Air Quality Mapping

Final Report for CS39440 Major Project

Author: Robert David Mouncer (rdm10@aber.ac.uk)
Supervisor: Dr. Neal Snooke (nns@aber.ac.uk)

24th April 2018

Version 1.0 (Release)

This report is submitted as partial fulfilment of a Meng degree in Software Engineering (with integrated year in industry) (G601)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

Name Robert David Mouncer

Date 27/04/2018

Acknowledgements

I am grateful to Dr Neal Snooke, my project supervisor, for overseeing the project and guiding me in the right direction when help was needed.

I am grateful to Information Services and Computer Science Support for the hosting of data and webpages for this project to be a success.

I'd like to thank Riversimple, my industrial placement of 2016 to 2017, for the idea of the air quality mapping project and allowing me to pursue it.

Abstract

Air pollution has had a large impact on the world from hazardous gases effecting the atmosphere to the death of millions of people each year. This has become an increasingly concerning problem in recent years as the effects have been researched and have been linked to causing damage to multiple targets. 40,000 deaths within the UK each year have been associated with air quality levels [1] with a large proportion of the deaths happening in major cities such as London. January-March 2017 it was estimated that nearly 40 million vehicles are on Great British roads [2]. This has a large impact on the air pollution levels within the UK, but these vehicles may help provide a solution to this problem.

In 2016 the Royal College of Physicians released the 14 steps they felt were needed in order to improve air pollution levels, one of these was to "monitor air pollution effectively" and to "educate the public" [1]. The purpose of this project is to provide a means of educating the public of the air quality on public roads. The project will contain two systems, a monitoring system that will be used within vehicles to collect air quality data while travelling, and a visualisation system to be used to educate the public in a proactive way. The monitoring system has been designed with the idea of the system being implemented on a small percentage on public roads to help build a dataset.

This report will explain the procedure taken to complete these two systems and the process leading up to their success.

Overall the project was a success with two working systems, though the overall functionality is limited the main goals of this project have been completed. This project provides a proof of concept to the idea originally by Riversimple.

Contents

1. BACKGROUND, ANALYSIS & PROCESS **7**
1.1. BACKGROUND 7
1.2. ANALYSIS 12
1.3. PROCESS 14
2. DESIGN **16**
2.1. OVERALL ARCHITECTURE 16
2.2. HARDWARE DESIGN 17
2.3. MONITORING SOFTWARE DESIGN 20
2.3.1. Data Logging (Data_logger.py) Design 21
2.3.2. Upload (Upload.py) Design 22
2.4. VISUALISATION SOFTWARE DESIGN 23
2.5. VISUALISATION USER INTERFACE DESIGN 24
3. IMPLEMENTATION **26**
3.1. HARDWARE IMPLEMENTATION 26
3.2. SETTING UP THE ENVIRONMENT FOR THE RASPBERRY PI 26
3.2.1. Installing Linux and connecting to the RPI 26
3.2.2. Setting up the Samba server 26
3.3. MONITORING SYSTEM – DATA LOGGER 27
3.3.1. Retrieving serial readings 27
3.3.2. Format of GPS data 27
3.3.3. Starting a logging method 27
3.3.4. Implementing the I2C interface 28
3.3.5. Logging method 31
3.3.6. Creating cron jobs for the data logger 31
3.3.7. Analysis of the data logger 31
3.4. MONITORING SYSTEM – UPLOADING DATA 32
3.4.1. Connecting to the MySQL server 32
3.4.2. Executing statements on the MySQL server 32
3.4.3. Analysis of the uploading script 33
3.4.4. Creating Cron jobs for the upload function 33
3.5. VISUALISATION 34
3.5.1. Finding a suitable plugin 34
3.5.2. Implementing Leaflet.heat 34
3.5.3. Retrieving data from the MySQL server 34
3.5.4. Selecting data to view 35
3.5.5. Viewing the data 35
3.5.6. Interactive tools 37
3.5.7. GUI for webpage 38
3.5.8. Problems found with Leaflet.heat 38
3.5.9. Analysis of Leaflet.heat 39
4. TESTING **40**
4.1. OVERALL APPROACH TO TESTING 40
4.2. UNIT TESTS 41
4.2.1. Unit Tests – Data Logger 41
4.2.2. Unit Tests – Upload Script 42
4.3. MANUAL TESTING 43
4.3.1. RPI Security Testing 43
4.3.2. Data logger 43
4.3.3. Upload script 44
4.3.4. Visualisation Application Testing 44

4.3.5. User Interface Testing 45
4.4. KNOWN BUGS 47
4.4.1. Data logger 47
4.4.2. Visualisation Index.html page 47

5. CRITICAL EVALUATION 48
5.1. REQUIREMENTS AND DESIGN 48
5.2. PROCESS 48
5.3. STARTING THE PROJECT AGAIN 48
5.4. FUTURE WORK 48
5.5. OVERALL 49

6. APPENDICES 50
A. THIRD-PARTY CODE AND LIBRARIES 52
B. ETHICS SUBMISSION 54
C. CODE SAMPLES 56

ANNOTATED BIBLIOGRAPHY 59

1. Background, Analysis & Process

1.1. Background

During an industrial placement at Riversimple from September 2016 to September 2017 a telemetry unit was being built to collect data from a hydrogen vehicle to monitor how customers were driving and report any fault with the vehicle. An employee at Riversimple mentioned an idea of being able to collect air quality data with the telemetry unit and use that data to show the public a comparison of air qualities around the country. With permission from Riversimple to pursue this project, the aim was to create a method of monitoring and visualising air quality data inspired by the telemetry unit.

During the industrial placement it became clear of the state that the air pollution in the country was in due to personal and public transport. The motivation of this project stems from this and has developed over time. With little experience of such systems, the challenge was part of the motivation to complete this project and ensure its success.

Background Preparation

There was little background preparation for this project until the project had been reviewed and accepted as a major project. Preparation began between the project being accepted and the start date of the major project.

The telemetry unit was built using a Raspberry Pi Model 3B+ (RPI). With little experience of using a RPI, preparation was carried out to ensure general knowledge of the microcontroller was known before work was to be carried out.

The operating system (OS) for the RPI was a vital part of the preparation, it needed to be decided what OS was to be used to suit the task at hand. Most RPI operating systems run using a variation of Linux. The time leading up to the project hand out, comparisons were made between the OS's.

Supported protocols for the RPI were investigated, it became clear that most protocols could be used but had to be connected to the dedicated pin on the RPI board. If a protocol was not supported locally, a RPI HAT could be used to allow for support. A RPI HAT is an add-on board for the RPI, it stands for hardware attached on-top.

Hardware components needed to be investigated to ensure that it was possible to achieve what had been set out. Using the microcontroller - location and air quality data needed to be collected. It was previously known that a GPS can be used to establish their current location, however there was little knowledge about the use of air quality sensors. Air quality sensors were found online but most of them needed additional components or were not suitable for the project. The accurate sensors and multi-gas sensors were very expensive and due to the extent of the project were not suitable. Several sensors were appropriate for the project. From this I knew I could create the hardware needed for the project, and even if they were not very accurate or reliable, it would still prove as a proof of concept.

Having previously used a Google Maps API for commercial purposes, it was known that the licence agreement was not very permissive. With that knowledge, research on other online map providers was conducted; OpenStreetMap was very permissive as it only required recognition on the webpage using it.

Research was conducted on similar products to understand what solutions had been created and what to avoid for copyright purposes.

A blog was created to record what had been learnt and to keep track of any work that had been completed. This blog was hosted on the Aberystwyth University public_html directory. Access was given to the supervisor, in order to check on the progress of the work.

A GitHub account had already been created for personal use and this was used for the major project, the repository was set to private.

Similar Systems

When researching the monitoring hardware and implementation it was discovered that a team was measuring air pollution within London using pigeons [3], the pigeons would wear small backpacks with air quality sensors and a GPS. The type of hardware is the same, but the deployment is different.

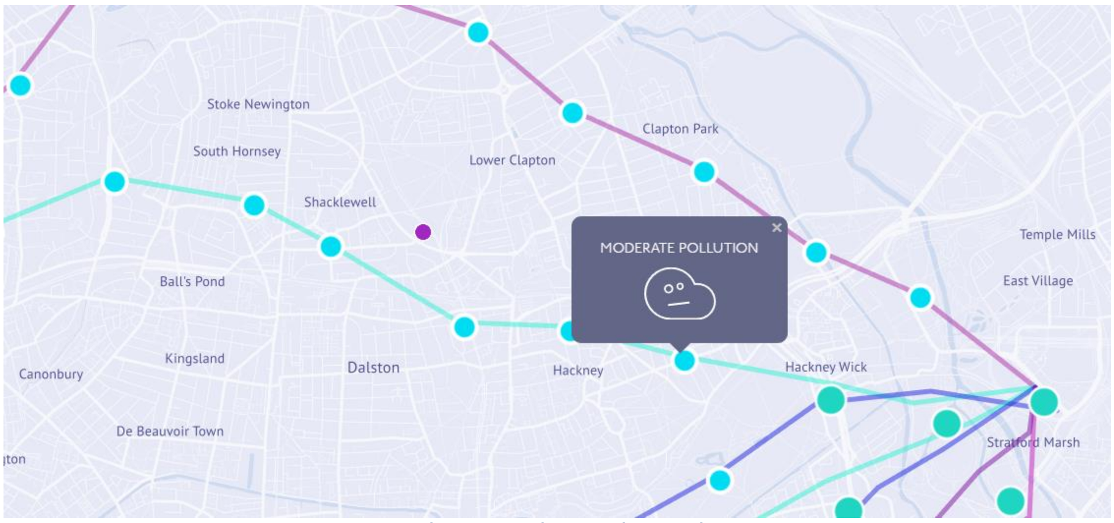
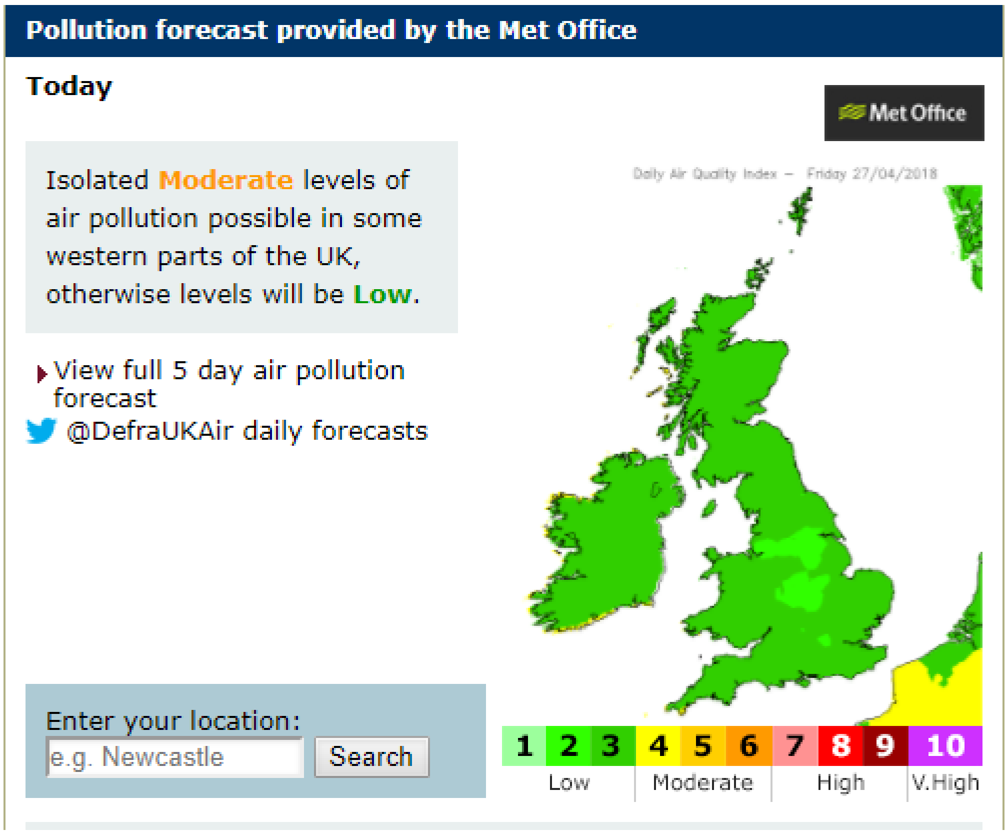


Figure 1 - Pigeon Air Patrol
http://www.pigeonairpatrol.com/

Pigeon Air Patrol also uses an interactive map which shows the air pollution across London. As evidenced in Figure 1, no values are shown, rather an indicator is used whether the area has “fresh air” or a certain level of pollution such as “moderate” or “high”.

Plume labs, the same company that deployed Pigeon Air Patrol also are working on a device to measure air quality and location. It is a smart air quality tracker designed to be attached to a user’s possession, such as a bag or bicycle. This has not yet been released so information on it is limited.

The Department for Environment, Food and Rural affairs (DEFRA) have a pollution mapping website that is forecasted by the Met Office. The information states that the data is collected in various regions from monitoring sites and generated from current air quality issues.



Pollution forecast provided by the Met Office

Today

Isolated Moderate levels of air pollution possible in some western parts of the UK, otherwise levels will be Low.

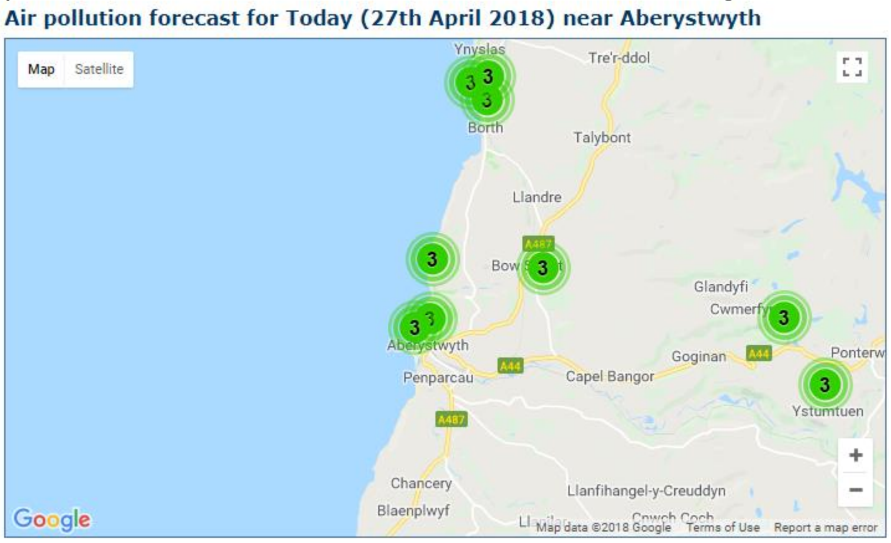
View full 5 day air pollution forecast

@DefraUKAir daily forecasts

Enter your location:

Figure 2 - UK-air DEFRA screenshot
https://uk-air.defra.gov.uk/

Figure 2 shows a forecast of the pollution over the UK, with a non-interactive map, the only interaction is a search functionality. Once a search has been complete, the map becomes interactive using google maps. This then shows values of pollution at different points, rather than the whole area, as can be seen in Figure 3.



Air pollution forecast for Today (27th April 2018) near Aberystwyth

Figure 3 - interactive map on UK-air DEFRA
https://uk-air.defra.gov.uk/

The difference between Figure 3 and the major project is that fixed monitoring stations weren't used. The idea was to use vehicles as monitoring stations as even for local use, can still collect a lot of information regarding the air quality. The visualisation of the map was designed as a heat map/contour map.

The DEFRA website also has links to Wales, Scotland and Northern Ireland air quality sites. The Welsh site lacks the functionality of the UK site. The Welsh site uses less monitoring stations than Wales has on the UK site. The Scottish site has the most interactive welcoming screen.

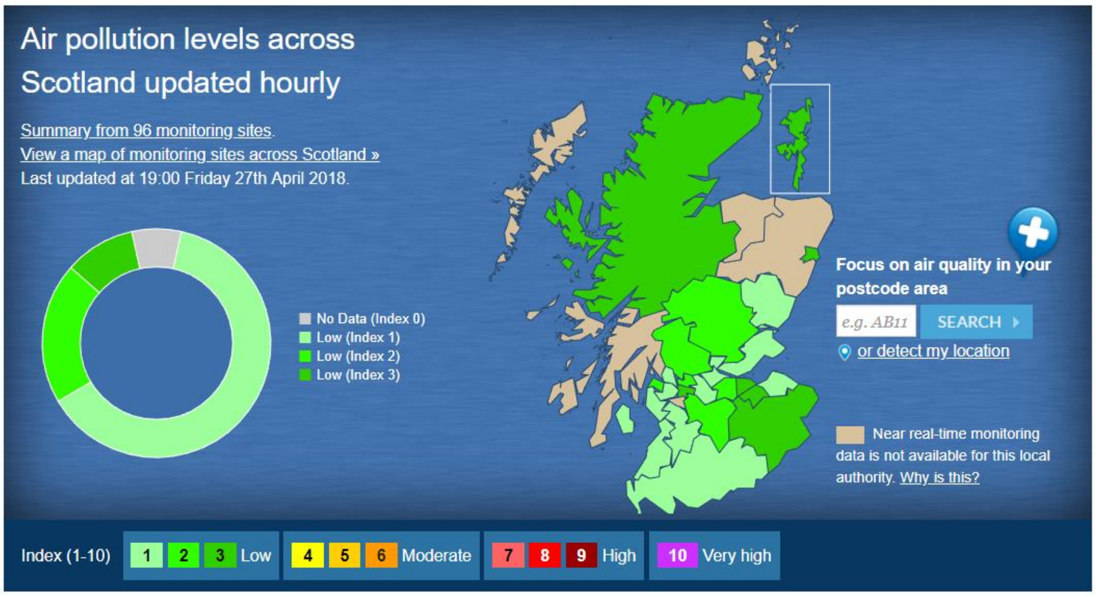

Air pollution levels across Scotland updated hourly
Summary from 96 monitoring sites.
View a map of monitoring sites across Scotland »
Last updated at 19:00 Friday 27th April 2018.
No Data (Index 0)
Low (Index 1)
Low (Index 2)
Low (Index 3)
Focus on air quality in your postcode area
e.g. AB11
SEARCH
or detect my location
Near real-time monitoring data is not available for this local authority. Why is this?
Index (1-10)
1 2 3 Low
4 5 6 Moderate
7 8 9 High
10 Very high

Figure 4 - Interactive welcome screen Scottish Air Quality
http://www.scottishairquality.co.uk/

Figure 4 allows users to select their location by clicking on their province. The design for the visualisation site is different to this and resembles no correlation. The interactive map is the same as the Welsh and UK site, this is not what the major project aimed to achieve.

Riversimple got in contact before the major project started to make me aware of what they thought was a similar website. They had been in contact with the company at an event and were made aware of the website before I was.

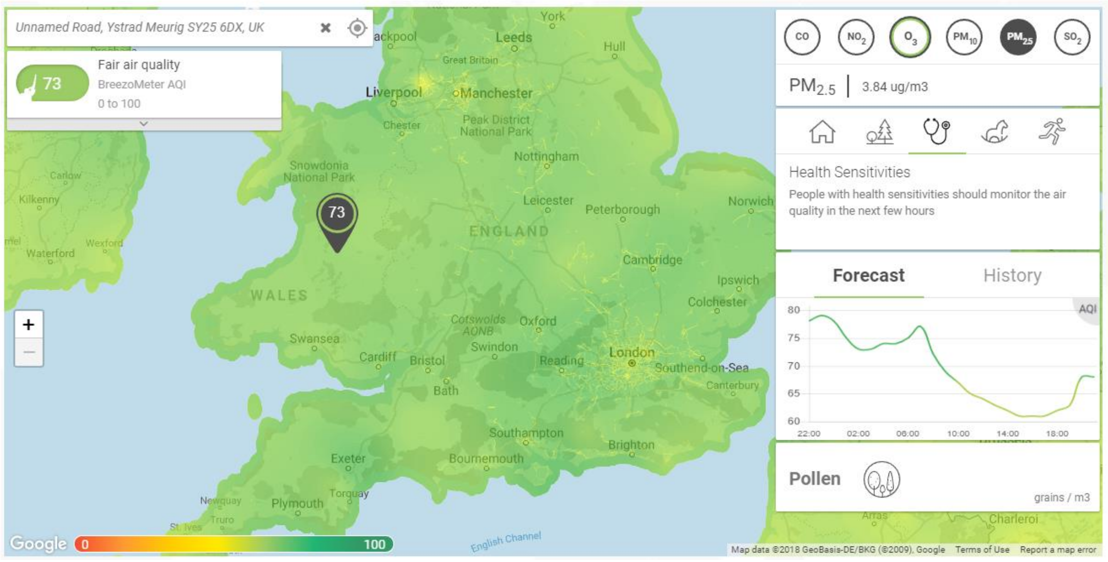


Figure 5 - Breezometer interactive map
https://breezometer.com/air-quality-map/

Breezometer uses weather/air pollution stations and machine learning algorithms to predict the values as seen in Figure 5. The monitoring system designed for this project is to use real world values rather than from monitoring stations and predicting the values. The interaction on Breezometer is very good and offers a lot of information. It shows different pollutants as one value on the map, but on the information menu, it shows individual information on the pollutants. Information is also given on the health issues and sensitivities, this is a similar idea that was not fully implemented on the final version of the visualisation tool.

1.2. Analysis

After studying the background research, it was drawn to a conclusion that this project would be possible within the timeframe. Various assumptions were made during the start of the project. The project would be split into two major components:

1. Monitoring system
2. Visualisation System

The monitoring system would collect the data and would need to be designed to be suitable within automobiles. The output of the monitoring system would be sets of data from journeys that would need to be uploaded to Aberystwyth University MySQL server. The monitoring system would primarily use python3 for development as it was an easy to use and fast learning programming language.

The visualisation system would need to use the data collected from monitoring system that would be stored on the MySQL server. This would then be displayed to the user in a proactive way.

It was decided to use the Aberystwyth University MySQL server to store the data and public hosting for the website as access to it was very easy. It was already set up to suit the needs of this project.

Implementation of the project would require several different skills ranging from the use of hardware components in the creation of the monitoring system to web development.
The required skills needed and those to be developed are:

- Hardware selection
- Hardware creation
- Linux installation, command line and configurations
- Network Administration
- Python skills
- MySQL skills (Database Administration)
- Web development skills

The hardware would consist of a Raspberry Pi Model 3B+, air quality sensor and a GPS. The Raspberry Pi Model 3B+ was used because this resembles the microcontroller that Riversimple use for the telemetry unit that is currently being developed. This would be controlled by a Linux distribution called Raspbian Lite, meaning there is no GUI (Graphical User Interface) and would require an SSH (Secure Shell) connection or serial connection to be communicated with. Raspbian Lite was selected due to the RPI being a lightweight microcontroller, GUI is resource intensive and when the monitoring system is working a GUI is not required. Using the Lite version of Raspbian would require Linux command line skills, it was also chosen to improve these skills for future projects.

The RPI supports many different protocols for components such as I2C (Inter-integrated Circuit), SPI (Serial Peripheral Interface) and provides GPIO (General-Purpose Input/Output). This was another reason that the RPI was chosen. A Raspberry Pi HAT can be used if a hardware component was not supported by default such as CAN (Controller Area Network) which is most often used in the automotive sector.

As it wouldn’t be viable to create the software running on the RPI through the SSH, a Samba server needed to be set up to allow a standard client-server interaction. The Samba server would host the files as it was possible to connect to the server from another computer and edit the files using an IDE or advanced text editor.

It became clear that the software on the RPI would not need to be that advanced but would rely on the Linux distro to be correctly configured to allow the scripts to run on start-up and upload data collected. It was decided that only two files would need to be created to collect and upload the data.

The use of the Aberystwyth University resources would need to be considered. Hosting the webpage on the public_html directory would be needed as it allows the webpage to be accessed from anywhere, which would be useful when giving demonstrations or presentations on the current state of the project. Equally, the hosting server also allows the use of PHP scripts. This would be used to get the data from the server and not allow the user access to the database, it would be handled server side.

It was decided that the visualisation application would use OpenStreetMap (OSM) as the map provider. Due to the fact that the geologically data is open to anyone to use, with only recognition required, the online community for developing OSM applications is quite large. The data provided by Google Maps is copyrighted by many organisations and it wasn’t clear whether it would be limited by Googles API or Terms.

The visualisation application would need to show the data in a proactive way for easy educational purposes. It was proposed that either a contour or heat map would be used to display the data for easy comparisons of different areas. OSM supports a variety of plugins, including ones for creating heat maps plugins, this was another reason to choose OSM over Google Maps.

Security was an issue with this project at this point as connection to a personal database is needed on both systems being developed. At this point security solutions had not been considered.

The functionality of each system had been defined by this point, how the functionality would be implemented was not.

The objectives of work at this point in the project were:

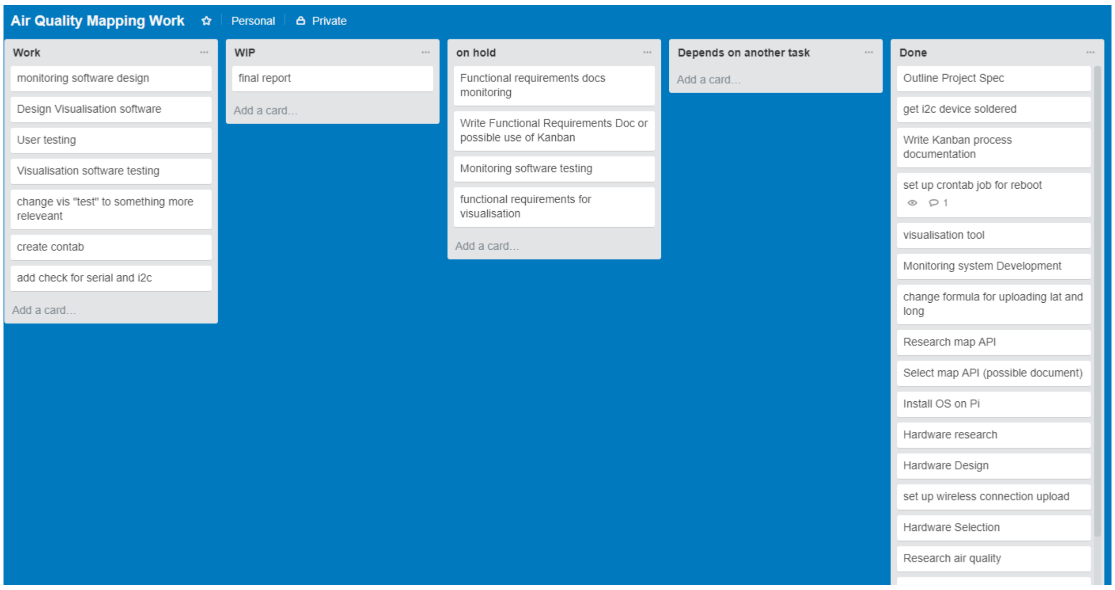
- Select the appropriate hardware to work with the model 3B+ RPI
- Install Raspbian Lite onto an SD for the RPI
- Set up a Samba Server on the RPI for easy development
- Hardware components design and assembly
- Design of Monitoring system
- Development of the monitoring system
- Design of Visualisation system
- Development of the Visualisation system
- Testing of both systems
- Final Report

1.3. Process

At the start of the project, the process instinctively took off with a waterfall approach. Rather than starting with the requirements, the project commenced in the design stage, without requirements being harnessed. This was a bad start to the project as requirements had not been set in place and therefore implementation varied from one choice to another. About half way through the design phase, a new process was decided.

Having experienced agile methodologies within a team, it was known that these methods work by suiting the process to the project's needs. The difference in a solo project was that it would have to be tailored to the project, no matter the methodology chosen.

The changing priorities was a clear problem; it was obvious that a process was needed to fixate on tasks and to ensure that no change occurred unless it was vital. The process that was chosen was Kanban. A document was created for the Kanban process to document how the methodology was adapted for a solo project. The approach created was to focus on software development as much as possible, and only focus on other tasks where necessary.


Air Quality Mapping Work ☆ Personal Private

Work
- monitoring software design
- Design Visualisation software
- User testing
- Visualisation software testing
- change vis "test" to something more releveant
- create contab
- add check for serial and i2c
- Add a card...

WIP
- final report
- Add a card...

on hold
- Functional requirements docs monitoring
- Write Functional Requirements Doc or possible use of Kanban
- Monitoring software testing
- functional requirements for visualisation
- Add a card...

Depends on another task
- Add a card...

Done
- Outline Project Spec
- get i2c device soldered
- Write Kanban process documentation
- set up crontab job for reboot 1
- visualisation tool
- Monitoring system Development
- change formula for uploading lat and long
- Research map API
- Select map API (possible document)
- Install OS on Pi
- Hardware research
- Hardware Design
- set up wireless connection upload
- Hardware Selection
- Research air quality

Figure 6 - Example of Trello

Trello is an online project management application that will be used for the digital boards and cards it can visualise. This online application will be used to manage the project, this includes harnessing requirements of work to be completed. One air quality mapping board will be used (see Figure 6).

The requirements/tasks would be created as cards and these cards would move around the board dependent on what stage they're in. The use of "stories" was not considered, but rather just a task on the cards. The stages were defined as columns, five columns were created:

1. Work – At the start of the project, this would contain all cards that would need to be completed. If a problem or task arose a new card would be created in the work section. The cards would move to the appropriate column when work had commenced on the task.
2. Work in Progress (WIP) – If a card was being worked on then the card would move to this column. A maximum number of 3 cards would be worked on at the same time, this was to prevent many tasks being allocated at once. If cards are dependent on each other and need to be worked on, then they would move to the "depends on another task column".
3. On hold – If a task needs to be put on hold for any reason (prioritise work) then this column should be used. This column was a last resort and would only be needed when necessary.
4. Depends on another Task – If work had begun on a card and it was dependent on another task being complete first, then the card would move to this section. An example of this was developing python code to connect to an I2C bus but the I2C bus had not been connected.
5. Complete/Done – Once a task had been completed then it will move to this column.

A Gantt chart was created at the start of the project to measure progress, though the completed stories would show progress in its own way, it did not show effort complete and remaining effort. The Gantt chart was created for the first nine weeks of the project (Appendix A), once the first nine weeks had been completed then another up to date Gantt chart would be created for the remaining time (Appendix B). The Gantt chart predicted the time it would take to complete each major task. It would be often updated to show work completed on each task with the use of a percentage. The Gantt chart was created using a template provided by Microsoft Excel.

A version control system was used throughout the project, this was GitHub. All work regarding the project was to be stored and version controlled. This was to prevent any work to be lost or accidentally deleted. It is a way to manage work without having multiple versions scattered across multiple devices. In general, it makes the process of completing the project a lot easier.

Once a task had been completed the idea was to write an in-depth explanation on an online blog to keep track of what had been completed. This was for the benefit of writing this report and for the supervisor to keep up to date.

2. Design

2.1. Overall Architecture

As stated in the analysis (Section 1.2) the overall system is split into two systems. The two systems do not communicate with each other but rather share the same data that is hosted on the Aberystwyth MySQL server. The two systems are the monitoring and visualisation systems, these could be split into further subsystems:

1. Monitoring System
    a. Hardware Architecture
    b. Data Collection
    c. Data sharing
2. Visualisation system
    a. User interface
    b. Data retrieval

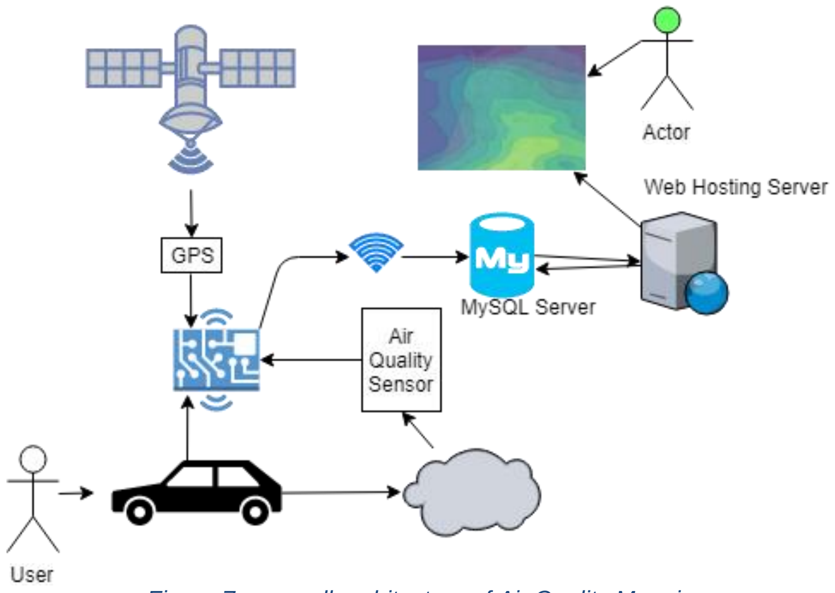


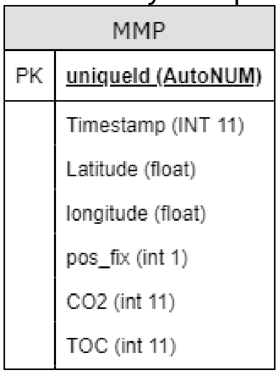
Figure 7 – overall architecture of Air Quality Mapping

As Figure 7 shows the overall architecture of the project. The device is designed for vehicles; those vehicles will produce pollutants which the air quality sensor will detect and communicate a reading to the microcontroller (RPI). The GPS will do the same task when a location has been found. An NMEA sentence will be transmitted to the RPI and will need to be decoded. The air quality and location of the reading will need to be matched with one another, so the data isn't rendered useless. Once a connection to an online network has been made, the data that has been captured should be upload from the RPI to the MySQL server. Once the data has been uploaded to the server it should be removed from the device. The data will remain on the server and be queried by the web visualisation application. The data will then be displayed to the user using OpenStreetMap and a heat map overlay.

As the data is not very detailed and will only require several parameters, the MySQL table will not be very advanced.

Rather than a web application, a standard desktop application was considered with the use of Java or Python. This would not be viewable through a browser but rather its own application. This was dismissed when considering the customisability of using HTML (Hypertext Mark-up Language) and CSS (Cascading Style Sheets) on webpages.

Hosting the webpages and SQL server locally rather than the university network was considered but the two servers were already set up for needs of the project.



MMP	
PK	uniqueId (AutoNUM)
	Timestamp (INT 11)
	Latitude (float)
	longitude (float)
	pos_fix (int 1)
	CO2 (int 11)
	TOC (int 11)

Figure 8 - Table UML

As the MySQL server only needs to host a small amount of data, only one table would be necessary. Every table needs a unique id to identify tuples from one another. The attributes in Figure 8 are very minimalistic and does not include a lot of information that can be retrieved from the RPI, GPS and air quality sensor. Both the monitoring and visualisation systems will require access to the MySQL sever, so the data stored needs to fit the requirements of both.

2.2. Hardware Design

Having little knowledge of hardware made the design process difficult. One of the only decisions made entering the design was that a RPI was to be the microcontroller. The RPI could communicate with multiple protocols and had Wi-Fi technology built into the board.

The process started by looking at the niche component, the air quality sensor. Many sensors were considered in this process, there were several reasons why some sensors were disregarded:

- Detecting if a gas was present rather than returning a concentration value
- Detecting gases that were no harmful to the environment of human health
- Too expensive
- Required the use of third party libraries or required RPI HATs to communicate with, essentially a non-supported protocol was being used.

The ams IAQ-CORE P sensor (Appendix C) suited the needs of the project using an I2C bus that was supported by the RPI. The sensor returned a reading of TVOC (Total Volatile Organic Compounds) which are produced from several sources and are hazardous to both the environment and human health. Along with the TVOC reading, a carbon dioxide estimation was made by the sensor and could be returned in the I2C message. This was the sensor that was implemented in the design and implementation.

A GPS was needed, this was an easy decision as the GPS modules would do the same task. The deciding factors were to find a GPS module that had a supported protocol on the RPI and that would be easy to wire on a breadboard. The PmodGPS receiver (Appendix D) was chosen as it fit these requirements. It uses the UART protocol which is supported by the RPI and is a variation of a serial connection.

Once the components were selected, a diagram was created using Fritzing, which is an open source computer aided design software package that allows the design of circuits.

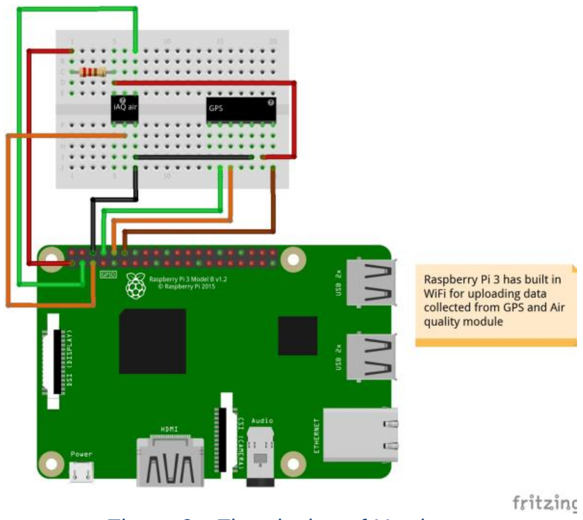

iAQ air
GPS
Raspberry Pi 3 has built in WiFi for uploading data collected from GPS and Air quality module
fritzing

Figure 9 - First design of Hardware

RPI Pin	GPIO number	Use
1	-	3.3v power used for the Air Quality Sensor and GPS.
3	2	I2C data bus connected to the Air Quality Sensor.
5	3	I2C clock connected to the Air Quality Sensor.
6	-	GND used for 0v for both components.
8	UART0 TX	The transmit serial line connected to the receive line on the GPS.
10	UART0 RX	The receive serial line connected to the transmit line on the GPS.
12	18	General output connected to the reset pin on the GPS.

Figure 10 - RPI Pinouts

Figure 9 shows the first design that was created for the hardware. It consists of the use of the serial and I2C dedicated pins. Figure 10 details the use of each pin on the RPI and its use. This was the design when development first began but it was soon realised that this would not work and the design for the monitoring system hardware would be changed.

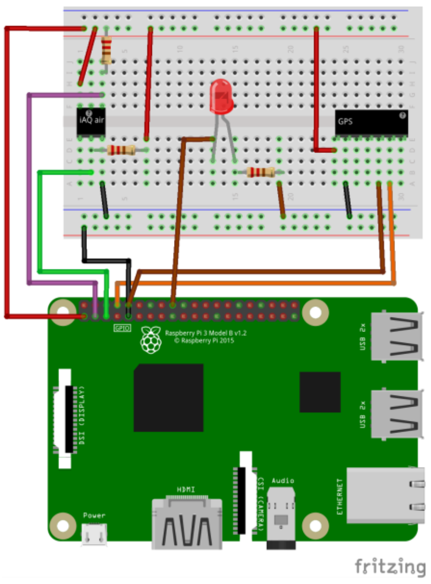


Figure 11 - Final design for the Hardware

RPI Pin	GPIO number	Use
1	-	3.3v power used for the Air Quality Sensor and GPS.
3	2	I2C data bus connected to the Air Quality Sensor.
5	3	I2C clock connected to the Air Quality Sensor.
8	UART0 TX	The transmit serial line connected to the receive line on the GPS.
9	-	GND used for 0v for both components.
10	UART0 RX	The receive serial line connected to the transmit line on the GPS.
18	24	General output used to flash LED, this will show what state the data logger is currently in.

Figure 12 – Final RPI Pinout

Figure 11 shows the final design for the hardware made mid-way through development. The changes made to the I2C were to include a pull-up resistor on the clock and data bus. This is necessary for I2C as the interface can pull the signal low but cannot drive it high, the pull-up resistors are used to restore the signal to high when there is no low signal.

The reset pin on the GPS was disconnected as it was never used within the implementation. The GPIO output has been moved to pin 18 and will control an LED to signal the user what state the data logger is currently in.

2.3. Monitoring Software Design

The software for monitoring is split into two parts. The data logging which consists of retrieving data from the GPS and air quality sensor then writing it to file and the uploading of data collected by the logger to the MySQL server.

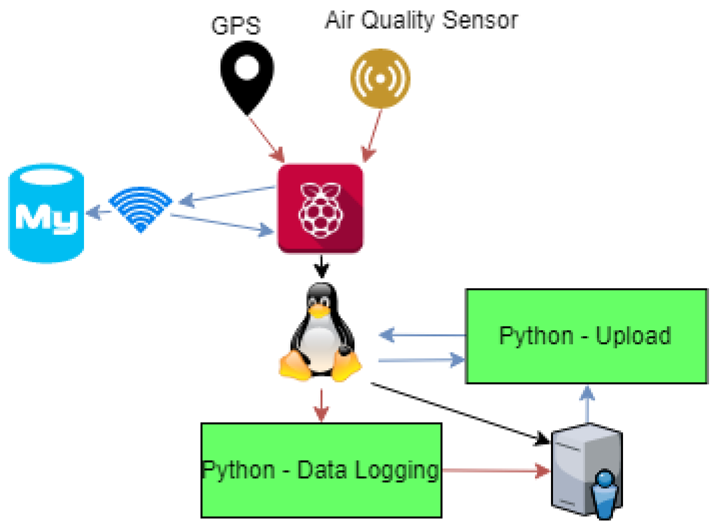


Figure 13 - RPI Overall Software Design

The language chosen to develop the software for the monitoring system was python, this was due to its vastly available libraries, functionality with hardware and online support. As the operating system on the RPI will be a distribution of Linux, python was already installed and libraries would be easy to install. The python files would be created in a directory of the default user with a secure password, the default user is Pi. The style of programming would be procedurally; Python allows multiple programming styles, but procedurally programming favours iteration, which both files were predicted to use most of the time.

As shown in Figure 13 the data logger python script would take readings from the GPS and air quality sensor. These readings would be passed to the OS from the RPI hardware. This data would then be stored onto the local directory of the Pi user. The upload file will be run on occasion to check if a connection has been successful to the MySQL server. If a connection to the server is a success, then uploading should begin. Once a file has been uploaded to the server then it should be removed from the RPI.

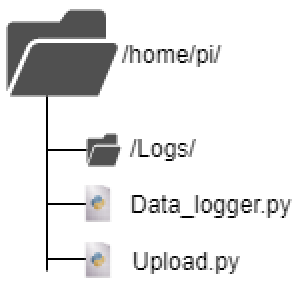


Figure 14 - pi user file structure

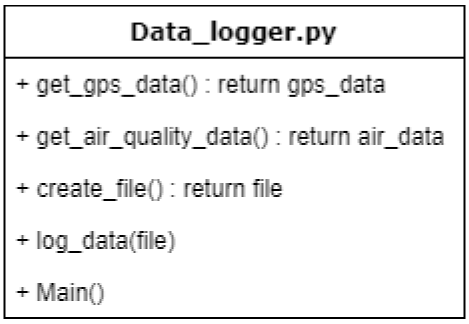
The designed file structure for the pi user is shown in Figure 14, this contains the two files needed to run the monitoring system. The files are shown to be in the pi user's directory. The logs folder was designed for storing log files created by the monitoring system.

The files stored in the logs directory were designed to use the same attributes as the table attributes shown in Figure 8. On the first line of the log file the attributes were to

be listed. The data would then follow in the same order as the attributes on a separate new line each time data was appended to the file. An example of the design of the log file is shown in Appendix E.
The log file name was designed to be unique, it was to be the exact time and date the file was created to prevent any conflicts. If a file were being written to, it was decided the file name should contain a unique character at the start. This had not been decided at this point.

2.3.1. Data Logging (Data_logger.py) Design

The data logger's intention is to retrieve readings from the GPS and air quality sensor, then write these to logs files. The functionality of the python script was not difficult to understand, so it was believed that not many functions would exist in the script.



Data_logger.py

+ get_gps_data() : return gps_data

+ get_air_quality_data() : return air_data

+ create_file() : return file

+ log_data(file)

+ Main()

Figure 15 - Data_logger.py functions

The functions shown in Figure 15 were expected to provide the required functionality for the data logger script.

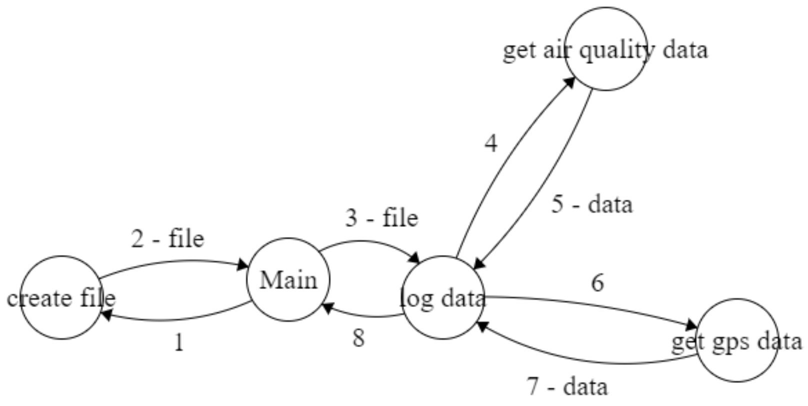


Figure 16 - state machine showing designed function calls for data logger

The "*main()*" function would be used to call other functions in the correct order and would act as a loop when creating new files to log.
The first file that would be called would be the "*create_file()*" function which would return a file-path or file object type. This would return back to the "*main()*" function. The file would then be passed to the "*log_data()*" function, data would be passed back to the log data function from "*get_air_quality_data()*" and "*get_gps_data()*" in a loop to gather multiple tuples of data to store in the file. Once enough data has been appended to the file the cycle would begin again starting at main. This process is shown in Figure 16.

2.3.2. Upload (Upload.py) Design

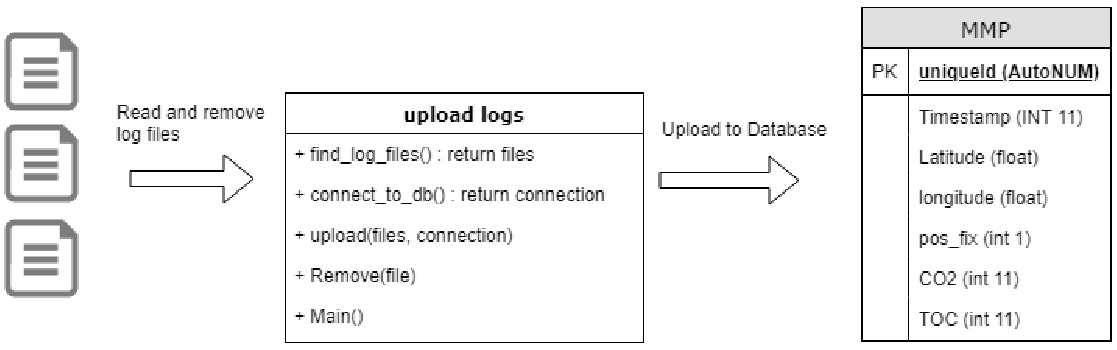


Figure 17 – Upload.py functionality

The script will use the same programming style as the data logger which is procedurally. The scripts intention is to use the data collected by the data logging script and upload the data to the University's MySQL server as shown in Figure 17. The design for this wasn't really thought about until implementation but a broad design was considered.

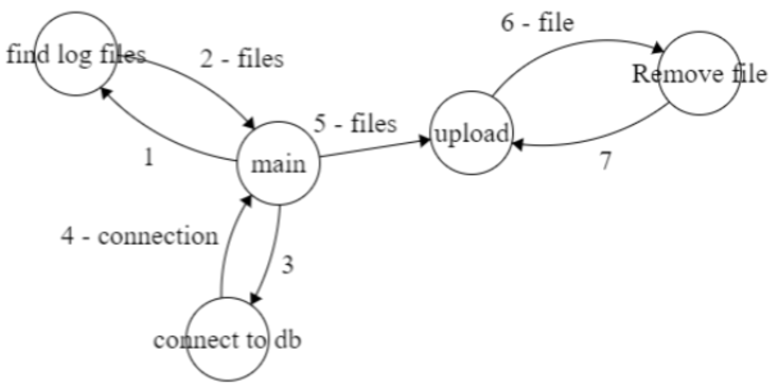


Figure 18 - state machine showing designed function calls for upload.py

The "*main()*" function will provide a similar functionality to the "*main()*" function in the data logger script. It's to ensure that functions are called in the correct order and sets any variables needed. The first function to be called would be the "*find_log_files()*" which would provide a list of the available files to upload in the "/log/" directory. The "*connect_to_db()*" would be called next and that would provide a secure connection to the MySQL server. The "*upload()*" function would use this connection to read the files and upload the data. Once the file had been completed it would be removed from the directory and the function would move onto the next file.

2.4. Visualisation Software Design

The aim for the visualisation software was to have an online application to educate the public about air pollution on public roads. This meant that the web application had to be used to communicate with the MySQL server. To communicate with the server credentials would be necessary, and these would need to be hidden from the public. Therefore, a hosting server with PHP was necessary.

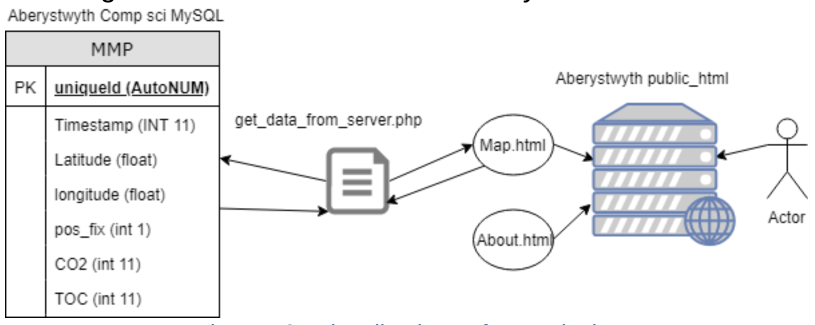


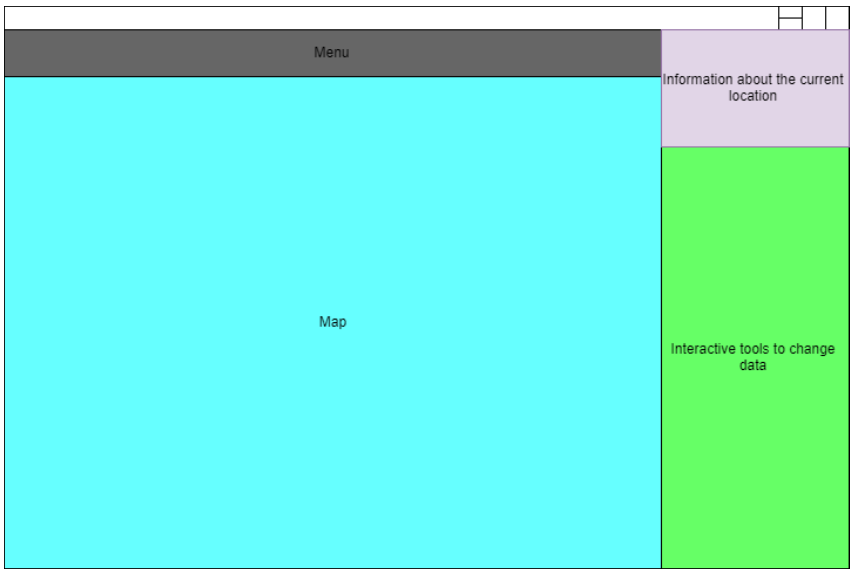
Figure 19 - visualisation software design

Figure 19 shows the Two pages were designed to be visible to the user and these were the main page ("map.html") and the about page ("About.html"). The user would access the pages through the Aberystwyth University student hosting server. Any PHP scripts could be run server side, this was designed with security in mind. The PHP would contact the MySQL server and retrieve all the data required for the map page.

The about page would not require any special functions (JavaScript or PHP). The map page however would require additional functions other than the PHP script. This was to handle the third party OpenStreetMap and additional plugins to draw the heat maps. The functions would be necessary if any manipulation of data was needed for the plugin. As the web page was designed to have interactive tools to change the data, functions would be needed in this case.

2.5. Visualisation User Interface Design

Each page on the website will contain a menu at the top of the page used for navigation purposes. The navigation options will be the “map” and the “about” pages. This will be used using an unordered list and links.


Menu
Information about the current location
Map
Interactive tools to change data
Figure 20 - map.html page user interface design

The page was designed to have designated sections (as seen in Figure 20) this would be completed using the “div” tags provided in HTML and using CSS to allocate sections of the page. This is designed to be responsive on window size change using the “%” in CSS rather than “px”. Percentage will take a percentage of the page rather than use a pre-allocated number of pixels.

The map page was designed to be the home of the website. A large proportion of the page was dedicated to the embedded OpenStreetMap world map. This would allow users to interact and view anywhere in the world. There would be an overlay on the map controlled by a plugin that would provide the heat map.

Pollution level Information about the current location of the maps would appear in the top right of the page. This was to include information on the effects to human health and the environment. It would state what levels are low and which are dangerous.

The interactive tools section of the webpage would allow the user to switch data from TVOC and CO2 readings to be shown on the map. The interactive section would allow the user to change the data, but the ways in which the user could do this was not considered. The only thought about design was to include filtering to remove certain data points.

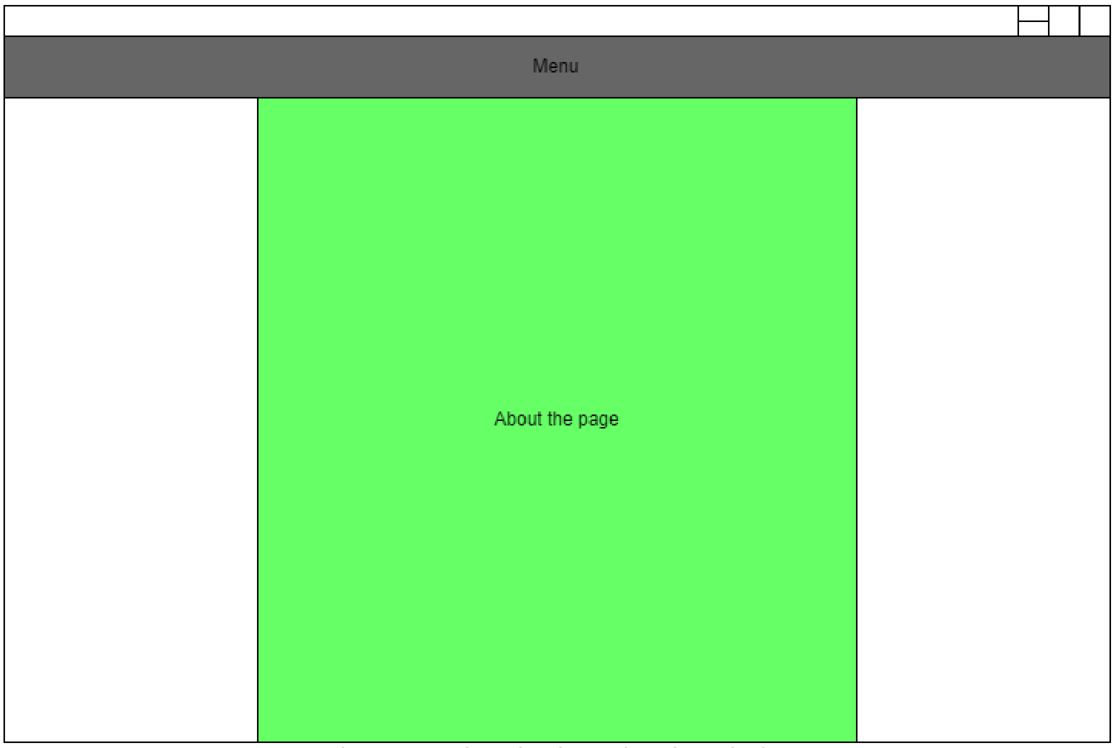


Figure 21 - About.html user interface design

The about page was designed to display information about the website. This could include any thanks needed for libraries or installation instructions for the website. The “div” tags will be different for this page compared to the map webpage. The menu bar was designed to be 100% of the window width, stretching across the page. The CSS should centre the information on the page.

3. Implementation

3.1. Hardware implementation

When the hardware implementation had been designed and the interfaces were confirmed to work with one another, the parts were ordered.
Between the first hardware design and implementation a mistake was noticed when selecting the air quality sensor. The type of mounting had not been considered, the air quality sensor was a type of surface mount and had no pins to connect to. This was overseen as the functionality of the device was focused on. To overcome this problem a solution was found by Dr Neal Snooke, who suggested that because of the spacing between the mounts was like the spacing found on a breadboard, 2.5mm pin header strips were soldered to the air quality sensor mount spaces.

The design for the I2C interface changed when it was realised that pull-up resisters were needed (as mentioned in Section 2.2 Hardware Design). This was noticed during development of the communication between the air quality sensor and the RPI (as mentioned in Section 3.3.4).

During development it was difficult to know whether the RPI was logging any data, this then led to the LED being implemented for the hardware. The script would cause the LED to flash a number of times depending on the current state.

3.2. Setting up the environment for the Raspberry Pi

3.2.1. Installing Linux and connecting to the RPI

When setting up the environment it had been decided that the OS wold be Raspbian Lite distribution of Linux. The OS image was downloaded from the Raspberry Pi website. The website mentioned the use of a SD flasher for mounting the image called Etcher. Once the installation had completed, it was connected to the RPI. The RPI was connected to the local network using an Ethernet cable, the default user that is created is “pi” with the hostname of the RPI being “raspberrypi”. Putty was used in attempt to connect to the RPI to set up the WLAN using “pi@raspberrypi” as the hostname, though this failed. After researching it was found that an additional file needed to be created on the RPI in order to use SSH. The file that needed to be created on the boot partition was “ssh” with no file extensions. The process was repeated and the RPI could successfully be connected to the WLAN, not needing the Ethernet connection anymore. The hostname was then changed, it is advised to change the default hostname to prevent confusion if another pi was introduced to the same network, and it was changed to “rdm10pi”.

3.2.2. Setting up the Samba server

As logging into the pi and editing files would no be an efficient or graphically friendly way of creating the python scripts, it was decided to set the RPI as a server to simulate the standard client-server arrangement. This took longer than expected. Samba was installed through the command line using “*sudo apt-get install samba samba-common-bin*” which installed with no problem. The Samba configuration file needed to be correct for it to work with the Windows computer on the network. The Samba configuration was located in “/etc/samba/smb.conf”. A configuration (Appendix F) was appended to the end of the file but it didn’t seem to appear on the network. This lead to researching the documentation for Samba. It was discovered an additional setting needed to be included to the configuration file which was “`wins support = yes`”, this enabled Windows support. The file share was easily mountable to the windows file explorer, this led to easily creating new files and folders within the pi user’s home directory. An IDE or source code editor could then be used to develop.

3.3. Monitoring System – Data Logger

3.3.1. Retrieving serial readings

The data logger started by trying to get readings from both the UART interface on the GPS and the I2C interface on the air quality sensor. Before Serial or I2C could be implemented, it needed to be enabled on the RPI. This was done through the RPI configuration screen using the command "*sudo raspi-config*".
This started by using the "serial" python library, this had to be installed using "*sudo apt-get install python-serial*". It was very simple to set up. Firstly, a serial port needed to be opened, this involved using the GPS's data sheet to find the correct settings of the serial port. This included the baud rate, whether the serial used parity or stop bits and the byte size.

Once this object was created, named "serial", all that was needed to read a NMEA sentence on the serial port was "*serial.readline()*" though this was in the format of a serial message object, part of the serial library, and was difficult to access the important parts of the message, such as longitude and latitude. To transfer the message into a string, the message must be decoded into a string encoding such as "UTF-8" using the in place function "*.decode('utf-8')*". To easily access parts of the serial message the message was split up into a dictionary (a dictionary is an associative array, they are indexed using keys of any data type) using a string as the key. The string chosen for each value represented what the value stood for. The dictionary was created using two lists, a list of keys, being labelled strings, and serial sentence. The serial sentence was one string containing all the values, but using "*.split(',')*" would split the string into a list every occurrence of a comma. For an example of this see Appendix G.

3.3.2. Format of GPS data

It was decided that when getting data from the GPS, a specific sentence would be required before returning the dictionary. This sentence was a "GPGGA", this was chosen due to the contents of the data. The most important values to retrieve from the GPS were longitude and latitude for this project. GPGGA contained these values along with a position fix indicator that was used to identify whether the GPS had a fix or not.

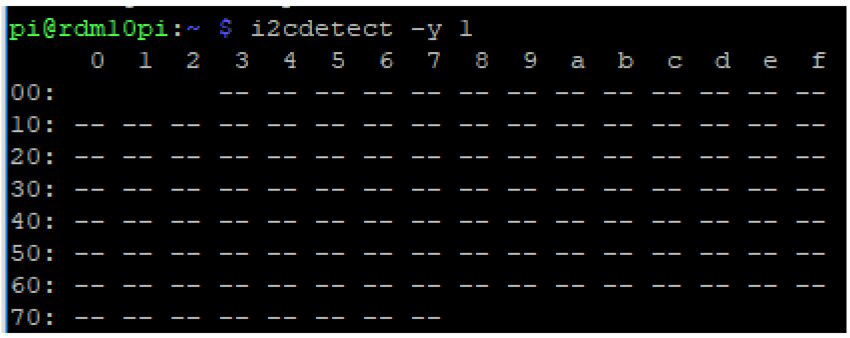
Late in to the project a mistake was noticed when converting the latitude and longitude values. It was presumed that the values needed to be divided by 100 to get the correct format of decimal degrees. The format of the data was given in degrees minutes seconds (DDMM.MMMM) which is a string of two numbers concatenated. The degrees needed to be separated from the minute's seconds and a calculation needed to be performed.

3.3.3. Starting a logging method

Before implementing the I2C interface, logging was introduced. Writing to a file in python is very easy with required functions being in the python core package. Information from the GPS was stored in a file, but when viewing the file blank lines would appear between each line of information. This was due to the decoding of the serial sentence to a string and the function used to write to the file. These two functions would add '\r\n' causing a blank line between each message. Headings were added to the file on creation. Creating the file and logging were both similar to the design in the sense of functionality but had different variables than expected or were called in a different order.

At this stage it was realised that the implementation was already deviating from the design with required functions. Two functions had been created "*get_and_translate_gpgga(serial)*" and "*set_up_serial()*" instead of the one function in the design. This wasn't a problem though due to the process being much focused on software development rather than documentation and requirements. Kanban allowed for easy change in requirements and design with new cards being easily added to the board.

3.3.4. Implementing the I2C interface

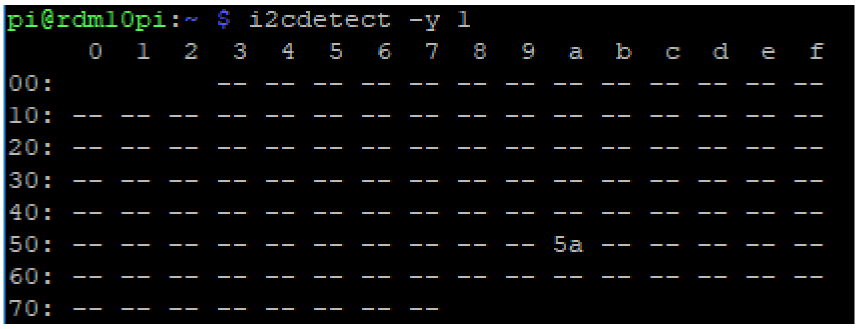


```
pi@rdm10pi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 22 - Detecting I2C interfaces

Creating communication for the I2C device was difficult compared to the serial from the start to finish. Firstly I used the Linux command "*sudo i2cdetect -y 0*" but the I2C default bus on the RPI 3 changed to bus 1 so this caused an error. Once I changed the bus I was given a table showing all available addresses. This worked by sending a signal to the address and checking for a reply (Figure 22).

The datasheet stated that the device needed a very stable 3.3v otherwise there would be no output. As the I2C was connected to the 3.3v from the RPI it wasn't thought that this would be the problem, but experimentation was necessary. A voltage divider was created from the 5v output of the RPI to have an output of 3.3v. This didn't change anything, so the circuit was reverted back to its original state.

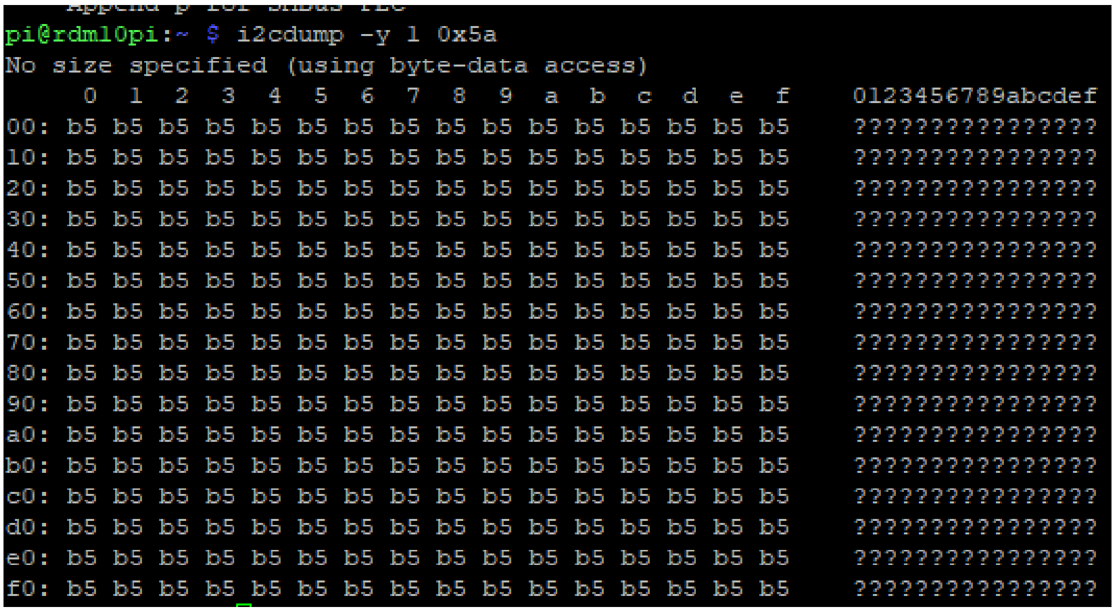


```
pi@rdm10pi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- 5a -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 23 - Successfully detecting I2C interfaces

After many attempts it appeared to be a connection issue, not that wires were in the wrong place, but that pull-up resistors were needed. The company that manufactures the air quality sensors also produce a click board with the same air quality sensor. A datasheet was shown with the click board and an electrical schematic was available [4]. The schematic showed an implementation of the I2C interface, including pull-up resistors. This showed the values required for the pull-up resistor which was 4.7K. Resistors with similar values were used and an output was successful (Figure 23).

Now that the RPI was correctly communicating with the I2C interface, software development could begin on receiving measurements from the sensor. SMBus (System Management Bus) functions were used to attempt to request data from the air quality sensor. The SMBus functions were part of the SMBus library. A small python script was created to test interaction with the I2C device. Once the bus had been configured using simple python command, requesting data was attempted (Appendix H - Requesting data using SMBus).

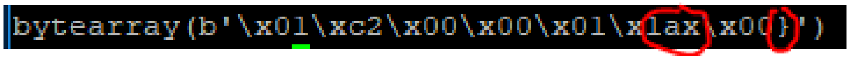


```
pi@rdml0pi:~ $ i2cdump -y 1 0x5a
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
10: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
20: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
30: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
40: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
50: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
60: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
70: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
80: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
90: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
a0: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
b0: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
c0: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
d0: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
e0: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
f0: b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5 b5    ????????????????
```

Figure 24 - i2cdump on Raspberry Pi

The only response, no matter what was requested, would be "181" or 0xB5 in hexadecimal. After looking at the datasheet for the air quality sensor it was noticed that the start of each message, the first byte would be 0xB5. When asking for multiple bytes each byte would still only contain the value 0xB5. "*i2cdump*" is much like the i2cdetect function, it displays all available registers through the I2C bus. Once "*i2cdump -y 1 0x5a*" was executed on the command line it printed all available registers (Figure 24). An attempt was made to read all the bytes available on the I2C bus address, but this made the RPI completely unresponsive.

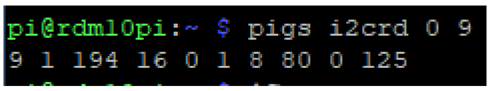
Different functions within the SMBus library were used to attempt to request data from the air quality sensor but none worked. Research began on possible solutions. On the StackExchange forum [5] a post was found of a person with the same component having the same problem. The problem was caused by the I2C device resetting its internal state when a start condition is seen. The solution to the problem was using a different module called "pigpio".



```
bytearray(b'\x01\xc2\x00\x00\x01\x1ax\x00}')
```

Figure 25 - Unusual output of byte array

Once the SMBus library was replaced by the pigpio in the interface testing script, reasonable results were being returned from the device and printed to the console. When 9 bytes were requested and were printed to the console in a byte array unusual symbols would appear that weren't hexadecimal. "x1ax" and "0x00}" were not hexadecimal numbers so it made it difficult to use these values, especially when these contained figures needed for the air quality. It was believed that this was due to printing the results before closing the I2C connection. This however was not the case.



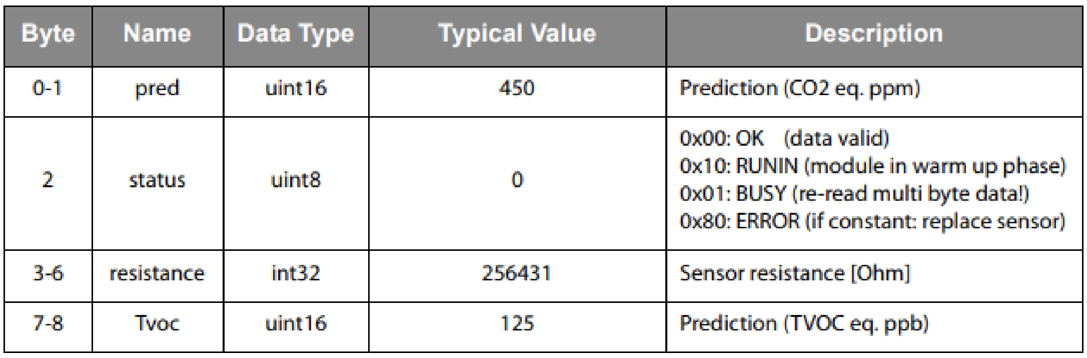
```
pi@rdm10pi:~ $ pigs i2crd 0 9
9 1 194 16 0 1 8 80 0 125
```

Figure 26 - pigs I2C success

As pigpio was a library for the RPI and as well as a module for python, this allowed commands to be run through the terminal. Using "pigs" commands I was able to get a reasonable result without fail (Figure 26). The problem seemed to be occurring in the python script.

After consulting with users on the Raspberry Pi Stackexchange [6] it appeared to be how python would display a byte array. If the value of a byte is the same as a printable character, it would display the printable character instead. The "x1ax" would actually be two separate elements, such as "\x1a\x78\". Once that was cleared up the byte array was accessed in the same way a normal array would be.

The software to communicate with the I2C bus could then be implemented alongside the serial in the data logger. The air quality sensor has a warm up state, this is shown in one of the bytes depending on the value. When requesting information from the sensor if it is in warm up mode then another message will be requested after a short period until the warm up state has finished and the sensor is returning values.



Byte	Name	Data Type	Typical Value	Description
0-1	pred	uint16	450	Prediction (CO2 eq. ppm)
2	status	uint8	0	0x00: OK (data valid) 0x10: RUNIN (module in warm up phase) 0x01: BUSY (re-read multi byte data!) 0x80: ERROR (if constant: replace sensor)
3-6	resistance	int32	256431	Sensor resistance [Ohm]
7-8	Tvoc	uint16	125	Prediction (TVOC eq. ppb)

Figure 27 - Format of bytes (Air Quality Sensor)

While implementing it was noticed that the CO2 and TOC bytes were in big endian format, a function was created to extract the data from the array and create a value from the two bytes before logging the values. The bytes in position 3-6 were ignored as this is just the resistance across the sensor, after checking the datasheet, this would not affect the TVOC or CO2 readings.

3.3.5. Logging method

When readings were able to be logged from both sensors, the file that the data was being logged to would continuously grow. This was thought to be a bad idea as a lot of data relies on one file not corrupting. Data was then stored in maximum sized files of 25kb. This led to another problem of not knowing which file was currently being written to. To resolve this issue, while a file was being used the filename would start with "~", like temporary documents. Another function was then added to prevent the build-up of files starting with "~" if the RPI was turned off during logging. The function "*check_previous_files()*" is called before logging any results. The function searches through the logs directory for files beginning with "~" and will remove the character from the filename.

The first prototype had been created for the data logger. The RPI needed to start logging when it was switched on as no GUI would be provided when collecting data. This caused frustration as it was difficult to know whether the device was able to log any data or was waiting on the GPS and air quality for data. This is when the LED was implemented in the hardware. An "*led_out(flashes)*" function was created to control the GPIO that the LED was connected to. At different stages in the script the LED would output several flashes to identify to the user what the current state is.

3.3.6. Creating cron jobs for the data logger

To ensure that data would be collected on start-up of the RPI a cron job was created. Cron jobs are used for scheduling tasks at required times, this is used to schedule jobs in most unix systems, in this case it would be to start the logger on reboot. "*crontab -e*" on the command line would allow a cron job to be created. To run the logger "@reboot sleep 20 && /home/pi/startlogger.sh" was appended to the end of the file. The shell script that the cron job would execute would start the "piggpiod" process to allow the interaction between the pigpio library and with the RPI interfaces. The script would then start the python script to log data.

3.3.7. Analysis of the data logger

Overall the software was not complicated to get readings from either the serial or the I2C bus. It took a lot of preparation with the hardware and time to use the libraries in the correct way. This caused the data logger for the monitoring system to take a lot longer than expected.

3.4. Monitoring system – Uploading Data

The python script to upload data (python.py) needed to check for an internet connection and then upload the data collected by the logger to the Aberystwyth University's MySQL server.

3.4.1. Connecting to the MySQL server

The first function created was to check for an internet connection. It's easier to check for a network connection rather than an internet connection so this required some thought. The simplest method of doing this was to ping an online target. This was done using a socket library, this would allow access to the socket interface and would create a connection to an address. While implementing the script the address "www.google.com" was used. The function would return true if a connection was valid but false if there were an error.

At this point the script needed to connect to the MySQL server. It was soon realised that to access the server a local on-site connection needs to be made, or a virtual private network (VPN) be set up. As the University had strict rules on what could connect to the network this left one option. The VPN had to be set up on the RPI.

Firstly, an attempt was made to use a Linux package called "pptp" meaning point-to-point tunnelling protocol. Through the terminal to test the VPN connection the following command was executed:

```
sudo pptpsetup –create abervpn –server vpn.aber.ac.uk –username
rdm10@aber.ac.uk –password ************ –start
```

This caused an authentication error. It was thought that this was due to the wrong protocol being used, that the university didn't use a point-to-point tunnel protocol. Communications with information services at the university suggested that OpenVPN be used as this works for unix based systems. OpenVPN is supported on the RPI so pptp was removed from the RPI and OpenVPN was installed. OpenVPN requires a configuration file with certificate keys and various other parameters to start the VPN. The windows client that information services provides for connection to the VPN had the configuration zipped inside of the .exe. This configuration was edited to read in the username and password from a file in the /etc/openvpn directory. Once this was extracted and moved to the correct directory "*sudo openvpn Aberystwyth.opvn*" would appear to connect successfully to the VPN. To run this as a background process the "—daemon" tag was needed. Though the VPN stated it was successful, a test was conducted to ensure it was working. This involved using the command "curl http://ipecho.net/plain" to return an ip address before and after the connection had been made.

To run the VPN in python a module was needed called subprocess, this allowed unix commands to be run in script. It was very simple to use, the commands needed to be in an array with each command being its own element.

3.4.2. Executing statements on the MySQL server

Now that the VPN was running and connected to the university, network connection to the MySQL server could be established in the python script. To connect to the server an additional package was used called "pymysql", a free open source MySQL client library. The GitHub readme showed many examples of the package being used, this was very easy to follow. A connection (representation of a socket) needed to be made using the servers host name and user credentials. Instead of these details being written in the code, these were placed in a separate text file and read in, though they were still unencrypted.

A cursor object would then be used to execute commands on the server. A test was connected to print all the available tables, this was then implemented into a function to ensure that the table needed for the data upload existed on the database. If the table didn't exist, then it would create the table using the cursor object. This was very easy to set up as many examples guided the process, though one mistake was made. When testing the commands, a "*`DROP table`*" command was being issued but was replaced with a command to drop all the tables from the database. This was a problem as a blog with in depth detail of processes during this project was unable to connect to the table containing all the posts. After being resolved with computer science support, backups of the database were made on a frequent basis.

As connection to the university network was necessary to upload any of the collected data, rather than checking the internet connection for a connection to Google in the *`connected()`* function, a connection was made to the host name of Aberystwyth University's VPN "vpn.aber.ac.uk" to ensure it was online. This caused issues as a connection couldn't be created but it was possible to ping the address; the function then used a ping method instead.

At this point a successful connection to the database could be made and if the table that was needed didn't exist, it would be created. One last function was created and that was to construct a MySQL statement for each file in the logs directory. The files would be read one by one and uploaded to the database. The statement was built using a recursive function to add the values to the end of a string. This statement would then be executed to the server but would not be committed to the table. This wasn't until the "*`commit()`*" command was run. This was to ensure all data from the file was uploaded to the server in case the RPI was powered off and ensure a valid transaction. Once a valid transaction has been complete then the file would be deleted. All files starting with "~" would be ignored as these are being used by the logger.

3.4.3. Analysis of the uploading script

Overall the implementation of the script was a success with the most difficult part being connection to the VPN. Compared to the design, the main functionality remained the same but additional functions came into place that allowed for a more robust script. This included the function to check for a table that already existed on the database and creating and checking for a VPN connection.

3.4.4. Creating Cron jobs for the upload function

To automatically run the script on the RPI another cron job was enabled in the crontab configuration file. The script was to run every two minutes; if no connection were given then the script would exit. This was done by appending the line "**/2 * * * * runUpload.sh*" onto the end of the configuration file. The shell script that the cron job would execute would run the python script and once it had finished would kill all processed with an OpenVPN instance. This was to prevent multiple OpenVPN instances being created.

Now that the data could be successfully logged and uploaded to the database, the visualisation tool could begin development.

3.5. Visualisation

3.5.1. Finding a suitable plugin

The first task to start the visualisation application was to find a plugin for OpenStreetMap that would allow for heat maps. The OpenStreetMap recommended the use of a library called “Leaflet” that could be used when deploying a slippy map to a webpage. A slippy map is an interactive map that can be easily deployed onto a web page. Leaflet is a JavaScript library that uses OpenStreetMap for its visual map; Leaflet can easily be extended with the use of plugins. These plugins are easily available on the Leaflet website.

Multiple heat maps plugins were available, one plugin was very easy to implement and had examples showing its capabilities. This was called “Leaflet.heat” and was created by Vladimir Agafonkin, the creator of the Leaflet library.

3.5.2. Implementing Leaflet.heat

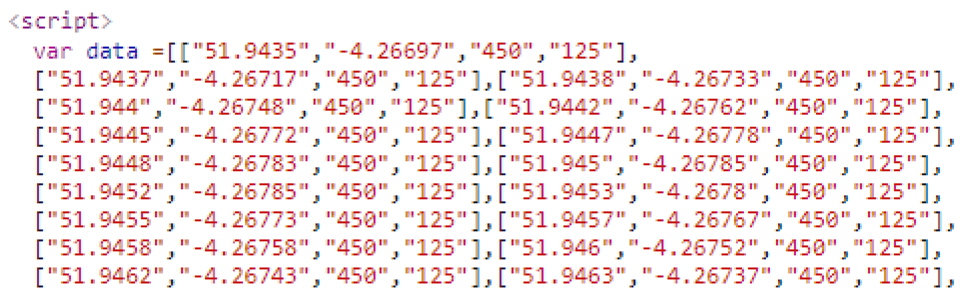
The leaflet files were downloaded, and this contained the CSS, JavaScript and map files. These needed to be referenced in the HTML file to use the interactive map. Next the heat map plugin was downloaded along with one example. The example was stripped down to its bare minimum in order to show an interactive map without any other elements on the page and the heat map not referenced (see Appendix I). When opening up the web application the “*`setView()`*” function was used to set the opening location to Aberystwyth. HTML Geolocation could have been used to get the users current location but the user would need to agree to this, during development this was kept to the latitude and longitude of Aberystwyth instead.

3.5.3. Retrieving data from the MySQL server

The next stage was to retrieve the data from the MySQL server. A PHP script was needed to access the server and request the information. At this point it was realised that the HTML files would need to be PHP files if a PHP script was needed, HTML can’t run PHP but it works the other way around. The PHP was to be executed on the server and just allow the user access to the output. The page created was now called “index.php” (“map.html” in the design) and was hosted on the university network.

An additional script was created in the public_html directory called “*`getData.php`*” and was referenced in the index using “*`<?php include  'getData.php';?>`*”, with the intention of returning the data from the MySQL server.

Retrieving the data from the server was very straight forward and similar to the method used for the RPI upload script. A connection PHP data object (PDO) was created and then a statement was issued to the connection. The statement that is used is “*`SELECT latitude,longitude,CO2,TOC FROM MMP;`*” which would retrieve only the data that was needed. Using echo commands within the PHP script would allow the script to output JavaScript that would be sent to the user (see Appendix J). The data was needed to be in the format of a 2D JavaScript array with no labels (only numbers), this is why the “*`fetchAll(PDO::FETCH_NUM)`*” was called. To translate this in to a JavaScript 2D array from PHP, a JSON encoder was used and it worked as it should (see Appendix J).



```
<script>
  var data =[["51.9435","-4.26697","450","125"],
  ["51.9437","-4.26717","450","125"],["51.9438","-4.26733","450","125"],
  ["51.944","-4.26748","450","125"],["51.9442","-4.26762","450","125"],
  ["51.9445","-4.26772","450","125"],["51.9447","-4.26778","450","125"],
  ["51.9448","-4.26783","450","125"],["51.945","-4.26785","450","125"],
  ["51.9452","-4.26785","450","125"],["51.9453","-4.2678","450","125"],
  ["51.9455","-4.26773","450","125"],["51.9457","-4.26767","450","125"],
  ["51.9458","-4.26758","450","125"],["51.946","-4.26752","450","125"],
  ["51.9462","-4.26743","450","125"],["51.9463","-4.26737","450","125"],
```

Figure 28 - Data being passed back to index.php

When inspecting the page source through a web browser the results of the getData.php page would be viewable where the PHP script had been called (Figure 28). This meant the server was executing the PHP file and returning the output to the user, this is exactly how it was designed.

3.5.4. Selecting data to view

Each array within the array had a length of 4, but this needed to be a length of 3 to work with the heat map plugin, in the format of latitude, longitude and intensity. The current format was latitude, longitude, CO2 value and TVOC value. Either CO2 or TVOC needed to be removed.

To allow the user to choose between viewing the CO2 or the TVOC readings radio buttons were created. These radio buttons would call a function on changing value that would splice the original array to ensure each array within the 2D array was a length of 3 (see Appendix K for the *getCO2orTOC()* array splicing function). The splice was used to change the data to only contain latitude, longitude and the value the user had selected. On loading the webpage the default radio button to be checked was the CO2 radio button.

3.5.5. Viewing the data

To add the data to the heat map a heat layer needed to be created. The heat map plugin needed to be imported, this was imported using "*<script src="./dist/leaflet-heat.js"></script>*". The file leaflet-heat.js was the heat map plugin. Once the plugin was included the heat layer could be added to the leaflet map. This was done using the same method as was shown in the heat map example "*var heat = L.heatLayer(data,{}).addTo(map);*". "Data" is the spliced array. When the data changed, using the radio buttons, the heat map would need to be removed and redrawn (see Appendix K for the *changeData()* function to redraw the heat layer).

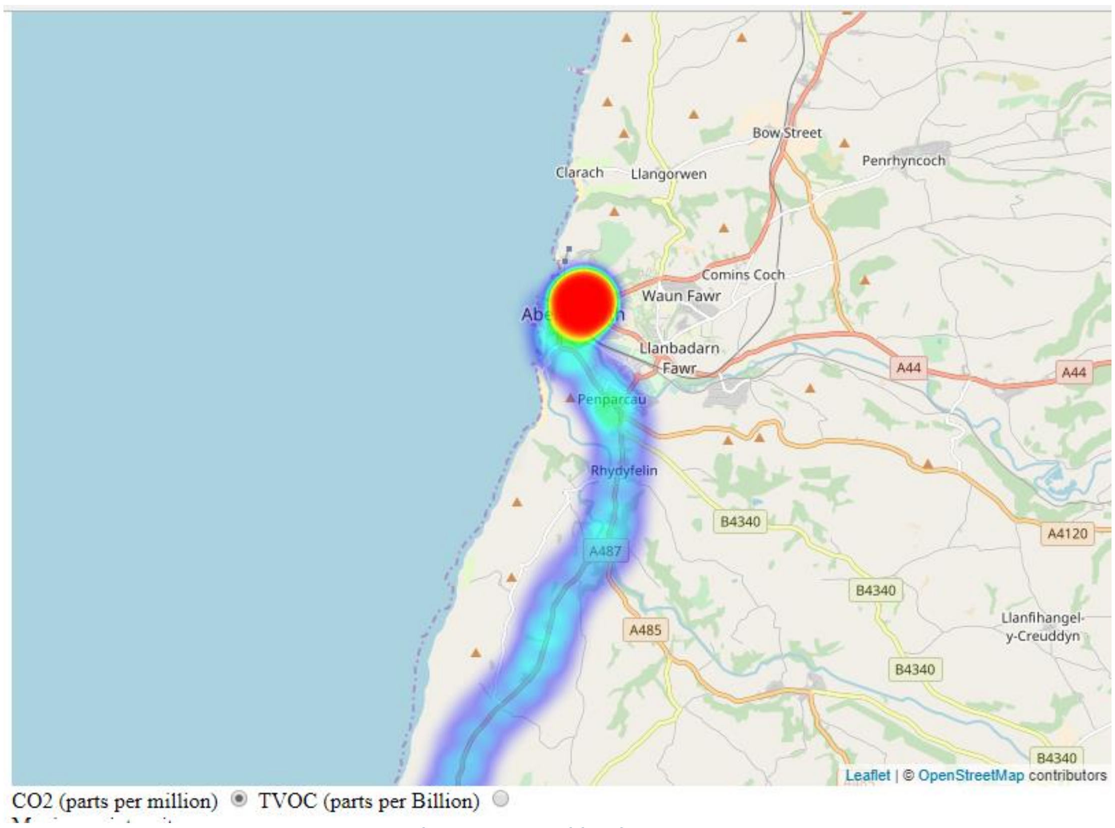


Figure 29 - working heatmap

The functionality of the heat map was complete, the user could select between the CO2 and TVOC radio buttons and the data would change on the Leaflet Map (see Figure 29). Sliders were added to change option in the heat layer using "*`.setOptions()`*". The function would read the current value of the slider and use those in the set options function (see Appendix K for the *`changeData()`* function to set the heat layer options).

3.5.6. Interactive tools

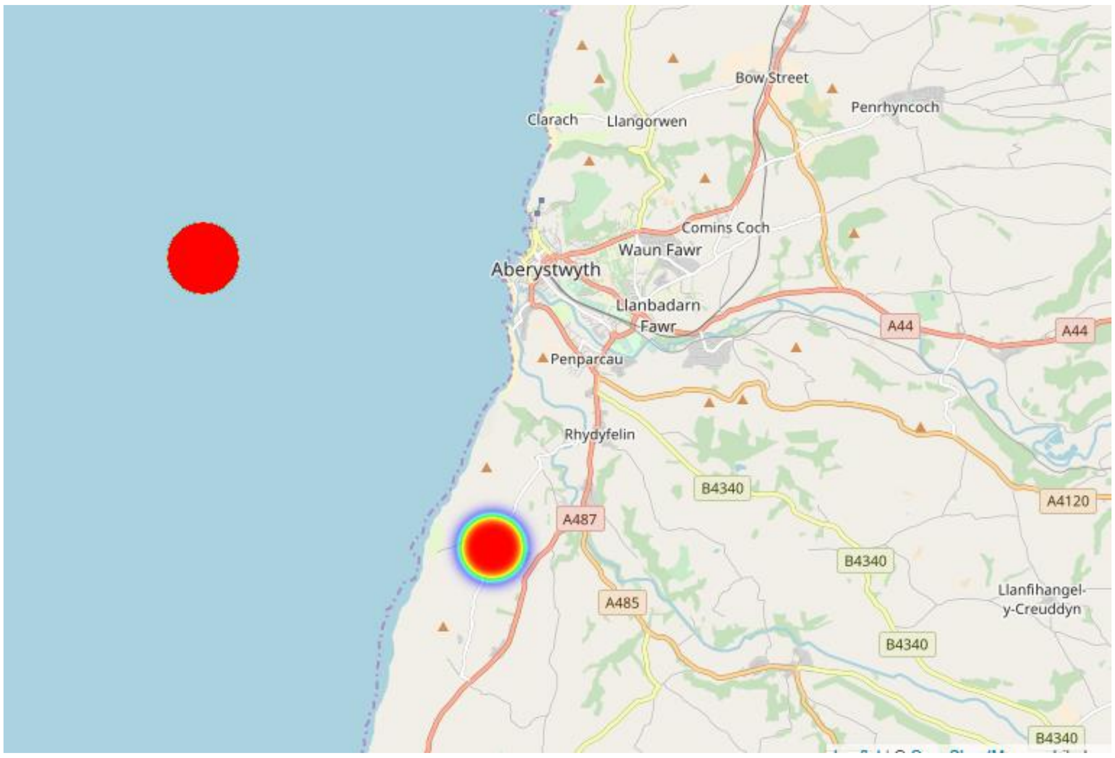


Figure 30 - bugged options

When attempting to change the options of the heat layer, many of the options would not change or cause the heat layer to change massively (see Figure 30). This later turned out to be a bug in the leaflet.heat plugin with many options causing the functionality not to work. The two working options were maximum value and minimum opacity value which changed upon moving the sliders. The sliders were implemented but the functionality was removed.

While researching how to get co-ordinates from clicking the Leaflet map a question was found on StackOverflow of how to find the closest location to a set of latitude and longitude co-ordinates using geolocation [7]. This was very similar to the overall functionality that was trying to be implemented. The functionality was changed from using the geolocation to the location retrieved from clicking on the map. These functions were then further adapted to provide the closest reading to the users click on the map. "No data" would appear if no data had been provided in that area.

3.5.7. GUI for webpage

A navigational menu bar was added using a tutorial from W3schools [8], the main part of this was the CSS. This was when the custom CSS file was made alongside the CSS for the Leaflet application. Additional pages were created once the navigation bar was working. This included an about page with a short explanation of the website and a source page, where anyone could download the source (credentials to the MySQL server had been removed).

Small changes were made to the layout using the CSS. Unfortunately, it was difficult to set up the layout that was planned from the design, though the functionality remains the same.

The footer was created for each webpage this included the information services required statement when hosting pages. It also included two validators used for checking the HTML and CSS were in valid format.

3.5.8. Problems found with Leaflet.heat

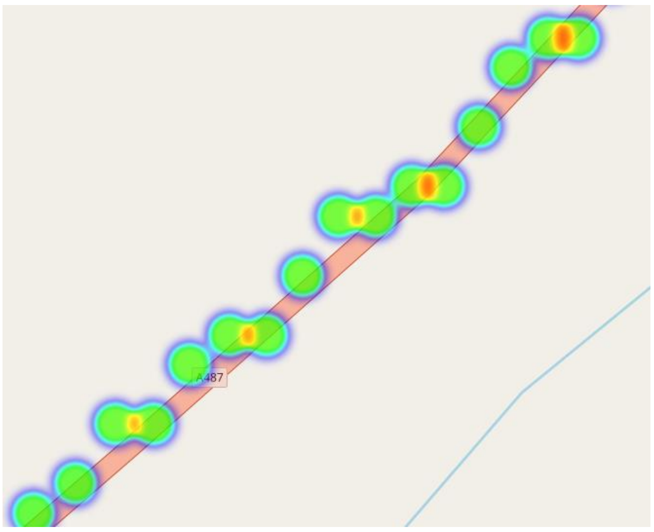
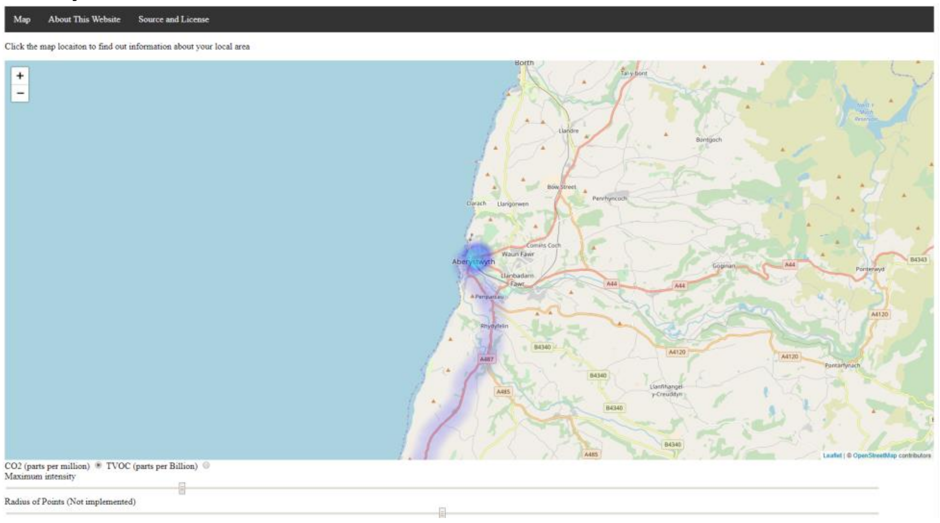


Figure 31 - Heatmap bug

When viewing the heatmap in greater detail it became obvious that the heatmap wasn't using the intensity value (CO2 or TVOC) correctly. Rather than using the value it was passed, it would create a heatmap depending on how many points were in that area (Figure 31). After researching on the GitHub page for the heatmap plugin, many users had been complaining about the frequent bugs that occurred throughout the plugin, including bugs that were found earlier in the project. Thankfully since this plugin is opensource, other users had corrected the bug, it just needed rebuilding using npm. Npm is a package manage for Javascipt.
Using the "package.json" and using the new source from the user on Github the new leaf-heat.js file could be built. This was created using the command "`npm install && npm run prepublish`". When viewing the points, they now showed an intensity corresponding to the value, which is correct. The settings of the data were still bugged though (as shown in Figure 30) with users complaining on the GitHub about it not working.

3.5.9. Analysis of Leaflet.heat



Click the map locaiton to find out information about your local area

CO2 (parts per million) [✓ ●] TVOC (parts per Billion) ○
Maximum intensity
Radius of Points (Not implemented)

Overall the visualisation deviated slightly from the software design with an additional webpage being implemented for a source download (though this was not necessary). The user interface changed from the design due to a lack of knowledge with CSS. The maps intended functionality is implemented but the additional tools are not. The information regarding the current pollution levels is not implemented though this could be as simple as a lookup table outputting the effects of the pollution levels. The interactive tools to change the data do not work correctly as there are limitations with the plugin used with Leaflet.

4. Testing

4.1. Overall Approach to Testing

Testing didn’t play a major part in the approach to this project. Most testing was completed as the project was developing using manual continuous testing. This meant that the functionality of functions were tested by hand as they were being developed.

Small unit tests were created for the python scripts that were running on the RPI. It was difficult to create tests that are tests themselves (e.g. a test for internet connection).

The visualisation website undertook user interface testing, this meant using the interactive tools and checking that they work correctly. This was compared to the design that was created.

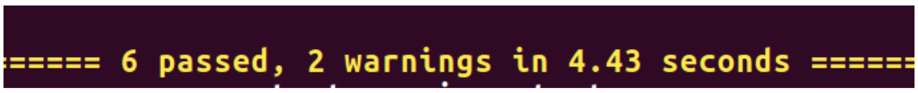
4.2. Unit Tests

Unit tests are created to test the internal functionality of single functions. These were used on the python files that are on the RPI.

4.2.1. Unit Tests – Data Logger

As unit testing was started once the project had been complete, many of the unit tests rely on other functions to work, an example of this is test DL3 requires communication to have started with the I2C device, the `set_up_i2c()` function was called in order to receive a message from the device.

Test Id	Test Description	Expected Result	Result
DL1	Create file for logging	The function should create a file using the a current timestamp	Pass
DL2	Check previous files	A file is created with the character '~' at the start of the filename. This is then run through check previous files and will remove the first character	Pass
DL3	Testing the I2C functions	A message should be received and the CO2 and TVOC values will be converted to equal the calculated values	Pass
DL4	Testing the Serial functions	A message should be received when and only when the "position_fix_indicator" within the dictionary is equal to 1	Pass
DL5	Testing the conversion of GPS coordinates from degrees minutes seconds to degrees	A change in format	Pass
DL6	Log a message from the GPS and air quality sensor	A file created with 2 lines – headings and a line of data	Pass



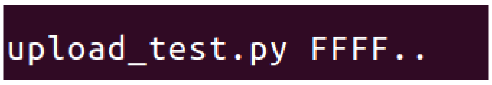
```
====== 6 passed, 2 warnings in 4.43 seconds ======
```

Figure 32 - All data logger unit tests passing

The data logger unit tests do produce warnings (see Figure 32) this is due to the tests creating a new instance of serial connection. This warning states that the instance has already been created but it's continuing anyway.

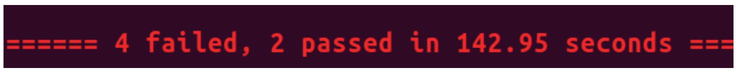
4.2.2. Unit Tests – Upload Script

Test Id	Test Description	Expected Result	Result (pass/fail)
UP1	Test whether a live connection has been made with the server using PyMySQL	Connection should be open, the VPN will need enabling	Pass
UP2	Create test table in the database, the vpn and internet connection will be necessary. At the end of the test the table will be removed and checked whether table exists again.	Table will be created and then removed	Pass
UP3	Connected() is a test in itself. Test case UP4 ensures it fails when there is no connection	An internet connection will be a pass	Pass
UP4	Run the VPN and test that it’s running	VPN will run successfully	Pass
UP5	Test will only pass when there is no internet connection (causing other tests to fail). Opposite of UP3. This test will be run with no internet connection.	All other tests will fail apart from UP5 and UP6	Pass
UP6	Test will only pass when there is no access to the VPN (causing other tests to fail) oppoite of UP4. This test will be run with no internet connection.	All other tests will fail apart from UP5 and UP6	Pass



```
upload_test.py FFFF..
```

Figure 33 - Tests failing when internet connection is removed



```
====== 4 failed, 2 passed in 142.95 seconds ===
```

Figure 34 - Tests failing

Figure 33 shows the first four tests failing when the internet connection is removed, this is because they need an internet connection for completion. The last two tests passed as they were created to test when there is no internet connection.

4.3. Manual Testing

Manual testing involved going through a process manually to ensure functions and configurations functionality was correct.

4.3.1. RPI Security Testing

Testing the security of the RPI was important as credentials to the Aberystwyth VPN and MySQL server were stored in the /et/c directory and Pi user directory. This involved one simple test when trying to access the RPI.

Test Id	Test Description	Expected Result	Result (pass/fail)
Man1	Check a password is requested when accessing the pi user	Password request enter on correct password	Pass

4.3.2. Data logger

Test Id	Test Description	Expected Result	Result (pass/fail)
Man2	Unplug GPS before logger starts	Software will remain in one state, not logging	Pass
Man3	Unplug air quality sensor before logger starts	Software will remain in one state, not logging	Pass
Man4	Log file size	Log files are no more than 25kb	Pass
Man5	Cron job starts the data on startup of the RPI	Cron job will start the python script and start logging	Pass/Fail this works sometimes. The cron job will start the script 100% of the time. But the script sometimes remains in a state (GPS fix) on reboot this usually fixes problem. May be a hardware error due to the

			number of requests.
Man6	LED flash 2 times when trying to get GPS fix	LED should flash 2 times	Pass
Man7	LED flash 3 times when waiting for air quality sensor	LED should flash 3 times	Pass
Man8	LED should continuously flash when logging	LED flashes on and off continuously.	Pass

4.3.3. Upload script

Test Id	Test Description	Expected Result	Result (pass/fail)
Man9	Cron job executes script every 2 minutes – monitor using watch command	Cron job is executed	Pass
Man10	Files are deleted after being uploaded	Logs file empty	Pass

4.3.4. Visualisation Application Testing

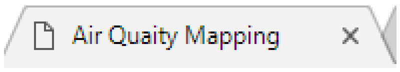
These tests were completed using

Test Id	Test Description	Expected Result	Result (pass/fail)
Man11	Data is passed from PHP script to webpage (not showing user credentials). This is a security related test.	Variable is created within script	Pass
Man12	HTML5 validates using W3 validator	No errors in HTML	Pass
Man13	CSS validates using CSS validator	No errors in CSS	Fail – Due to errors from Leaflet CSS. Created CSS has no errors though
Man14	All HTML5 are within appropriate tags	Appropriate tags	Pass

4.3.5. User Interface Testing

Manually testing user interface ensured functionality worked on GUI elements. These have been based off of requirements that were set out in the design.

Test Id	Test Description	Expected Result	Result (pass/fail)
UI1	Menu bar on the index page is visually there.	Menu bar on the index page is visually there.	Pass
UI2	Menu bar on the about page is visually there	Menu bar on the about page is visually there	Pass
UI3	Menu bar on the source page is visually there	Menu bar on the source page is visually there	Pass
UI4	Map webpage is visually similar to Figure 20	Similar layout to Figure 20	Fail
UI5	About page is visually similar to Figure 21	About page is visually similar to Figure 21	Pass
UI6	Map is shown on the homepage (index.html)	Homepage has map displayed	Pass
UI7	Values can be changed between CO2 and TVOC	Radio buttons change values on radio button change	Pass
UI8	Heatmap is visible	The heat map can be seen on the home page	Pass
UI9	Interactive slider “maximum intensity” changes visible data	Labelled slider is visible	Pass
UI10	Interactive slider “Radius of Points” changes visible data	Labelled slider is visible	Fail
UI11	Interactive slider “Blur of Points” changes visible data	Labelled slider is visible	Fail
UI12	Interactive slider “Minimum Opacity of Points” changes visible data	Labelled slider is visible	Pass

UI13	Heat map shows intensity of values	Heat map is displayed with intensity effecting the outcome	Pass/Fail – two scripts have been used (one built from a GitHub solution) these can be switched between. One shows the intensity by using the number of points, the other shows the intensity using the data (this is the correct way)
U14	All links on the menu navigate to the correct page	When a page is clicked the browser will change to that page	Pass
U15	Map can be moved by clicking and dragging	Map will move	Pass
U16	Information service statement about contents of page displayed on all pages	Statement at footer of page	Pass
U17	W3 validators for CSS and HTML both work on all pages	Both HTML5 and CSS are valid	Pass
U18	Map shows contributors (Leaflet and OSM)	Contributors are shown	Pass
U19	Window Title is "Air Quality Mapping" on all pages	Window Title is "Air Quality Mapping" on all pages	Pass  Air Quaity Mapping
U20	About page contains information about site with links to plugins	About page contains information about site with links to plugins	Pass
U20	Source page contains license details and link to download the source	Source page contains license details and link to download the source	Pass

4.4. Known Bugs

No bugs have been recorded from the upload script. The other two parts of the project have known bugs.

4.4.1. Data logger

The cron job used to start the data logger on start-up works, but when the script is waiting for the GPS to retrieve a position fix it will stay in this state indefinitely. Killing the process and then starting it again will have a GPS fix almost instantly, the script will then continue to the logging stage.

4.4.2. Visualisation Index.html page

When using the original Leaflet-heat.js script located in the /dist/ folder the shown intensity is not a rolling average of the points around it but rather the total sum of all intensities. When using this script the “radius of points” and the “blur” sliders do not affect the data. This is due to the values of the sliders being removed from the function to change the settings. When they were implemented to change the settings it would cause unpredictable changes to the viewable data.

When this is changed to the newly built Leaflet-heat1.js script located in the /dist/ folder the shown intensity is correct. The sliders to change the “radius of points” and the “blur” still do not work. When changing the radio buttons, the viewable data does not change, but upon clicking the map on data the values produced in the information section show a change. Using this script file does not allow the “maximum intensity” to change. The only slider that works when using the Leaflet-heat1.js is the “minimum opacity of points”.

5. Critical Evaluation

5.1. Requirements and Design

The requirements of the project were satisfied to some degree, the monitoring system did what it was designed to do occasionally. The only major downfall was that the GPS didn't return readings with a positional fix on occasions. The requirements for uploading the data were satisfied, data can be easily uploaded as the script is run every 2 minutes. The visualisation web application provides users with a map overlaid with a heat map, as per the requirements and this works very well. The functionality of the interactive were very temperamental, this was due to the Leaflet.heat library having bugs that were posted on the GitHub page, and these were overseen. The design of the web page was different to the requirements, this was down to a lack of knowledge of CSS. Overall the design and the requirements were satisfied but had been expanded on for implementation reasons.

5.2. Process

The process of the project worked very well. An adapted Kanban approach worked in some cases but at times the process was pushed aside while development was prioritised over management. Sprints weren't used as they play a bigger role when used in teams, individually it was thought that sprints wouldn't work. It would have been nice to attempt sprints rather than just the work being completed when it was completed with no timeframe. The use of GitHub and Trello were definitely beneficial for the process of this project. The two Gantt charts that were created were helpful for estimating the amount of time remaining to complete tasks, but sometimes wasn't realistic. The tasks could have been narrowed into subtasks to help estimate the work.

5.3. Starting the project again

If this project were to be started again then time management would play a very important role. The time management was not handled very well with work being started late and other university modules taking priority. An object oriented approach would be taken for the RPI python files rather than done procedurally. This would allow for more design and experience in software engineering.

Gathering more data would allow for a greater range of results. This would involve allowing users to use the monitoring device on car journeys. This could of involved further testing to improve the performance with a high number of data points involved.

5.4. Future Work

The work that could be implemented in the future would be to be part of the open source community and fix the problems with Leaflet.heat. This would involve having a greater understanding of npm and JavaScript.

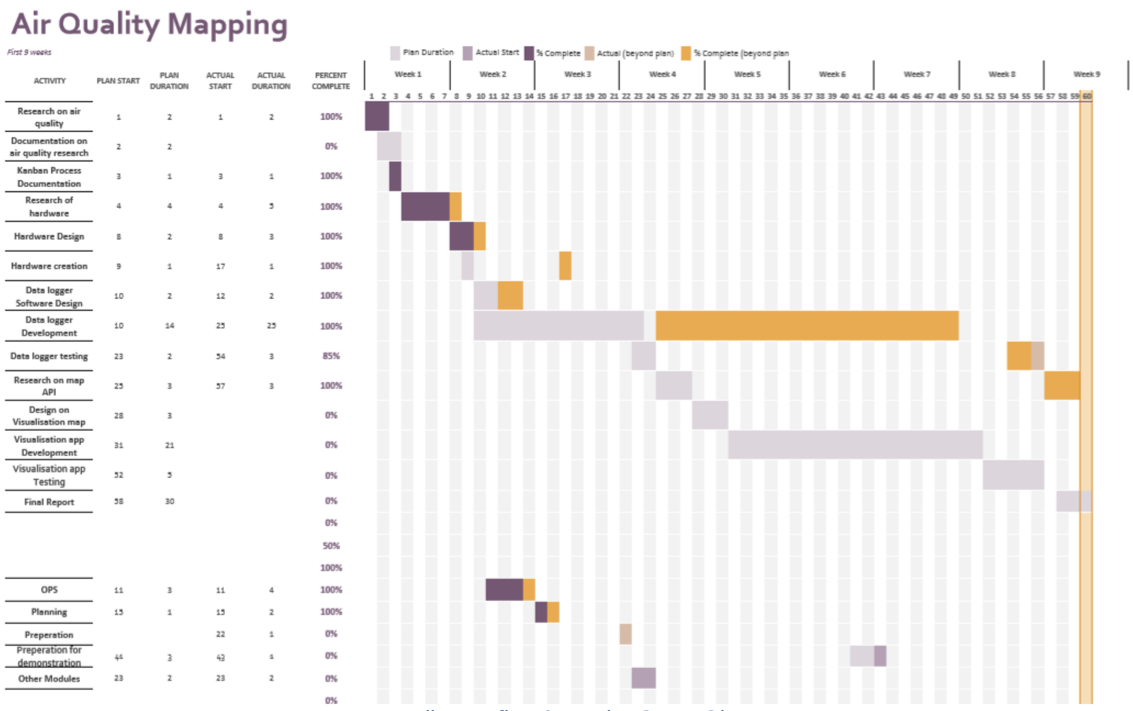
Collecting more data would be beneficial to the web application, rather than just one route being displayed.

5.5. Overall

The project was aimed to be something unique and different. After research of similar products it seemed that other people had tried to tackle the problem in different ways. The method of approaching the problem that this project took could be a success if more drivers were involved. The end result, the map, shows a clear trail of where the TVOC/CO2 values are at the highest (Aberystwyth and Swansea). More test data could have shown the potential of what the project was aimed to be with air quality levels shown across the UK.

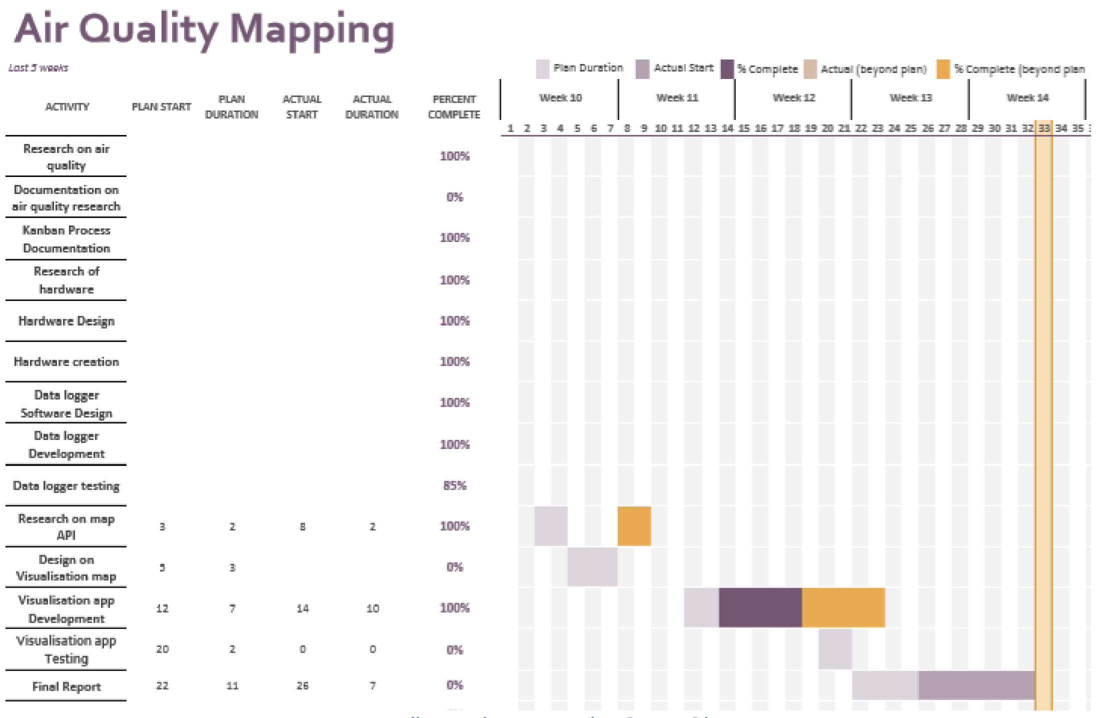
1 of the 14 steps that the royal college of physicians [1] mentioned in the report was to monitor pollution in an effective way and educate the public proactively. If this project was developed to its full potential it could possibly satisfy this need.
For a proof of concept this project has successfully proven that it is viable for full implementation with improved sensors and visualisation tool.

6. Appendices



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Research on air quality	1	2	1	2	100%
Documentation on air quality research	2	2			0%
Kanban Process Documentation	3	1	3	1	100%
Research of hardware	4	4	4	5	100%
Hardware Design	8	2	8	3	100%
Hardware creation	9	1	17	1	100%
Data logger Software Design	10	2	12	2	100%
Data logger Development	10	14	25	25	100%
Data logger testing	23	2	54	3	85%
Research on map API	25	3	57	3	100%
Design on Visualisation map	28	3			0%
Visualisation app Development	31	21			0%
Visualisation app Testing	52	5			0%
Final Report	58	30			0%
					0%
					50%
					100%
OPS	11	3	11	4	100%
Planning	15	1	15	2	100%
Preperation			22	1	0%
Preperation for demonstration	44	3	43	1	0%
Other Modules	23	2	23	2	0%
					0%

Appendix A – first 9 weeks Gantt Chart



ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Research on air quality					100%
Documentation on air quality research					0%
Kanban Process Documentation					100%
Research of hardware					100%
Hardware Design					100%
Hardware creation					100%
Data logger Software Design					100%
Data logger Development					100%
Data logger testing					85%
Research on map API	3	2	8	2	100%
Design on Visualisation map	5	3			0%
Visualisation app Development	12	7	14	10	100%
Visualisation app Testing	20	2	0	0	0%
Final Report	22	11	26	7	0%

Appendix B – last 5 weeks Gantt Chart

Appendix C – Chosen air quality sensor

Appendix D- chosen GPS

```
Timestamp,Latitude,Longitude,pos_fix,CO2,TOC
1234,56.52,254.2,1,255,655
1235,56.52,254.2,1,256,664
1236,56.52,254.2,1,255,655
1237,56.52,254.2,1,256,664
```

Appendix E – example of log file design

```
[pishare]
comment = pi dir
path = /home/pi
browseable = yes
writeable = yes
only guest = no
create mask = 0777
directory mask = 0777
public = no
```

Appendix F - Samba file example

A. Third-Party Code and Libraries

Leaflet – An open source library that uses OpenStreetMap to allow for easy developing of map plugins and the use of slippy maps on webpages. This has been used to create the map tile on the index.html webpage [9]. Covered by the BSD-2 License.

Leaflet.heat – An open source plugin for Leaflet that is created by the Leaflet creator. Allows for the use of heat maps through simple commands, has been implemented on the index.html webpage [10]. Covered by the BSD-2 License.

Pigpio – Used to communicate with the RPI hardware through python. This was used with the I2C interface when a reading could not be completed when using SMBus [11]. Covered by the MIT license.

Pytest - A small library for easily testing python code. Unit tests were created using this library to test a small amount of the functionality of the data logger and the upload scripts [12]. Covered by the MIT license.

Pymysql - A Python MySQL database API. The library is used to create a connection to a MySQL server [13]. Covered by the MIT license.

Third-Party Code
Code used to find nearest point of data of where the user clicked [8]. Has been adapted to suit the needs of the project.

```
// Get User's Coordinate from their Browser
window.onload = function() {
  // HTML5/W3C Geolocation
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(UserLocation);
  }
  // Default to Washington, DC
  else
    NearestCity(38.8951, -77.0367);
}

// Callback function for asynchronous call to HTML5
geolocation
function UserLocation(position) {
  NearestCity(position.coords.latitude,
position.coords.longitude);
}



// Convert Degress to Radians
function Deg2Rad(deg) {
  return deg * Math.PI / 180;
}

function PythagorasEquirectangular(lat1, lon1, lat2, lon2) {
  lat1 = Deg2Rad(lat1);
  lat2 = Deg2Rad(lat2);
  lon1 = Deg2Rad(lon1);
  lon2 = Deg2Rad(lon2);
  var R = 6371; // km
  var x = (lon2 - lon1) * Math.cos((lat1 + lat2) / 2);
  var y = (lat2 - lat1);
  var d = Math.sqrt(x * x + y * y) * R;
  return d;
}

var lat = 20; // user's latitude
var lon = 40; // user's longitude

var cities = [
  ["city1", 10, 50, "blah"],
  ["city2", 40, 60, "blah"],
  ["city3", 25, 10, "blah"],
  ["city4", 5, 80, "blah"]
];

function NearestCity(latitude, longitude) {
  var mindif = 99999;
  var closest;

  for (index = 0; index < cities.length; ++index) {
    var dif = PythagorasEquirectangular(latitude, longitude,
cities[index][1], cities[index][2]);
    if (dif < mindif) {
      closest = index;
      mindif = dif;
    }
  }

  // echo the nearest city
  alert(cities[closest]);
}
```

B. Ethics Submission

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
rdm10@aber.ac.uk

Full Name
Robert David Mouncer

Please enter the name of the person responsible for reviewing your assessment.
Neal Snooke

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
nns@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
Air Quality Mapping

Proposed Start Date
29th January 2018

Proposed Completion Date
4th May 2018

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
A monitoring and visualisation system for air quality. A raspberry pi designed and made to use in vehicles that logs the current GPS location and air quality. An online web page showing a visualisation of the data collected.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here: If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

C. Code Samples

```
message = serial.readline().decode('utf-
8').replace('\r\n','')
gpgga_message = dict(zip(["message_ID",
                          "timestamp",
                          "latitude",
                          "ns_indicator",
                          "longitude",
                          "ew_indicator",
                          "position_fix_indicator",
                          "satellites_used",
                          "HDOP",
                          "msl_altitude",
                          "units",
                          "geoal_seperation",
                          "units",
                          "age_of_diff_corr",
                          "checksum"],message.split(",")))
```

Appendix G - reading from serial

```
import smbus

BUS=smbus.SMBus(1)

ADDRESS = 0x5a
print(BUS.read_byte_data(ADDRESS,0))
```

Appendix H - Requesting data using SMBus

```
<!DOCTYPE html>
<html lang="en">
      <head>
            <title>test</title>
            <link rel="stylesheet" href="leaflet.css" />
            <script src="leaflet.js"></script>
            <style>
                        #map { width: 800px; height: 600px; }

            </style>
      </head>
      <body>
            <div id="map"></div>

            <script>
                  var map = L.map('map').setView([52.4147, -
                  4.0842], 12);
                  var tiles =
      L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png',
{
                              attribution: '&copy; <a
      href="http://osm.org/copyright">OpenStreetMap</a>
                  contributors',}).addTo(map);
            </script>
      </body>
</html>
```

Appendix I – HTML to produce Leaflet interactive map

```
<?php
ini_set("display_errors", 1);

$server = "db.dcs.aber.ac.uk";
$user = "rdm10";
//password not included
$pwd = "";
$db = "rdm10";
//create connection
$connection = new PDO("mysql:host=$server;dbname=$db",
$user,$pwd);
$sqlQuery = "SELECT latitude,longitude,CO2,TOC FROM MMP;";
//execute query

$result=$connection->query($sqlQuery)-
>fetchAll(PDO::FETCH_NUM);
//return result as 2d Javascript Array
echo "<script>";
echo "var data =". json_encode($result). ";";
echo "var databack = data.map(function(arr) {return
arr.slice();});";
echo "</script>";
?>
```

Appendix J - getData.php

```
function changeData(){
      //change array depending on radio buttons
      getCO2orTOC();
      //get values from sliders
      var max_value =
document.getElementById("maximum_range_id").value;
      var radius_value =
document.getElementById("radius_id").value;
      var blur_value =
document.getElementById("blur_id").value;
      var minOpacity_value =
document.getElementById("minOpacity_id").value;

      heat.remove();
      heat=L.heatLayer(data,{}).addTo(map);
      heat.setOptions({max:max_value,
minOpacity:minOpacity_value});

}
function getCO2orTOC(){
      //restore data back to the original dataset
      data = databack.map(function(arr) {return
arr.slice();});
      //get the current value of the radio button
      var radioButton = document.querySelector('input[name =
      "data_type"]:checked').value;
      //splice the data depending on value of the radio
buttons
      if (radioButton == 'TOC'){
            for( const array of data){
                  array.splice(2, 1);
            }

      }else{
            for( const array of data){
                  array.splice(3, 1);
            }

      }
}
```

Appendix K - functions called on radio button change

Annotated Bibliography

[1] – Royal College of Physicians. (2016, February 23). *Every breath we take: the lifelong impact of air pollution.* Retrieved from www.rcplondon.ac.uk: https://www.rcplondon.ac.uk/projects/outputs/every-breath-we-take-lifelong-impact-air-pollution

This report is aimed to educate what the effects of bad air quality can have on our bodies over a lifetime. It states what needs to be done to combat air pollution and one step is to “monitor air pollution effectively” and to use the results to communicate “proactively to the public”.

[2] – Department of Transport. (2017, January). *Vehicle licensing statistics: January to March 2017.* Retrieved from www.gov.uk: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/620223/vehicle-licensing-january-to-march-2017.pdf

This reports aim is to educate the public about the current vehicle licensing statistics between the months of January and March in the year 2017.

[3] – Plume Labs. (2018, 04 27). *Pigeon Air Patrol*. Retrieved from http://www.pigeonairpatrol.com/

A method of measuring air quality using pigeons with sensors in London.

[4] – MikroElektronika. (2018, 04 30). *Air quality 2 click schematic v100.* Retrieved from rs components: https://docs-emea.rs-online.com/webdocs/159a/0900766b8159ab32.pdf

Schematic showing the connections that can be made to the air quality sensor

[5] – *smbus/I2C sensor returns fixed data.* (2018, 04 30). Retrieved from Stack exchange: https://raspberrypi.stackexchange.com/questions/79091/smbus-i2c-sensor-returns-fixed-data

A question asked and answered that helped provide a solution to a problem. This introduced the pigpio library into the project.

[6] - *Strange return from pigpio i2c_read_device*. (2018, 04 30). Retrieved from raspberrypi stackexchange: https://raspberrypi.stackexchange.com/questions/80207/strange-return-from-pigpio-i2c-read-device

A question created that the author of pigpio responded relating to the output from I2C devices. Printable characters in a byte array will appear as ASCII characters.

[7] – (2018, 05 01). Retrieved from StackOverflow: https://stackoverflow.com/questions/21279559/geolocation-closest-locationlat-long-from-my-position

Code that had been adapted to calculate the closest data point from a click position. The original idea was the same but was used to calculate the closest city.

[8] – *CSS Navigation Bar*. (2018, 05 02). Retrieved from w3Schools: https://www.w3schools.com/css/css_navbar.asp

A tutorial used to implement the navigational bar at the top of the visualisation tool web pages.

[9] – Leaflet. (2018, 05 03). *Leaflet*. Retrieved from an open-source JavaScript library: https://leafletjs.com/

A library used that extended the use of OpenStreetMap to allow development for JavaScript plugins.

[10] – *Leaflet.heat*. (2018, 05 03). Retrieved from github.com: https://github.com/Leaflet/Leaflet.heat

A heat map plugin for Leaflet.

[11] – *pigpio*. (2018, 05 03). Retrieved from abyz: http://abyz.me.uk/rpi/pigpio/

Library used with Python to gather readings from the I2C interface. It is a library that allows control of GPIO.

[12] – *pytest*. (2018, 05 03). Retrieved from pytest.org: https://docs.pytest.org/en/latest/

A small library for testing python functionality. This was used to create unit tests for the data logger and upload scripts.

[13] – *PyMySQL*. (2018, 05 03). Retrieved from github.com: https://github.com/PyMySQL/PyMySQL

An API able to communicate with MySQL servers using python commands.