Internet of things air quality monitoring system

Adam O’Brien

Final-Year Project– BSc in Computer Science
Supervisor: Prof. Dirk Pesch

School of Computer Science and Information Technology University College Cork

April 2021

Abstract:

Air pollution leads to around 1,300 yearly deaths in Ireland and over 150 million globally. Environmental air parameters affect our quality of life, changing daily even hourly. Over the past few decades there has been a rapid increase in industrialisation, followed by the want and need for people to monitor their local air.

Particulate Matter (PM) is considered one of the most dangerous air pollutants. PM could originate from natural sources such as volcanic ash or desert dust particles, but the main source of PM comes from the human impact on the environment such as industrial processes, transport and agriculture.

This report presents a cost effective air quality measurement system using the Raspberry Pi 3 (RPI3) Single Board Computer (SBC) and a PMS5003 air quality sensor. This device will allow people to measure the Air Pollution Index (API) in any location they wish. The main pollutants focused on are PM1, PM2.5 and PM10.

Six nodes were created and placed outside in various locations around Cork City. Each node consists of a PM sensor, a SBC and a power supply. The nodes are encapsulated in a weatherproof box and relay data using secure transfer protocols to a cloud storage device. The readings are then queried from the database and live measurements are graphed and visualised on a map of Cork City. Graphing each sensor 24 hours at time will display peak times of high particulate matter count in each area.

Keywords: *Sensor *Air pollution *Single Board Computer *PM *Internet of things

Declaration of Originality:
In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

- This is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- With respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- With respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: Adam O'Brien.

Date: 5/4/2021

1. Introduction

Air pollution is ranked fourth worldwide in risk factor for death[1]. People nowadays need to become more aware of the air quality around them and know whether the air they're breathing is clean or polluted. The extreme health effects of PM (Particulate Matter) are well documented. Exposure to PM is mainly associated with respiratory and cardiovascular disease[2]. The International Agency for Research on Cancer (IARC) classified PM as carcinogenic to humans[3].

Particulate matter is very small particles which can be solid or liquid. Some of these particles occur naturally, and many are man made. Particulate matter is usually referred to as PM with a number after it to show how small the PM is. The EPA monitors PM1.0, PM2.5, PM10.0 and compares levels to limit values in the CAFE (Clean Air for Europe) Directive and WHO guidelines. For example Pm10.0 means the particulate matter is 10 microns or less in diameter, small enough to lay 10 of these particles across the width of an average human hair. PM2.5 signifies that it is particulate matter of 2.5 microns or less in diameter – 40 of these particles could be laid across the width of an average human hair[6].

Different countries have different quality indices for example, United States and China uses Air Quality Index (AQI), Canada and Hong Kong uses Air Quality Health Index (AQHI), Singapore uses Pollutant Standards Index (PSI) and Europe uses Common Air Quality Index (CAQI) [4]. Most of the countries in the world have regulated six criteria pollutants, including Carbon Monoxide (CO), Ozone (O3), Nitrogen Dioxide (NO2), Sulfur Dioxide (SO2), Lead (Pb), and particulate matter (PM) with the size of less than 10μm (PM10) and 2.5μm (PM2.5) [5].

Currently, low-cost pollution monitoring is possible via different commercial sensors and a growth in the popularity of the use of such devices is observed worldwide. Sensors for particulate matter measurements are also available in many types. The common feature of all commercial PM sensors is the principle of operation - they measure light scattered by particles carried in an air stream through a light beam. Prices of such optical sensors range from tens to hundreds of euro, because they are generally cheap to manufacture. In addition, PM sensors are easy to use and in many cases ready to connect to microcomputers. Because of this, they are often adopted for use by citizen scientists [7,8].

Low-cost sensors are a solution to recognising and improving the temporal resolution of particulate matter data. A main example of their advantages is their low cost power consumption. They require a power supply voltage level of 5V and a working current usually around 250mA. They are relatively small, light and collect data at a high frequency. For these reasons and many more they are great choices for creating a wide spread sensor network.

The Internet of things (IOT) describes the network of physical objects - a.k.a. 'Things' - that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet[19]. This report designs and implements outdoor cost effective, portable air monitoring nodes. These IOT devices consist

of a SBC, air quality sensor and a power supply. These devices will transmit readings from the connected sensors wirelessly to a cloud store. A network with a total of six IOT devices is created and each node's PM reading is displayed in real time on an accompanying website designed to visualise the data for each sensor. Each node's previous readings can also be viewed over a 24 hour period and graphed according to the time and level of the PM in that area. This report sets out to encourage greater understanding and involvement of the public in air quality issues and citizen science initiatives using the internet of things.

2. Analysis

2.1 Related work

Some popular technologies used to map air quality are www.purpleair.com & www.airquality.ie. While both are adequate to read PM values they are somewhat lacking.

Purple-air offers twelve sensors around the cork city area with others around the country and the globe. PurpleAir sensors use a laser particle counter to count the number of airborne particles in the air. That count is used to calculate a mass concentration, assuming an average particle density in an algorithm developed by the laser counter manufacturer, Plantower. An average density must be used because not all PM of a particular size is made of the same build up. For instance, PM2.5 from wildfire smoke will have a different density than PM2.5 from dust blowing off a gravel pit. This means that mass concentration reported by a PurpleAir sensor can vary depending on the specific composition of PM for a given area thus making the sensors appear to "read high." So far, two different research groups have completed studies for their areas and created conversion factors specific to the composition of particulates in their air: AQ&U and LRAPA[9]. Purple-air offers two sensors available for purchase so users can monitor air closer to their area and add to the PurpleAir sensor network.

The first sensor is the PurpleAir PA-II sensor. Including shipping totals €245. The second is their PurpleAir PA-II-SD including shipping totals €270. The only noticeable difference between these two sensors is the PA-II-SD PM2.5 measurement detector incorporates an SD card and real-time clock, allowing the sensor to record and store data locally. Other than that, both sensors operate the same and measure the same amount of data. PurpleAir sensors require WiFi connectivity to run and connect to the map. However they only support 2.4Ghz WiFi networks. They will need special authorization from the network administrator to allow them to communicate on captive portal networks, such as those used by coffee shops, universities, etc. PurpleAir sensors do not support WPA2-Enterprise[10].

Purple-air sensors have a large well connected network. While their sensors are professionally developed and trust worthy, their price point is excessive for what they are trying to achieve and for consumers who just want to measure their local air quality. A lot of the sensors on the maps data is actually acquired from the Environmental Protection Agency (EPA) sensors which are dispersed around the country.

Airquality.ie is a website developed by the EPA which is designed to monitor air quality in Ireland. These are far from household sensors but rather large monitoring stations spread across the country. It could be particularly hard to try and monitor air close to home using these, as the location of the station is influenced mostly by being 'representative of the exposure of the population to air pollution' considerations in this regard include but are not limited to: Proximity to sources, Predominant wind direction in the area, Topography of the locality, Avoidance of measurement of micro-environments, Knowledge and experience of site assessment team[11]. The monitoring stations have been set up as The Local Monitoring Network which measures near real-time indicative results for particulate matter. The network is being set up as part of the National Ambient Air Monitoring Programme (AAMP) 2017 - 2022. Currently the Local Monitoring Network for particulate matter 2.5 microns (PM2.5) and 10 microns (PM10) [12].

While the EPA's monitor stations have an incredibly reliable network throughout the country, even employing people to maintain and monitor the stations, they are not idyllic for all citizens looking to monitor their air. They are by no means portable and tend to only be located in parts of cities with vast open areas. The public have no real opinion on where they should be located and can not purchase one for personal use.

2.2 Objectives:

The network created by these two programs is no doubt outstanding but this report will outline a more cost effective approach to creating a local sensor network while still incorporating a lot of the qualities that make the two programs impressive.

The main objectives of this project include:

- Creating a more cost effective network of nodes without compromising the quality of each individual node.
- Creating an interactive easy to use map for citizens to view air pollution around the city.
- Creating secure protocols to connect the sensor network to the cloud based backend.
- Displaying peak times of high PM count, but also allowing citizens to track air pollution in their area or any area they wish.
- Provide air quality modelling and forecasting capabilities, to provide an ongoing air quality forecast to the public.
- To encourage greater understanding and involvement of the public in air quality issues utilising citizen engagement and citizen science initiatives. Citizen Science is research carried out by members of the public who volunteer to collect scientific data. This research often focuses on monitoring biodiversity, invasive species and climate[13].

3. Design

In order to collect data around the city, six IOT devices were to be placed outside in various locations. To get a more accurate reading, the sensors were allowed to collect data for one week. Each node would consist of a PM sensor, a SBC and a power supply. Due to the fact they were measuring outdoor air quality, each node needed to be protected in a weatherproof box. The data collected from the sensors needs to be fed back to a cloud storage device, then it will then be taken from the database and visualized on a map of the city.

3.1 Designing the node:

3.1.1 The microcontroller:

When deciding on a microcontroller for the IOT device, the options were limited to either a Raspberry Pi or an Arduino as they were the main microcontroller and microprocessor on the market. The main difference between the two being that the Arduino is programmed in C dialekt and is mainly used for hardware applications to interface with motors and LEDs, while the Raspberry Pi running Linux can be programmed in any language available on the distribution, and is very good for software application. This however does not mean we cannot connect sensors and LEDs to the Raspberry Pi. To encourage people to learn programming by controlling hardware, the Raspberry Pi consists of a 40-pin GPIO, through which to connect different electronic components like LEDs, Buttons, Sensors, Motors etc. On the Arduino, the GPIO is called as Digital IO (for digital Input and Output) and Analog IN (for Analog Input).

Other differences include:

- The Arduino is a microcontroller board, while the Raspberry Pi is a microprocessor based Single Board Computer (SBC).
- The Microcontroller on the Arduino board contains the CPU, RAM and ROM. All the additional hardware on the Arduino Board is for the power supply, programming and IO Connectivity. Raspberry Pi SBC has all features of a computer with a processor, memory, storage, graphics driver, connectors on the board.
- Raspberry Pi comes with a fully functional operating system called Raspberry Pi OS (previously known as Raspbian OS). Although it can use different operating systems, Linux is preferred by Raspberry Pi Foundation.
- The clock speed of Arduino is 16 MHz while the clock speed of Raspberry Pi is around 1.2 GHz[14].

The Arduino is a good choice for repetitive tasks such as opening the garage door, switching the lights on and off, reading from temperature sensors, controlling a motor as the user wants, etc. While the Pi is good for performing multiple tasks, driving complicated robots, connecting to the internet, interface cameras, etc.

To develop an application to monitor Humidity and Temperature that displays the results on an LCD, the Arduino would be used to implement this. But to monitor Humidity and Temperature and send an email with the results then the Raspberry Pi is the right choice[15]. Due to the amount of raw data being collected as well as the needed pre-processing, Python is the preferred language for this task. Combined with the fact of how often it will be connecting to the Internet for live data on the site and cloud storage of the previous data, the Raspberry Pi was the chosen SBC for the node.

3.1.2 The sensor

One of the most important parts of the IOT device is the air sensor itself. Four of the most popular IOT sensors were analysed. All four sensors were all relatively low cost, measured the values the project set out to capture and had good reviews:

Nova PM Sensor SDS011:

This sensor was developed by Nova Fitness It works on the principle of laser scattering and can get the particle concentration between 0.3 to 10μm in the air. This sensor consists of a small fan, air inlet & outlet valve, laser diode, and photodiode. While it may have been a good choice to try incorporating into the node, it was not available from the universities verified suppliers, so could not be used in the project.

Dfrobot SEN0177:

This pm2.5 sensor was designed by Dfrobot and uses a laser scattering principle. Namely the scattering of laser irradiation in the air suspended particles. While collecting the scattered light at a specific angle to obtain the scattering intensity versus with time curve. After the microprocessor data collection, it gets the relationship between the time domain and frequency domain by Fourier transform, and then through a series of complex algorithms obtains the number of particles in the equivalent particle size and volume units of different size[16]. This sensor was available on the supplier list and was purchased for testing. During testing of the SEN0177 an issue became very clear, the sensor not only needs a 5v power supply but the GPIO pins also require 5V to operate. While the Pi can power it, it does not have that high level of GPIO pins. All existing libraries are also in C++. Judging by this and any tutorials available it was clear this sensor was designed for the Arduino and not the Raspberry PI.

Plantower PMS5003:

Designed by plantower this sensor has a small fan that sucks air through the sensor and past a laser that can detect both the number (and hence concentration) and the size of particles in the surrounding air. This sensor also uses the laser scattering principle much like the SEN0177 and too needs a 5V power supply, however the GPIO pins only needed 3.3V which suited the Raspberry Pi perfectly. Adafruit also designed a helpful library PM25_UART.

Fig 1. PMS5003 Sensor.

The library was written in Python suiting the code perfectly. This sensor met all the technical requirements of the project and was available from the universities supplier list. It too was purchased for testing with the Raspberry Pi. During testing of this sensor and exploring the library and documentation, the sensor started giving useful readings. These values were deemed accurate enough to proceed with development of the node and the PMS5003 was now the chosen sensor for the IOT devices.

The below diagram explains the laser scattering principle. Air flows straight through the sensor but while inside a laser beam passes through a dispersed particulate sample, the particles scatter light relative to their size.

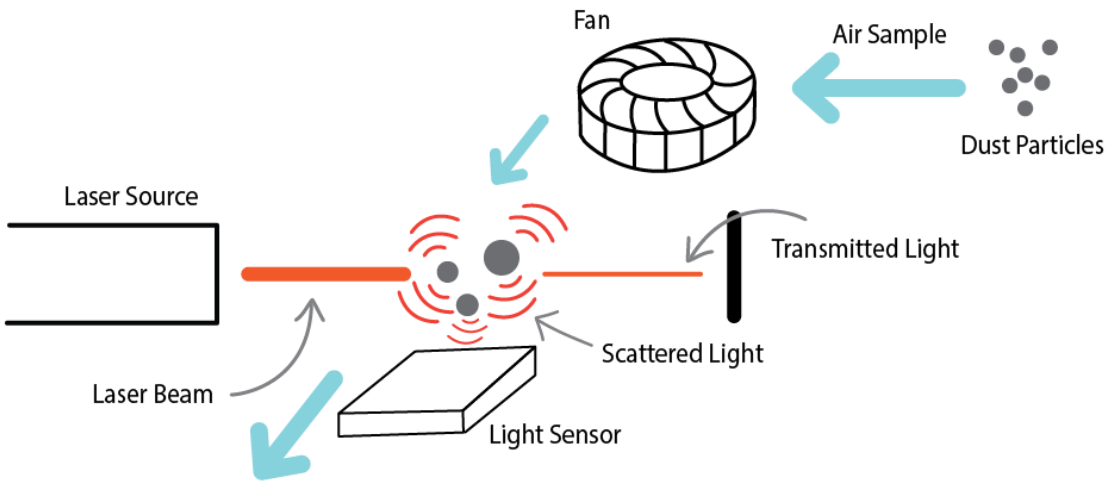

Fig 2. Laser scattering principle [17].

3.1.3 The power supply

The final piece of hardware for the IOT device is its power supply. The nodes need to be able to record data for a week long period meaning a reliable long lasting power supply is essential. Three power supply options were analysed for the sensor network:

Battery power:

Battery power appears a practical solution, with obvious advantages being the nodes could be placed almost anywhere around the city. However after examining the Pi and the sensor power consumption, it became clear the longest duration achievable from the batteries would be about 20 hours. This would result in a constant changing of batteries and with six nodes this added work could be easily avoided.

Solar power:

Solar power offers the same advantages as battery power however due to the fact the experiment is in Ireland, it was not a great option. It was not possible to guarantee having enough sun to run it for the week, and there would also be the added expense of the solar panels for the nodes.

Fig 3. Raspberry Pi 3 B+

Wired connection:

While it does not offer the same usability of batteries and is somewhat limited due the distance it can be away from the power socket, it was a safe reliable option of having power 24/7. This would just mean choosing locations more carefully to position it out a window. After examining all three options, a wired connection seemed the best option and was chosen for the project.

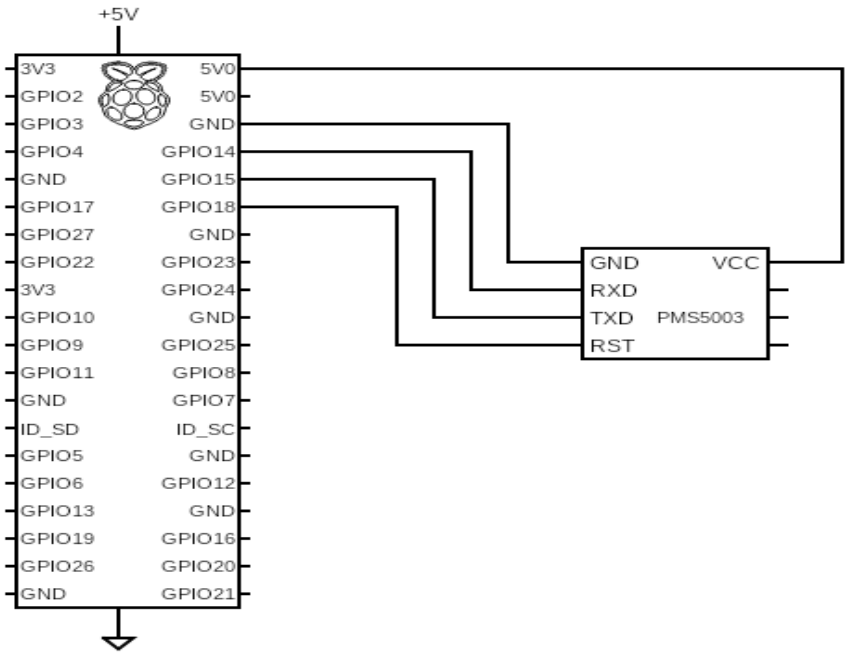

Fig 4. The wiring diagram for each IOT device.

3.2 Designing the website

3.2.1 Saving the data

Each node records data every ten minutes for an entire week. Meaning they would be storing over 1000 values each over the course of the experiment. This highlights a major task of deciding how the sensors are going to store data. There were two main options when tackling this problem.

Saving the data on Pi:

This consists of writing the data to a log file or creating a local database on each Pi. The data would then be processed at the end of the experiment and inputted into an external database. This would mean live data could not be displayed on the site, and the map would only be updated anytime the data was collected personally.

Saving the data externally:

Achieved by having an external cloud datastore that each node in the network would connect to. Meaning the website would have access to data in realtime for the interactive map. The idea of live data is a lot more interesting albeit more challenging. The nodes would process the raw sensor data and send it to a cloud datastore. This would be accomplished by each node needing a reliable WiFi connection as they would each have almost 150 daily inputs. While this option is certainly the more challenging of the two it was chosen for the project solely because live data would make the website much more interesting.

3.2.2 Sending the data

Another important decision that has to be made is how the data is to be sent to the cloud store. There are several ways the nodes could communicate with the datastore. TCP/IP transport supports connections to local or remote MySQL servers. Socket-file, named-pipe, and shared-memory transports support connections only to local MySQL servers. (Named-pipe transport does allow for remote connections, but this capability is not implemented in MySQL.) Since the data is being saved in an external cloudstore, TCP is a better option.

3.2.3 Storing the data

As for the actual datastore choice, the main options available were either a MySQL database or a firebase database. During the testing of both storage systems during the project, it became clear the Raspberry Pi interfaced far better with the sql datastore as opposed to a NoSQL. Firebase does not directly support Python and also requires the use of JSON on both the Raspberry Pi and the web page whereas MySQL would be solely Python.

The main issue with having an SQL datastore is the cost of renting an external database whereas firebase is free(to a certain extent). However each UCC Computer Science student is granted a free MySQL database on the CS1 server which is live 24/7. After evaluating all these factors, the decision became clear to progress with a MySQL datastore.

3.2.4 Displaying the data

Data stored in a database is not useful to people trying to monitor their air. Even if users could query the database, it would be thousands of raw data with no use unless compiled together. Using this data as input, a dynamic website can be created to show users different sensor values around the city. There are several options when it comes to displaying data on a webpage. The initial basic way being displaying a large table with all the values of each sensor

inside. Another option would be to give users a downloadable file from the site to the data of the sensor. Thirdly we could display a graph of all the sensors and change the dates accordingly. The final option being to display a map showing the location of all the sensors, with an embedded graph for each date and PM value.

Graphing the data is clearly a more user friendly option and would allow for much easier understanding of the data too. The idea being to generate a map of Cork City, displaying each sensor's current value. When the sensor is clicked on, it will show a graph for 24 hours of data for the selected date and PM value. Several of the most popular map options available were analysed when trying to develop a map for this project.

Open layers:

A completely free solution for displaying dynamic maps in a web page. It draws tiles from a variety of sources, including OpenStreetMap. In addition to the basic maps, OpenLayers also allows users to render vector layers on their map and place drop markers.

Map box:

Both the choice for Facebook and Snapchat's mapping needs. Drawing data from both open and proprietary sources, Mapbox offers maps, location searches, navigations, and custom map features. While it is a good choice for advanced mapping applications, other applications proved easier to integrate with this project.

Google maps platform:

Google Maps is by far the biggest player in the web mapping space. With a large library of related APIs and extensive documentation it is a reliable choice for the project. The Google Maps Platform replaced the old google maps API as a new set of APIs and SDKs that allows developers to embed Google Maps into mobile apps and web pages, or to retrieve data from Google Maps. Through testing the API by creating maps and custom markers, the platform appeared very useful but integrated alot better with PHP than Python. Due to this, the platform was chosen to be the mapping software and PHP was chosen as the language for the site.

3.2.5 Graphing the data

Displaying the data in graphs offers useful information and visualization of the data on the website to any user. As opposed to reading thousands of figures they can analyse the graph and see spikes of high PM and see at what time they occur. Several options were analysed for the graphing of this project's data.

Matplotlib:

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK[18]. While

Matplotlib is an appropriate choice for such a project, the Google Maps Platform integrated much better with PHP so could not be used for this project.

Chart.js:

This is a free open source javascript library for data visualization. While testing this library it was found to be very static. This library seemed great for hardcoded values in the source code but when it came to integrating with PHP it fell short. While it was possible to dynamically load data into the chart, it proved quite difficult. For this reason it was not chosen for the project as there were more user friendly options.

CanvasJS

CanvasJS is a HTML and Javascript Charting library. It runs across multiple platforms and has great maintainability and functionality. It has a well documented API and is Standalone - not depending on any other library. From testing of this library it suited the needs of the project perfectly and was chosen as the charting software. It integrates very well with PHP but not at all with Python, another reason PHP is the chosen language to query the datastore.

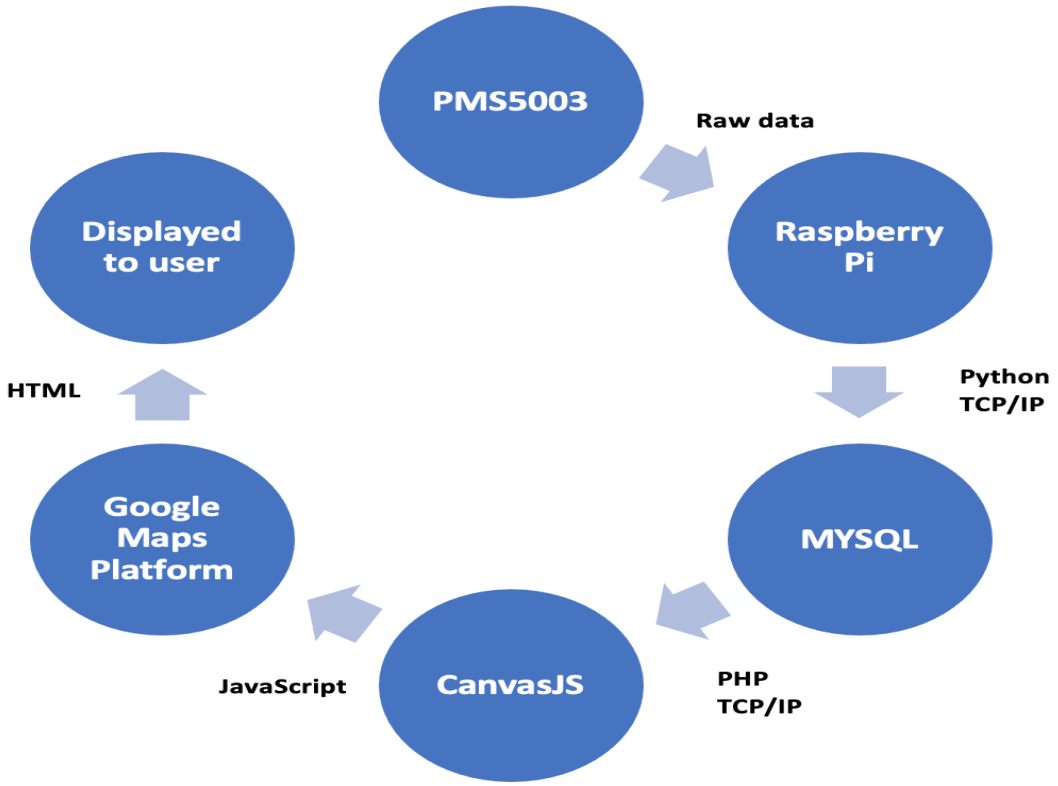

Fig 5. The flow of data from the sensor to the user.

4. Implementation

4.1 Hardware

This project creates an IOT sensor network from a total of six nodes. These nodes are spread through Cork City, to try and disperse them a bit they were placed at the Boole library, Western Gateway Building, Mardyke walk, Highfield Avenue, College Road and Bandon Road. Due to the fact the network is looking to measure outdoor air pollution, the sensors must be placed outside. This instantly raises some security issues for the nodes such as weather, power and vandalism.

Weather:

The problem of protecting the nodes from the elements is one of the most important. If this is handled incorrectly it could result in the Pi and hardware being water damaged or overheating. This damage could not only result in damage to the hardware but a loss of collected data throughout the experiment. To tackle this problem, all three components of the node were placed in weather proof boxes. This created a new issue as the boxes are water tight, meaning the node would have no access to the air it is supposed to be measuring.

By drilling a 2cm hole in in the front of the box, the fan in the sensor would have enough room to draw air in. To make sure the box was not hindering results, two sensors were set up side by side. One encased and the other not encased. Both sensors reported the same readings meaning the hole was big enough to access the air properly. While this is a practical solution, a tradeoff was made by sacrificing the nodes waterproof features with the newly added hole. The new access point to the box for water was addressed by placing the Pi at the furthest point away from hole inside the box, it will also be addressed in the placement of the nodes.

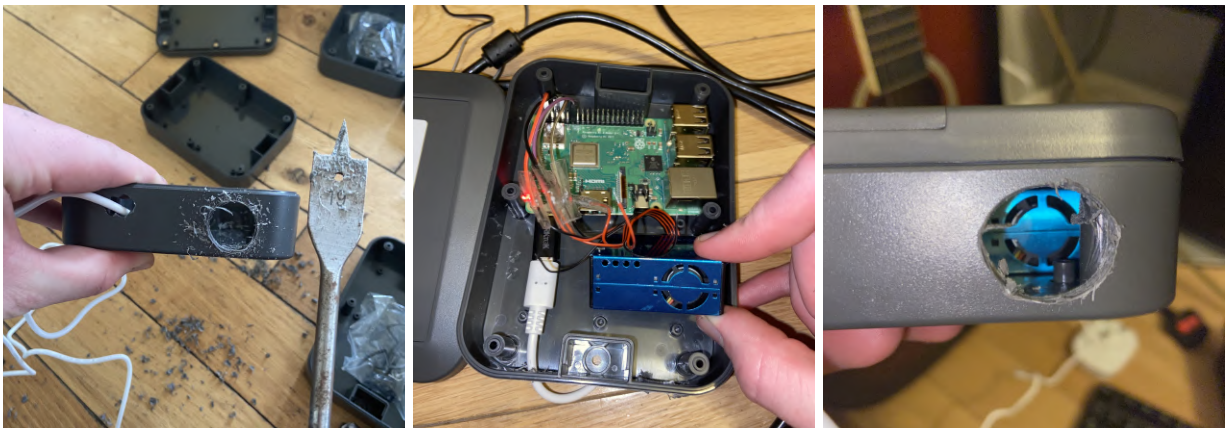

Fig 6, 7, 8. Preparing the nodes for deployment by connecting the sensors and drilling the holes in their weather proof boxes for the power supply and for the sensors fan to drag air in.

Power:

The issue of powering the nodes also became clear when trying to plug in the power supply. A second smaller hole was drilled on the same side of the first. Keeping both holes on the same side reduced the chance of water intake as three sides would be waterproof.

Vandalism:

A major risk with outdoor units is always vandalism. The nodes had to be placed in secure places or up high, rather than an easily accessible or visible place to pedestrians. The main choice of the node positioning were upstairs window sills. This allowed good access to air with relatively low access to any passers-by. The nodes were positioned along the slope of the window sills with the air intake side facing out the way. This not only allowed for the best airflow, but also protected the air intake side with the two holes from water running in. The nodes were then secured to the ledge with double sided tape and the power cable ran in through the corner of the window to allow it to close.

Deployment:

With all precautions taken into account, the nodes could then be assembled. There was minor soldering involved attaching the ribbon cable from the sensor to 5 female jumper cables to attach onto the Pi. This to make up for the lack of a breakout board on the unit. All wires were then wrapped in insulated tape. The power supply was also taped to the Pi in case it took fall damage so as not to knock it out of place. These images show the assembly of each node. The unit was enclosed in a 40 mm*110 mm*150 mm ABS plastic enclosure. The lid has an airtight seal and is attached to the base with screws.

Fig 9, 10. Shows assembly of the IOT devices

The measuring of the position of the screw fixings was not taken into account. Due to this the Pi did not fit exactly as planned but with careful construction eventually fitted comfortably. The screw fixings also acted as a barrier to keep the microcontroller from slipping down to the air intake side where it would be at risk of water.

After assembly of the six IOT devices, each node is thoroughly tested before being deployed. This ensured the Pi could still access the WiFi and the sensor itself could measure air. A longer power supply would have been more desirable as one metre can be quite limiting while trying to position a unit out a window.

Fig 11, 12. Shows the deployment of the six IOT devices around Cork City.

4.2 Software

The software of this project is split into major two parts. The first part being the software that runs on the node to collect and pre-process the data. The second part being the software that runs to create the website for data visualisation and analysis. Both of these two parts are then broken down into smaller parts covering from collecting raw data on the node to displaying it in a graph on the website.

4.2.1 Node software:

The nodes software was designed in Python3. Running on a Raspberry Pi with a Raspian OS. Raspian is built on top of a Linux kernel making it a Linux distribution. While Raspian suited this project quite well, a major problem arose. Due to the fact that the six nodes are distributed over the city, they are not necessarily easily accessible to monitor or maintain. They are also set up in a variety of locations where they could become accidentally unplugged or lose power. If the system loses power, when it turns back on, the script will have stopped. Meaning there had to be a way for the node's script to run automatically on boot. This was achieved from editing each Pi's /etc/rc.local file. This file is a list of commands the Pi runs on execution.

Adding the command: “python3 /home/pi/sensor.py &” to the file achieved this. As the script is essentially an infinite loop the “&” had to be added to the end of the command. This ampersand allows the command to run in a separate process. Without this the process would get stuck waiting for an infinite loop to end before it boots the rest of the Pi.

Reading data:

The SBC and the sensor are connected through the Pi’s GPIO pins using Universal asynchronous receiver-transmitter(UART) as the selected communication protocol. To read the raw data from the sensor, a module called PM25_UART from a package developed by Adafruit called adafruit_pm25.uart is used.

A function called readSensor was created for the project. It takes in a variable num which tells it how many reads of the sensor to take. It calls the previously imported library pm25 and triggers the read of the data. It returns a read of PM1.0, PM2.5 and PM10.0 particles as a dictionary called aqdata (Air Quality Data). While running the program for extended periods of time it encounters a rare RuntimeError, however the next iteration of the loop works. For this reason the reading of the data is placed in a try except statement. After this block each of the three values are put in global lists. By the end of the function each list will be the length of the initial variable num. Due to the fact the values of the sensors tended to fluctuate, the average of each list is taken and that is the result to be inputted to the database.

```
def readSensor(num):
    for i in range(num):
        time.sleep(1)

        try:
            aqdata = pm25.read()
        except RuntimeError:
            print("Unable to read from sensor, retrying...")
            continue

        l1.append(aqdata["pm10 standard"])
        l25.append(aqdata["pm25 standard"])
        l10.append(aqdata["pm100 standard"])
```

Storing data:

Since we are using MYSQL at the back end, Python will need a MYSQL driver in order to access the database. The chosen driver is MYSQL Connector as it enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API Specification. This method uses TCP/IP as the communication protocol ensuring data safety.

The main issue with the database connection was it added the need of a stable WiFi connection to the project. The network issue also only came from running the data over extended periods of time. If a connection was lost for any reason at a time the program tried to execute a

query, the program would crash and not reconnect. To solve this a loop was added before the execution of any queries. It checks whether any database connection exists, if not then it loops around, continuously checking until the WIFI connection has returned. When a connection is recognised it executes the query.

There are a total of eight databases on the server in this project. Each node connects to a total of three databases each. One individual database per node and the other two are shared among all six nodes. The first shared database is for the use of live data. This database stores each node's information such as the X and Y coordinates, the name of the location and the current PM1.0, PM2.5 and PM10.0 of each sensor. Each node sends its current reading of each value which overrides the previous value. This database doesn't store any past values and is very efficient to see the current values of each node.

```
mycursor = mydb.cursor()
sql = "UPDATE cork_air SET PM1 = '"+str(Average(l1))+ "'WHERE id = '"+sensor+"'"
mycursor.execute(sql)
sql = "UPDATE cork_air SET PM25 = '"+str(Average(l25))+ "'WHERE id = '"+sensor+"'"
mycursor.execute(sql)
sql = "UPDATE cork_air SET PM10 = '"+str(Average(l10))+ "'WHERE id = '"+sensor+"'"
mycursor.execute(sql)
mydb.commit()
```

The coordinates and location name are static and never change, whereas the list of values recorded previously are averaged here and placed in the corresponding column name. The sensor variable is hardcoded into the script to match with whatever node it is running on e.g. Sensor1.

The second shared database is used to dynamically populate the dropdown menu on the website. Dynamically populating this drop down means the data will not have to be hard coded as well as meaning it is only possible to select dates where data is available.

```
dt=datetime.datetime.now()
day=dt.strftime("%j")
day2=dt.strftime("%d")
month=dt.strftime("%B")
combineday=day2+" - "+month
minute=str(dt.minute)
hour=dt.hour
if len(minute)==1:
    minute=minute+"0"
combine=str(hour)+":"+minute
mycursor = mydb.cursor()
sql="INSERT IGNORE INTO Dates SET number='"+day+"',days='"+combineday+"'"
mycursor.execute(sql)
mydb.commit()
```

This method also allowed for pre processing of the date before it was put into the database. Every time the script hits midnight it will recognise a new day, it takes apart the current date and stores it as "31 - March" , "01 - April" etc which appear nicely in the dropdown when selecting a date. While originally only supposed to run on one node as they essentially would all be inserting the same day, a decision was made to run it on all nodes. This was a safety precaution incase of

the failure of the node that was running this function. If for any reason that node failed, the others would be inserting data for a date that could never be viewed. The problem of inserting the same date was solved with INSERT IGNORE meaning if the date was already stored, do not insert it again.

The final database each node communicates with is their own personal database. This database stores every read of the sensor throughout the whole experiment. By the end of the project each nodes database will have over 1,000 rows and thousands of data points. The decision of keeping each one separate was made for security reasons. Meaning if one database failed, the experiment could still work with 5 nodes. This also makes for easier data visualization when it comes to graphing the data.

```
mycursor = mydb.cursor()
sql = "INSERT INTO "+sensor+" (PM1,PM25,PM10,Day,Time) VALUES ('"+str(Average(l1)) \
+"','"+str(Average(l25))+"','"+str(Average(l10))+ "','"+str(day)+ "','"+combine+ "');"
mycursor.execute(sql)
mydb.commit()
```

This snippet creates a new row in each database for storing the PM values, the current day and the current time. The sensor variable is initialized for the database the script is querying.

The whole sensor script is essentially a forever loop. The variables are initialized for how many reads the sensor will do, how many minutes between each read and the current sensor database. Inside the loop the script checks the date and time, then runs the main function readSensor after the specified time has passed.

```
old=0
reads=10
readingTime=10
sensor="CorkAirSensor1"
while True:
    x=datetime.datetime.now()
    if((x.minute%readingTime)==0 or x.minute==0) and x.minute!=old:
        print("Resetting")
        l1=[]
        l25=[]
        l10=[]
        old=x.minute
        readSensor(reads)
```

The time period is checked by dividing the minutes of the time by how many minutes it is specified to wait for. If the result of this equals 0 or if the current minute is 0 (meaning on the hour) it passes the OR statement. For example if the time is specified for 10 minutes, the code will run at 1pm, 1.10pm, 1.20pm etc. The result of this is then fed into an AND statement checking the current minute with the variable ‘old’. This variable stores the previous minute recorded and makes sure it is not recorded twice. If this statement is True it initiates three global lists to store the PM values. The function readSensor is then called with how many times it

should read the sensor. When the function returns, the time is checked again, the lists are re-initialised to empty and the process starts again.

4.2.2 Visualisation software:

Throughout this experiment over eighteen thousands data points have been collected. To help visualise all this information a website was created to act as a more visual representation of the information instead of just meaningless lines of numbers.

Creating the site:

The website is currently being stored on the UCC web server. This server is free to all CS students and acts as a secure route to putting the information online. The site is created through HTML, CSS, Javascript and PHP. PHP was the chosen programming language as it integrates very well with not only the mapping software but also the charting software to display the graphs. To access the database, the MYSQLi extension was used. This is an improved version of the MYSQL extension and was developed to take advantage of new features found in MYSQL systems versions 4.1.3 and newer. This extension is included with PHP versions 5 and later. It is extremely useful for creating a connection to a MYSQL database.

The site itself consists of a main map of Cork city. The map was created using the google maps platform. Each marker on the map represents a node in the sensor network. The nodes display their current location and real time PM values. The markers are colour coded green, yellow and red to indicate the safety of the air in that location. This initial map of locations and live data is loaded from the one database. This database stores the coordinates of the sensors and its current PM1.0, PM2.5 and PM10.0 values.

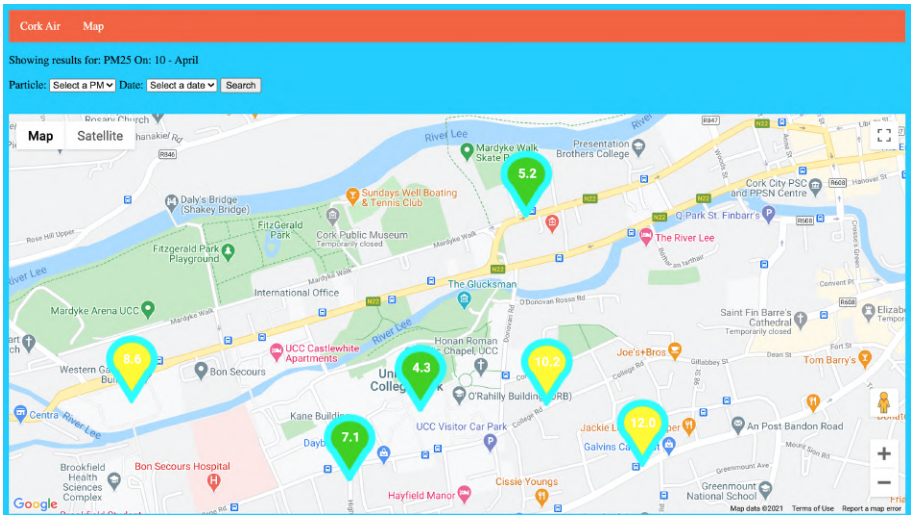

Fig 13. Screenshot of the sensor map.

Each marker not only represents the current location and value, but also acts as an entry point for viewing the previous values. When any marker is clicked on it will show the 24 hour PM values

for that location, for the selected date and selected PM value. The data to display the graph is fetched from a total of six databases, again using the MYSQLi extension as the previous main database.

```
//Connect for graph 5
$sql = "SELECT ".$pm.",Time FROM CorkAirSensor5 WHERE Day=".$date.";";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {

array_push($sensor5,array("label"=> $row['Time'], "y"=> $row[$pm]) );
  }}
```

Each graph is initialized with an empty array. Each database is then queried for the time and PM value. The appropriate array is then populated with several new arrays acting as a dictionary. The data is already ordered from the input from the Pi so no more processing is needed at this point. The graph now has its input, but the data is in PHP while the graph renders in JS.

```
function initMap() {
  var chart1 = new CanvasJS.Chart("chartContainer1", {
  axisX:{
    crosshair: {
      enabled: true,
      snapToDataPoint: true
    }
  },
  axisY:{
    title: "PM",
    includeZero: true,
    crosshair: {
      enabled: true,
      snapToDataPoint: true
    }
  },
  toolTip:{
    enabled: false
  },
  data: [{
    type: "area",
    dataPoints: <?php echo json_encode($sensor1, JSON_NUMERIC_CHECK); ?>
  }]
  });
```

The function initMap() is a JS function that creates the six graphs for display inside the modal window when a location is clicked. This creates the settings for the graph including how they look and operate. The main challenge of this function is it usually takes its input as static JS values. To try and combat this, a line of PHP code was entered for the dataPoints variable. The sensor array populated in the query function is encoded in JS and echoed to the console. The

function reads this as the JSON representation of the array. When compiled it will render a graph as displayed below. On the Y axis we can see the PM values and on the X axis we can see the time values of the 24 hours. This method of displaying data is a lot more user friendly than just displaying the raw PM values.

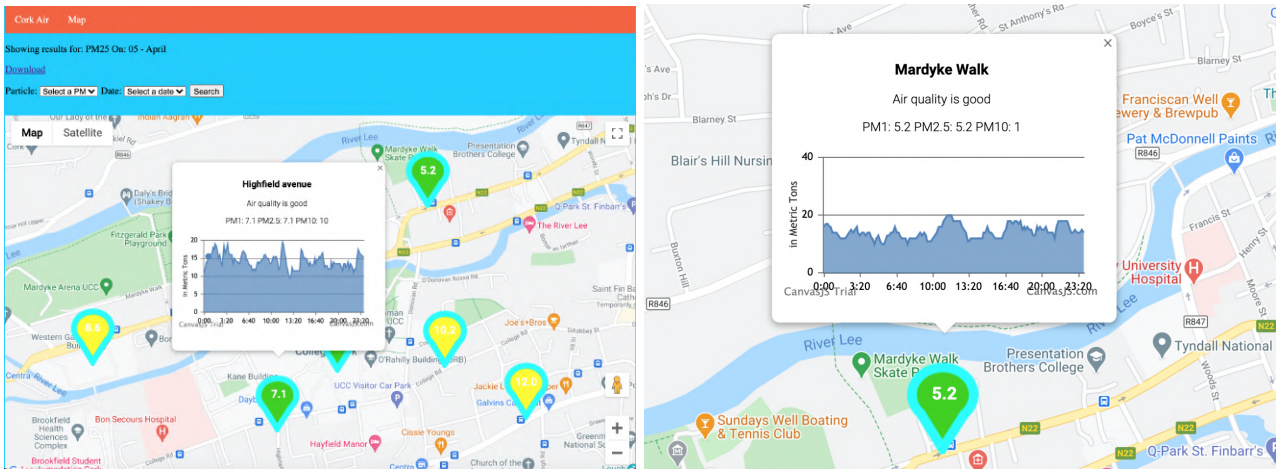

Fig 14, 15. Screenshots of the graphed data.

The main issue with displaying the data like this, is the browser would only render the data as one graph. This resulted in the map displaying data for only one graph. This issue was solved by creating a function called renderGraphs and setting a time out of 10 milliseconds between when each graph was rendered.

The search:

To be able to select values to query, a search medium must be created. The chosen search option was a pair of dropdown menus. The first selects PM1, PM2.5 or PM10. This is a static dropdown as it never changes. The second drop down selects the date for the search. This is a dynamic dropdown populated from a database solely devoted to storing the available dates to search. When requiring user input, an immediate issue raised is error handling.

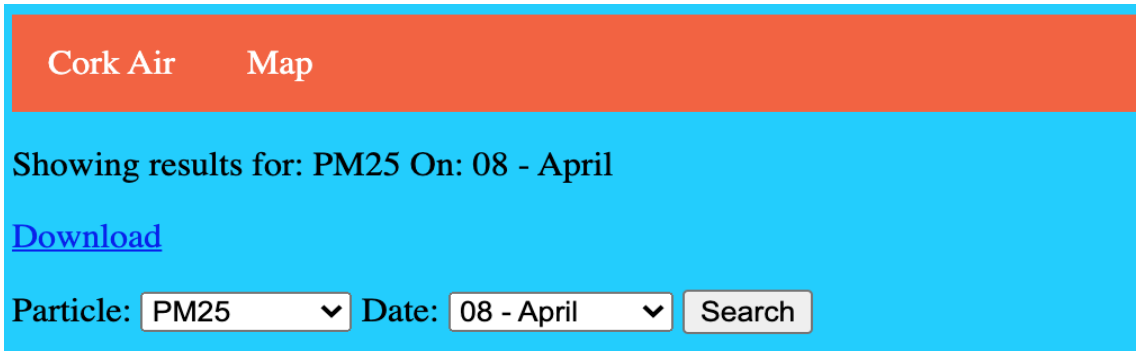

Fig 16. The dual drop down menus for the search.

The search was created with a default state of PM2.5 and a default date of the most recent entry in the database. This allows for if the user manages to enter a date or PM that would result in a failed query. If the query were to throw an exception, the values are changed to those shown in the above image. In the interest of contributing to open source, the raw sensor data was also made available for download for the selected date.

5. Evaluation

5.1 Hardware

After the week-long period was finished, the six nodes were collected, all making it back in perfect condition. While the weather fluctuated greatly from hot temperatures to heavy rain and strong winds, the weather proof box kept the internal hardware safe and secure. The holes drilled in the front of the box succeeded in capturing air without letting water in as the inside of the node remained dry.

The choice of wired power supply resulted in an up time of over 99% across the IOT network. While small power outages in the city were not desirable during the experiment, a wired supply was still a better option over batteries as they would have had to be changed less than every twenty four hours. While limited to the distance the nodes could be placed from a power supply, all nodes still managed to be placed outside in all locations.

The Raspberry Pi is a very useful and reliable SBC for the network. While both Arduino and Raspberry PI boards are similar in size but the main reason for using the Pi is it runs an operating system as opposed to the Arduino which only runs programs compiled on the Arduino platform. An OS was important as it had more network interfaces which is needed due to how often the node would be communicating with the database. Over the course of the week, the Pi's relayed information with almost 100% reliability. Some short power outages interrupted the flow of data, however the change to the rc.local files managed to start up the network autonomously, keeping the data live on the site.

5.2 Results

The website created to visualise the data successfully displayed live data from each sensor over the course of the week long period. After the sensors were collected the data is no longer live but still displays all previous values. The main way of testing the validity of the sensor readings, was comparing the data to purple airs sensors. There is an EPA sensor located in Fitzgerald's Park, not too far from the sensor that was placed on Mardyke Walk for the experiment. Using this sensor as a control for the experiment gave the sensors merit and something to verify the data off. This sensor is used to measure pollutants from gases to several

types of particulate matter. It measures the same three values set out in this experiment PM1.0, PM2.5 and PM10.0.

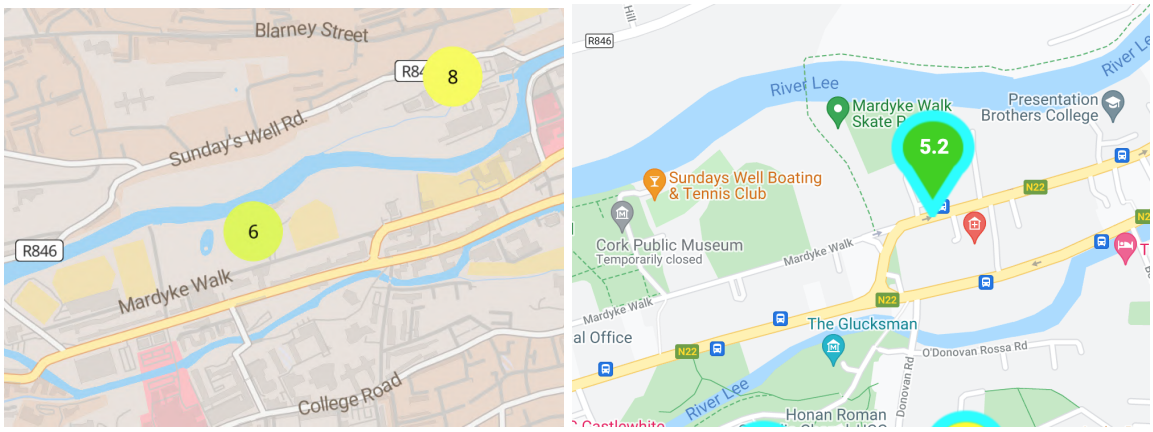

Fig 17, 18. Location of the purple air sensor and this experiments Mardyke Walk sensor.

Over the course of the week the Mardyke Walk sensors values were compared with the EPA sensor values. While measuring the values against the control it must be noted that the sensor was not placed in the exact position of the EPA sensor as a house on Mardyke Walk was the closest location that could be achieved.

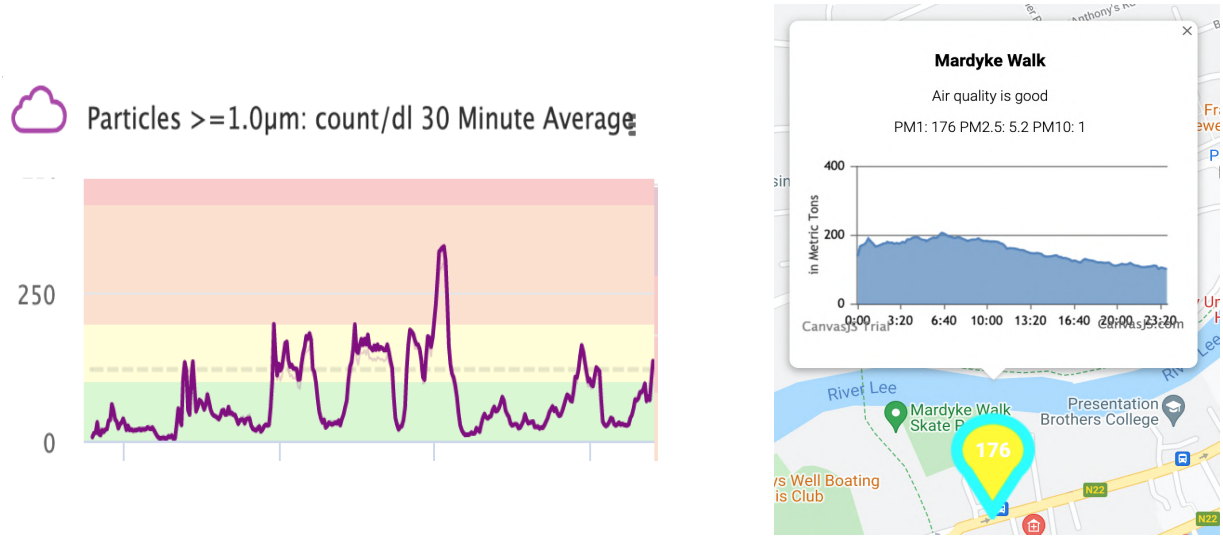

Fig 19, 20. PM1.0 comparison against purple air.

The above screenshots are an example of one of the sensor values for PM1.0 for a specific day. Many similarities can be seen between the two and both rest around 200PM. It can be seen that both sensors have high values at the start of the day and mellow out towards the end. The EPA sensor appeared more sensitive throughout the experiment and PM1 was also the value that matched the control values the least out of the three PM values.

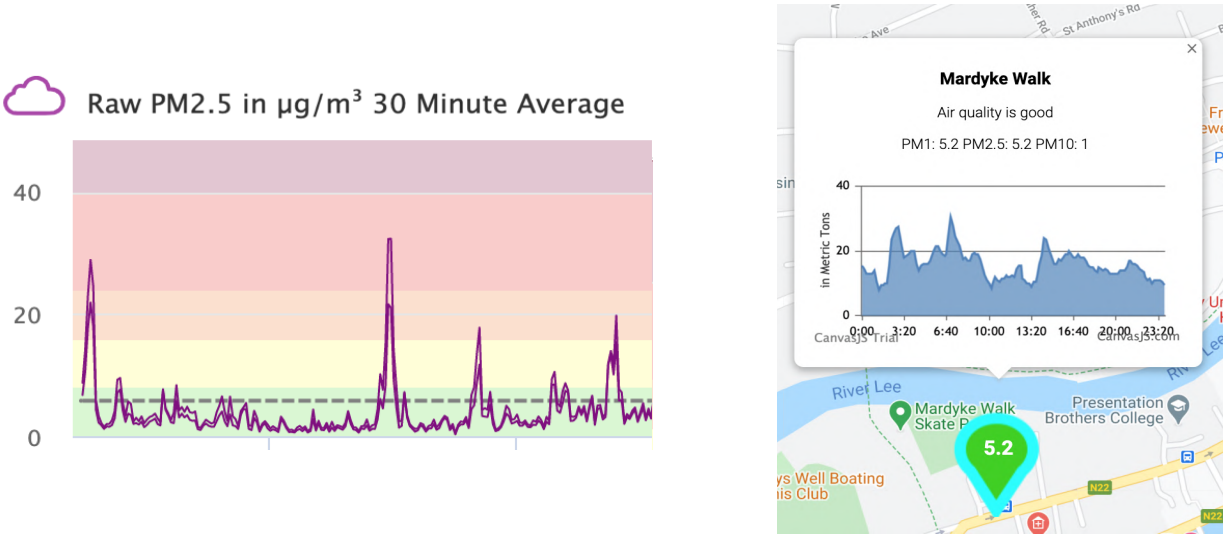

Fig 21, 22. PM2.5 comparison against purple air.

The above screenshot shows a day the PM2.5 values were tested against the control. The results of this particulate matter value matched the controls much better than that of PM1. Spikes in the earlier part of the day match that of the control. Then easing out in the middle of the day and spiking in the evening. This was a common trend throughout the experiment perhaps replicating rush hour before and after the working day. PM2.5 was the particulate matter value that matched closest to the readings of the EPA sensor. This is understandable as the PMS5003 is marketed mainly as a PM2.5 sensor.

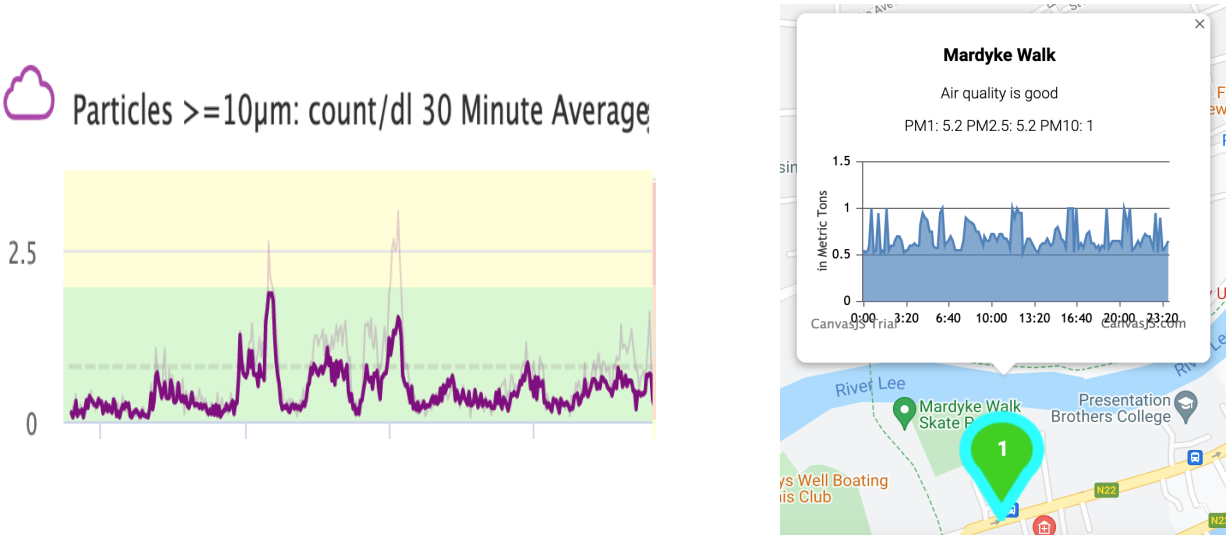

Fig 23, 24. PM10.0 comparison against purple air.

The above screenshot shows the comparison of a day the PM10.0 values were tested. Several spikes throughout the day matched that of the control. This too was a common trend throughout the experiment with values doubling every 2 - 4 hours and then returning to the original. The PMS5003 did a sufficient job at measuring PM10.0. While it did not quite match its performance against PM2.5, it did beat the PM1.0. Even though the readings were not expected to be identical but rather just give the node a sense of value, nonetheless using the EPA data as a control gave the readings some merit as they tended to match the values.

5.3 Cost

A main goal of this project was to develop a low cost IOT monitoring node. While the choice of microcontrollers did not influence the node cost too greatly, the choice of sensors had a great impact on price. While stronger more accurate sensors could be purchased, a tradeoff had to be made between cost and price in order to keep costs down as sensors ranged from less than ten euro to hundreds of euro. The choice of PMS5003 met the requirements of high quality readings at a low cost with a value of €34.80 each. The Raspberry Pi microcontroller used to control the sensor, including the accessories to power it totalled €38.70. The final component was the weatherproof box that contained the rest of the node. At €19.90 each they are expensive for just storing the unit, however the durability of the box earns its price as poor encasing could result in loss of all prior components. This brings the total cost of the node to €93.40 each.

Fig 25. Purple air sensor (€205)

Fig 26. Sensor node for the network (€93.40)

Purple air's sensors range from €205 to €230 on their site, meaning that this experiment produced the nodes for less than half the price.

6. Conclusion

6.1 Achievements and future work

Many low-cost PM sensors are available these days on the market. This research project successfully proved the capability to create a more cost effective IOT network of PM sensor nodes without compromising the quality of each individual node. At an end price of more than 50% less than purple airs node, it provides a more reasonably priced sensor for users who want to monitor their local air quality or contribute to citizen science initiatives. While the purple air sensor certainly looks more esthetically pleasing, this was not a product design experiment, and the overall look and feel of the node was not a major objective. Future work of this project could include designing a more attractive housing that would not stand out so much when set up outside each location.

By successfully creating an interactive easy to use map for citizens to view air pollution around the city, it removes the need of larger companies' unique selling point of offering data visualization of their nodes purchased for their network. While this project only uses several IOT nodes set up internally, future work could include making the project and network open source to try and help other people set up their own nodes and contribute to the sensor network.

The secure protocols created to connect the sensor network to the cloud based backend have many future potentials. This method of using TCP/IP ensures data safety with encryption of the header and payload to and from the datastore. This same technology could be used to manage a range of other sensors in the air pollution space or even any other research field. A lot of the technology created here is transferable to other IOT projects, especially those run on the Raspberry Pi. Such as using MYSQLi to create secure connections and the mapping methods as shown in this project to display data in a more visually appealing manner. Making this secure easy to use technology available could aid encouraging a greater understanding and involvement in citizen science initiatives.

By displaying peak times of high PM count, trends in air pollution can be observed. From analysing these trends citizens can not only see when the best time to go outdoors is to avoid high air pollution but are also be able to view the current PM count in their area.

From analysing the sensor data, the following conclusions can be observed. Sensors which were placed in high traffic areas such as Mardyke Walk, College Road and Highfield Avenue all seemed to follow the same trend over the week-long trial period. PM count would start off quite low in the early hours of the morning and see a spike towards around 6-7am. The PM would then tend to mellow out before spiking again around 2-5pm. This was a very common trend throughout the whole week especially for PM2.5. As a common cause of PM2.5 is the combustion of liquid fuels this makes sense as times around 7am and 4pm are peak rush hour times in the city for work and school.

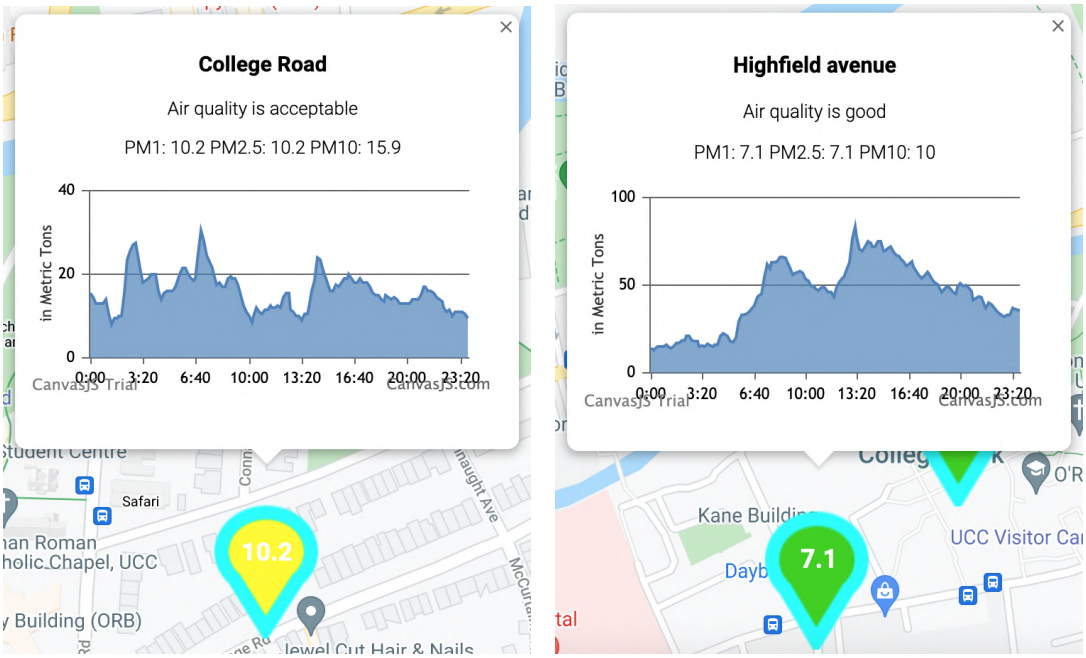

Fig 27, 28. The PM trends analysed on College Road and Highfield Avenue.

UCC appeared to have the best location for air quality. The sensor placed on the Boole Library had the best supply to clean air. Although it was close to other sensors such as the Mardyke Walk node, the sensor did not come in contact with as high levels of pollution as the others.

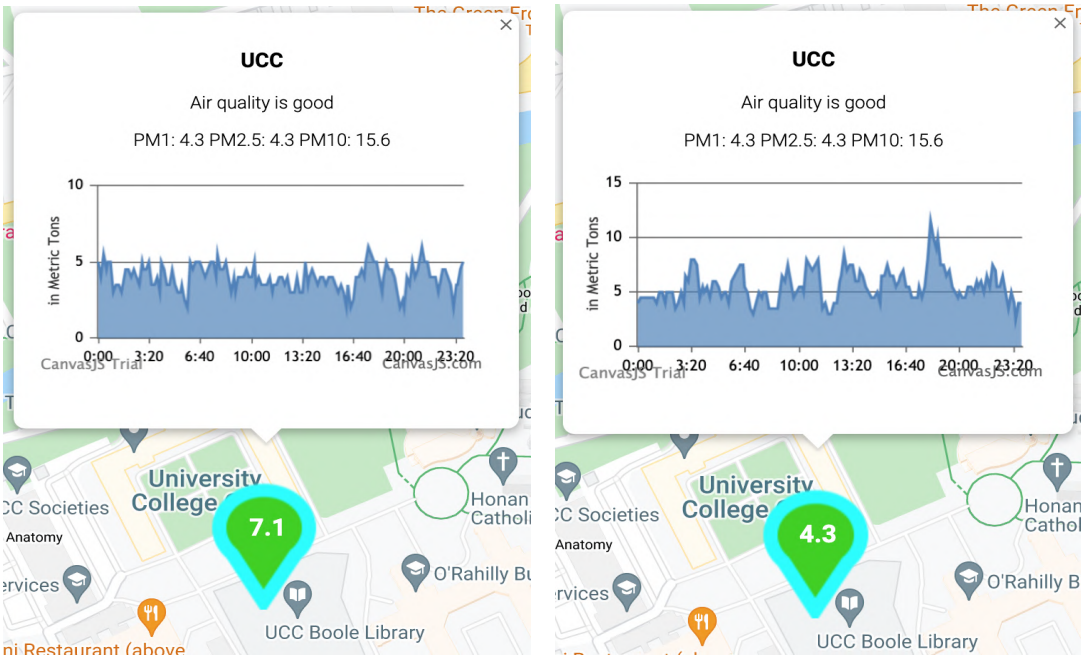

Fig 29, 30. The PM trends of UCC throughout the experiment.

As shown here, opposed to the rapid spikes of early morning rush hour, the UCC node spent the week measuring a higher purity of air. This may be due to the fact that the sensor was facing into the main campus away from any busy roads. Meaning the only way people would pass it may be on foot, or on a bicycle. These graphs clearly reflect the impact that vehicles are having on the environment and our air quality.

This project set out to encourage greater understanding and involvement of the public in air quality issues utilising citizen engagement and citizen science initiatives. The data of this project can be viewed on the site for the week-long period the network was active, even though the nodes are currently not recording anymore. Not only is the data collected visualized for the public but it is also made available for download to compare or be used with any future work looking at measuring air quality or IOT networks.

6.2 Reflection

Overall, the study showed that low-cost PM sensors are effective tools for air quality monitoring. The results of this experiment could help researchers create measuring systems and choose low cost air quality sensors. This project had a big learning curve, from creating and deploying a fully operational sensor network, to visualizing the raw data collected. This has been a great opportunity to learn new skills and technology that would not have been available to experience in the standard college degree.

While it has been very enjoyable incorporating hardware and electronics into the project as opposed to just a standard software project, it has also been very interesting studying air pollution and the effects particle matter is having on the human race and the environment. The reason I found studying air pollution was interesting, is because in a computer science course, you never get to study anything outside the scope of the degree, mainly consisting of algorithms and other elements of IT. It was very enjoyable to bring knowledge from the degree and try to incorporate it into a real world problem.

References:

[1] H. Amos, "Poor air quality kills 5.5 million worldwide annually," [https://news.ubc.ca/2016/02/12/poor-air-qualitykills-5-5-million-worldwide-annually/],

[2] J. O. Anderson, J. G. Thundiyil, and A. Stolbach, "Clearing the air: a review of the effects of particulate matter air pollution on human health," Journal of Medical Toxicology

[3] International Agency for Research on Cancer, "Outdoor air pollution a leading environmental cause of cancer deaths," Press Release No. 221, IARC Communications Group, 2013.

[4] A. Azid, H. Juahir, M. E. Toriman, M. K. A. Kamarudin, A. S. M. Saudi, C. N. C. Hasnam, N. A. A. Aziz, F. Azaman, M. T. Latif, and S. F. M. Zainuddin, "Prediction of the level of air pollution using principal component analysis and artificial neural network techniques: A case study in Malaysia," Water, Air, & Soil Pollution, vol. 225, pp. 2063, 2014.

[5] P. Zannetti, "Air pollution modeling: theories, computational methods and available software," Springer Science & Business Media, 2013.

[6]Epa.ie. 2021. [online] Available at: <https://www.epa.ie/pubs/reports/air/quality/Air%20Quality%20In%20Ireland%202018.pdf>

[7] M. Budde, M. Köpke, and M. Beigl, "Design of a light-scattering particle sensor for citizen science air quality monitoring with smartphones: tradeoffs and experiences," in Pro Science 3: Conference Proceedings: 2nd International Conference on Atmospheric Dust - DUST 2016, pp. 13–20, Castellaneta Marina, Italy, 2016

[8] M. Budde, L. Zhang, and M. Beigl, "Distributed, low-cost particulate matter sensing: scenarios, challenges, approaches," in Proceedings of the 1st International Conference on Atmospheric Dust, pp. 230–236, Castellaneta Marina, Italy, June 2014.

[9] PurpleAir. 2021. PurpleAir. [online] Available at: <https://www2.purpleair.com/community/faq>

[10] PurpleAir. 2021. Startup Guide for PurpleAir Sensors | Installation & WiFi Setup. [online] Available at: <https://www2.purpleair.com/pages/install>

[11] Epa.ie. 2021. Air Quality FAQ's :: Environmental Protection Agency, Ireland. [online] Available at: <https://www.epa.ie/air/quality/air%20quality%20faq/name,68463,en.html>

[12] Epa.ie. 2021. Air Quality FAQ's :: Environmental Protection Agency, Ireland. [online] Available at: <https://www.epa.ie/air/quality/air%20quality%20faq/>

[13] Epa.ie. 2021. Air Quality FAQ's :: Environmental Protection Agency, Ireland. [online] Available at: <https://www.epa.ie/air/quality/air%20quality%20faq/>

[14]/[15] Engineer, B. 2021. What are the differences between Raspberry Pi and Arduino?. [online] Electronics Hub. Available at: <https://www.electronicshub.org/raspberry-pi-vs-arduino/#:~:text=The%20main%20difference%20between%20them,the%20CPU%2C%20RAM%20and%20ROM.&text=Raspberry%20Pi%20needs%20an%20Operating,t%20need%20any%20operating%20system>

[16] Wiki.dfrobot.com. 2021. PM2.5_laser_dust_sensor_SKU_SEN0177-DFRobot. [online] Available at: <https://wiki.dfrobot.com/PM2.5_laser_dust_sensor_SKU_SEN0177>

[17] Aqmd.gov. 2021. [online] Available at: <https://www.aqmd.gov/docs/default-source/aq-spec/resources-page/plantower-pms5003-manual_v2-3.pdf>

[18] En.wikipedia.org. 2021. Matplotlib - Wikipedia. [online] Available at: <https://en.wikipedia.org/wiki/Matplotlib>

[19] En.wikipedia.org. 2021. *Internet of things - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Internet_of_things>