide endless recursive grouping.

Redis ‘change since previous’ processing

To calculate changes between a device’s latest and previous reading, we used Redis to cache the latest reading for each device. Redis is an in-memory data structure store used as a database and cache.

4.5 API

To build the API, we chose to use the Laravel PHP framework. This is due to two of our group members having a lot of prior experience in using it, with one being in an API context. This allowed us to promptly setup and framework with the required functionality as quickly as possible.

There are six main models in our API; ‘Device’, ‘Area’, ‘Stat’, ‘Reading’, ‘StillHere’ and ‘Error’, each having its own endpoint. ‘Device’ and ‘Area’ were given sub resource endpoints to combine with ‘Stat’, ‘Reading’, ‘StillHere’ and ‘Error’ data. To convert the model’s raw data array into the standard JSON response format (known as JSONAPI), we used PHP package ‘PHPLeague Transformers’. JSONAPI is widely used for JSON based REST APIs.

4.6 Scalability

Our devices send readings independently, which then get appended with its unique device ID when The Things Network encounters the uplinks and passes them on to our Node client. The unique device IDs received match the device IDs in The Things Network console. This is useful in maintaining consistency across components and quickly differentiating each device in our database. This means that there is room for expanding the data gathered in multiple areas, by integrating more sensor devices to the network. In doing so, we ensure that our system is scalable by offering the possibility of building a widespread picture of river levels across the country.

Furthermore, we use Redis to cache the last measurement received by The Things Network and the uplink receiver, in order to ensure that our system is capable of some performance scaling as more devices are added to the network.

5. Testing & Results

Our testing approach comprised of three software testing techniques; Unit Testing, Integration Testing and System Testing.

5.1 Unit Testing

Unit Testing involves testing the individual units (functions) of our software.

During this process, we noticed that when setting a large custom value for our delay period (in milliseconds), it was not being set correctly. We found that having this variable as type ‘int16_t’ was causing an integer overflow as our delay period value was larger than what a 16 bit integer could store. We fixed this by changing it to a 32 bit integer type instead.

Similarly, we changed the SD card’s text file size variable to the same type since we were explicitly setting it to a large value to test the ‘fileHasReachedSizeLimit’ function. Unfortunately this test kept failing, however, due to unknown reasons. See appendix 9 for the results of this test.

Furthermore, when testing the function responsible for returning the current measurement, we noticed at times, it would return a negative value, indicating that the river level is below the river bed, which is not possible. To fix this, we set the value to 0 if it was ever negative. However, it would be best in future to log and send this as an error, to make the user is aware that the device has been configured incorrectly.

5.2 Integration Testing

Integration Testing ensures that that all units and components work well together.

When testing our engineering menu, we realised that there was no indication that an invalid option was entered. We therefore implemented a check on the engineer’s input to ensure that it was actually valid .Otherwise, the user would be prompted with an “Invalid option” message. Other improvements also included an asterisk box around the “Engineering menu” title to separate it clearly from other potential text in the Serial screen.

When testing that the sensor was returning expected results, we carried out some tests against river levels in Canterbury, and then against moving objects indoors. When setting an initial river depth of 0, the sensor was expected to return decreasing values as objects moved closer towards it - indicating a higher hypothetical river level. When moving away from it, increasing values were expected as this would indicate that the river level was decreasing. In addition to this we performed another test to ensure that the full range capability performed accurately, as expected. This was done by attaching the sensor to a metal arm and moving it closer/further away from the ground. The closer the object or the floor was to the ground, the higher the hypothetical river levels as per our logic. Our tests proved to be successful, as they met our exact expectations.

Finally when testing LoRaWAN and The Things Network, we noticed that it was able to join the network. However, the data would not be sent correctly. To fix this, we added functionality to dynamically set a spread factor ranging between 7 and 12. This was done by looping through channels from 7 to 12 until a reliable connection is established. The higher the spread factor, the better connection to far away and obstructed locations. However, having a higher spread factor also increases our upload time to The Things Network - therefore, further limiting the maximum number of daily message uploads we can send. This functionality acted as a safety measure to ensure that the spread factor set is suitable for the device's range.

5.3 System Testing

System Testing involves testing our system as a whole given certain scenarios from start to finish.

Our system tests consisted of going through scenarios involving all of the major components of our system, to ensure that it performed as originally intended, meeting our user's requirements. We tried to test scenarios that covered the main functionalities that we focused on for the device. For example; Accelerated Readings Mode being triggered, ignoring measurements below the ignore threshold or that are not worth sending (measurements which are almost the same as the previous one) and the device losing power and reconfiguring itself.

As well as this, we performed integration tests for a single scenario that eventually spans the entire system - from our sensor taking a measurement, to it being displayed in our API front-end. Starting with device setup, we ensured that our communication component was able to connect to The Things Network. From there, we initialised the initial river depth, and assembled the device in its case. At this stage we noticed that it took some time for the device to be assembled. Consequently, we implemented a 15-minute delay, to give the engineer enough time to screw the device's case in place and mount it in the desired area before it is initialised based on its current height. Next, we ensured that the sensor was taking readings by checking The Things Network console, which also confirmed that LoRaWAN was successfully sending the data. We then tested our uplink receiver, which should have notified us that an uplink was received and saved into our database. The next step was sending a GET request to an endpoint, to ensure that new data was fetched from the database.

6. Conclusion

Over the course of this project, we have created a device and backend that gathers river levels data, which can then be aggregated together for flood monitoring applications. Additionally, further devices can be added to the network, which then increases the usefulness and relevance of the data.

6.1 What we achieved

Our device uses ultrasonic sensors that can provide data to the community about their local river areas.

Our API utilises our chosen web framework and transformers to take data points from InfluxDB, as well as device and area models from MySQL. Furthermore, it morphs them into RESTful JSONAPI responses. The system provides aggregated statistics based on the data points, giving a starting point to any application or service wanting to make use of our API.

One of our project's aims was to provide scalability, leaving room for expansion in the future. To achieve this, we used The Things Network which would ensure that as there is a gateway in range, a device can be placed and added to The Things Network, and the user's own database. The data can then be organised in our API, and be used to create custom interfaces and applications for end users to view.

Overall, we are happy with the results produced and the implemented functionalities.

6.2 What would we change

Throughout working on the project there were numerous choices that we had to make to better suit our requirements and scope. If we had the chance to work on it further, these would be the aspects we would improve on.

Regarding LoRaWAN, we were initially directly sending commands to the RN2483 module. However, in order to get a rapid initial start on the project, we switched to using the Arduino library provided by The Things Network. If we could change one thing surrounding LoRaWAN, we would instead build its interaction from the ground up, with our own code to interact with the RN module. This is due to the implications of not being able to interact with the code responsible for communicating with the module without forking the original repository as it was protected and privatised.

We would have liked to provide users with more default front-end templates aside from just the graphs. For example; heatmaps. We feel that this would have helped in advertising the power of the data that we gather and how it can be used. We would also consider building a mobile application to reach users anywhere on the move.

As the project is focused on community interest, we would have liked to have created a lower cost device that makes use of a pulley system instead of an ultrasonic sensor. This would still provide the exact same functionality but at a lower price. The aim would be to encourage more local community involvement.

Finally, we were unable to add all of the functions that we outlined in our engineering menu design document. Because of this, some option numbers were skipped. As such, a final feature that we would like to add would be the implementation of these functions and their addition to our engineering menu.

Bibliography

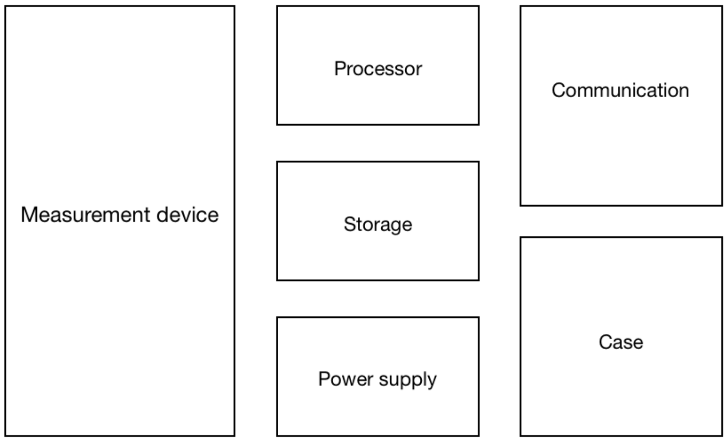
[1] Earth, P., stories, P. and floods?, I. (2018). *NERC - Is climate change causing more UK floods?*. [online] Nerc.ac.uk. Available at: http://www.nerc.ac.uk/planetearth/stories/1849/.

[2] Earth, P., stories, P. and floods?, I. (2018). *NERC - Is climate change causing more UK floods?*. [online] Nerc.ac.uk. Available at: http://www.nerc.ac.uk/planetearth/stories/1849/.

[3] Opac.vimaru.edu.vn. (2018). [online] Available at: http://opac.vimaru.edu.vn/edata/EBook/Handbook.pdf. Pp 3.

Appendices

Appendix 1 - System Design showing all required components.



Appendix 2 - Advantages and disadvantages of different communication types.

Type	**Advantages**	**Disadvantages**
3G/4G	Gives network access in more remote areas.	Can't Add to the network, if there's no signal the only chance is changing Provider. 3G/4G connections are typically charged. Signal can be lacking in cities. Our Testing in Canterbury showed many areas where 3G was lacking
WiFi	Provides high speeds. Normally set up in cities.	WiFi Access points only provide a short range and require being connected to the Internet. Can't easily gain access to existing WiFi Access Points.
LoRaWAN	Built for low powered devices to transmit data. Free to use with some limits set on data. Can add to the Network by purchasing gateways.	Gateways can be more expensive than a WiFi Access Point and also requires being connected to the Internet. Have to maintain any Gateways setup.

Appendix 3 - Advantages and disadvantages of measurement device types.

Type	Advantages	Disadvantages
Ultrasonic	Very accurate. Easy to set up in multiple environments. No physical contact required. Measurement is based on a cone shaped area to help avoid small objects interference.	Expensive.
Pulley system	Cheap.	Complex moving parts that need to be maintained. Physical contact means damage could be caused in freezing weather (e.g. water freezes). Possible danger to animals.
Lasers	No physical contact required.	Expensive. Reflective objects in river can possibly causes anomalous values. Very pinpoint measurements.
Water pressure sensor	Mid-priced.	Physical contact (Freezing water can cause damage to device).

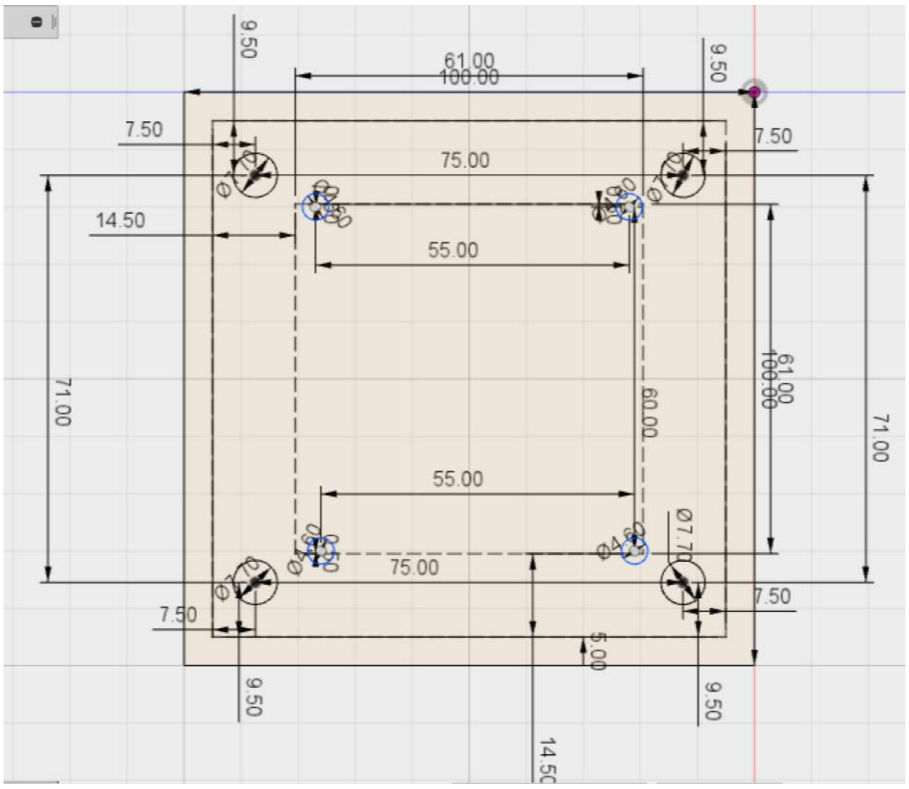
Appendix 4 - Advantages and disadvantages of different power supply types.

Type	Advantages	Disadvantages
Lithium-Ion Polymer	Rechargeable. Compact. Robust and stable. Holds charge well. (long periods of time) Performs well in various temperature conditions	Expensive
Lithium-Ion	Rechargeable. Cheaper. Performs well in various temperature conditions	Suffer from aging
AA batteries	Widely available. Very cheap. Performs well in various temperature conditions	Single use

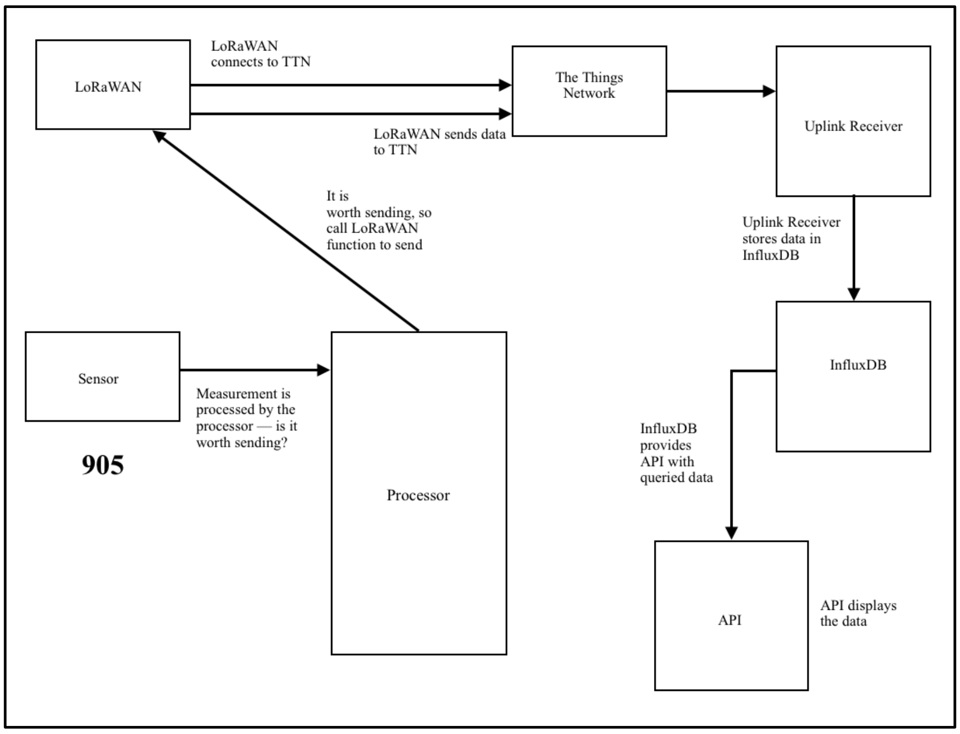
Appendix 5 - Advantages and disadvantages of different storage types.

Type	**Advantages**	**Disadvantages**
EEPROM	Easy to write to and erase from without needing to remove it from the board.	Low data retention and write cycles.
SD Card	Storage - larger storage space provided (some cards allow up to 1TB). Portable - can be ejected from the board and inserted into other devices. Easily replaceable. Larger write cycle allowance.	Unsecure - there is a risk of unauthorised individuals having access to the log. More prone to being corrupted.

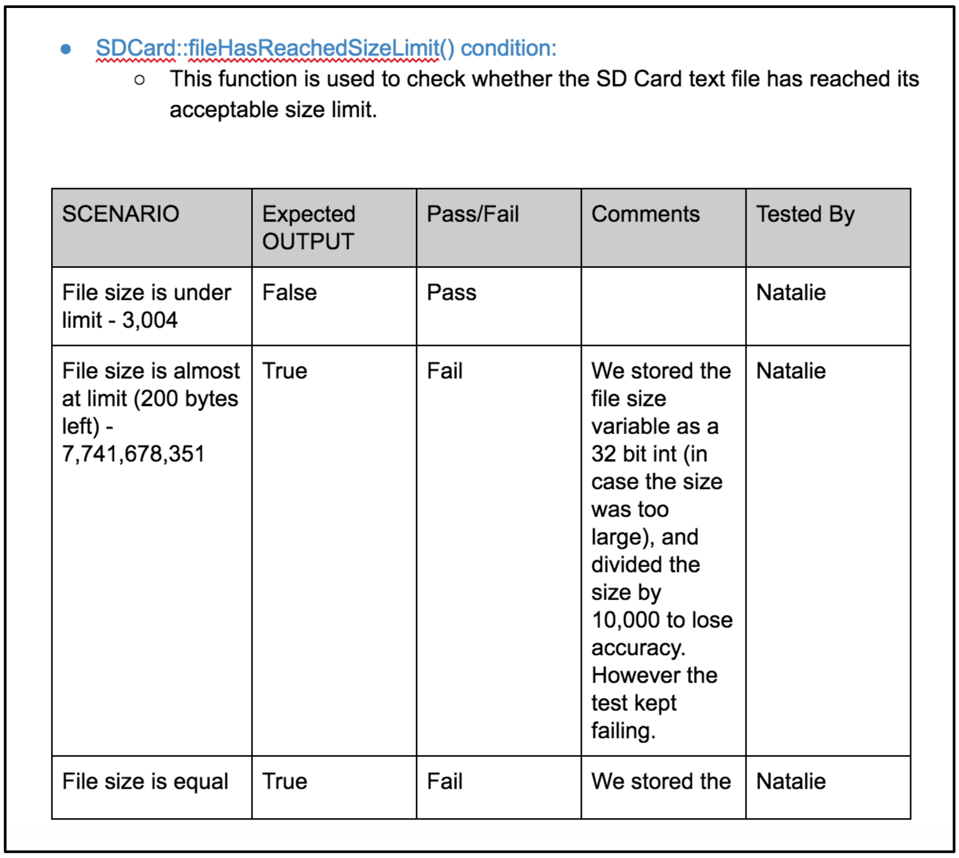
Appendix 6 - Top view of initial case design (to be 3D printed).



Appendix 7 - System flow.


LoRaWAN
LoRaWAN connects to TTN
The Things Network
LoRaWAN sends data to TTN
Uplink Receiver
It is worth sending, so call LoRaWAN function to send
Uplink Receiver stores data in InfluxDB
InfluxDB
Sensor
Measurement is processed by the processor — is it worth sending?
905
Processor
InfluxDB provides API with queried data
API
API displays the data

Appendix 8 - Results of the SD card's fileHasReachedSizeLimit function test.



- SDCard::fileHasReachedSizeLimit() condition:
  - This function is used to check whether the SD Card text file has reached its acceptable size limit.

SCENARIO	Expected OUTPUT	Pass/Fail	Comments	Tested By
File size is under limit - 3,004	False	Pass		Natalie
File size is almost at limit (200 bytes left) - 7,741,678,351	True	Fail	We stored the file size variable as a 32 bit int (in case the size was too large), and divided the size by 10,000 to lose accuracy. However the test kept failing.	Natalie
File size is equal	True	Fail	We stored the	Natalie