AirQfy

Fear Not the Particulates

Team 2 Design Portfolio

Justas Brazauskas
Kazmer Nagy-Betegh
Rene Molnar
Sabin Gheorghiu

April 2019

Table of Contents

Inquiry and Analysis

Problem statement

The need for good quality air is universal. Whether it Is an office, a gym or a pub, all human inhabited spaces eventually start suffering from a decrease in air quality. Having major implications for health and cognitive performance, clean and fresh indoor air is a major necessity, as most of the population spends their days indoor.

Poor air quality in indoor spaces is one of the hardest to detect by humans inside as our noses get used to it and we don't tend to notice a continuous decline. This can result in an undesirable slow increase of the level of drowsiness and fatigue.

Furthermore, based on EPA's indoor air quality guide [1], such decline in cognitive performance and productive results in costing the US tens of billions of dollars in lost productivity and medical bills (1989 EPA Report to Congress).

Recently built homes and buildings now have to meet the more restrictive new environmental standards. Older buildings were built in a way that fresh air could enter easily and used air could leave easily. Nowadays, this exchange of air is minimised by good insulation to keep heating and cooling costs low. In addition to the bad exchange of air, new sources of pollution from both outside (car exhausts, pollution from nearby manufacturing plants) and inside the buildings (particulates from photocopiers/laser printers, badly maintained air conditioning) further increase the need of air quality management systems. Apart from negative impacts on health, polluted and used air can reduce productivity in office environments as well.

Figure 1 Australian indoor air quality statistics

In the age where smart thermostats are used to control building heating systems, HVAC (Heating, ventilation, air-conditioning) systems remain largely temperature focused with only timer-controlled ventilation capabilities and no predictive analysis support. This is particularly noteworthy, as traditional Air Conditioning systems are consuming significant amounts of energy. Furthermore, most of the sensor feedback that current building automation systems receive is based on temperature and human presence, that is either on or off, often leaving such

systems lacking in capability to provide optimal conditions in buildings with permanent human activity.

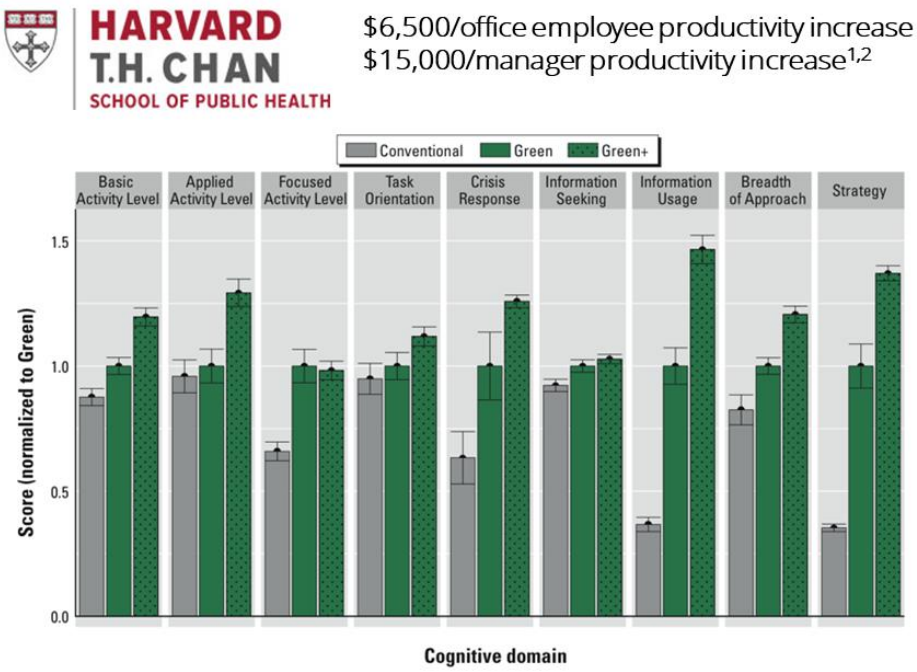

Figure 2 Harvard study of how air quality affects cognition. (https://www.enverid.com/iaq-research/air-quality-studies-office-environments)

This lack of optimisation in Air Conditioning systems, reduced control levels of building occupants and generally low regard for indoor air quality standards have been a major inspiration to solve this smart-buildings challenge.

The presented solution to this problem is a smart air quality control system with predictive analysis capabilities to control HVAC systems based on the number of people in a given space.

Solution Development

The initial approach was to only look at the air quality in the room, but it was found that reasonably priced air quality sensors have a tendency to saturate over time so it wouldn't be reliable in the long run and the pricier versions would be too big of an investment for a large building with many rooms.

From this a new approach was adopted which is to build a model of how the air quality changes with the occupancy of the rooms. The number of people in the room can be counted with a pair of time of flight sensors in the doorframes then the air quality sensor only needs to be used periodically to recalibrate the model. This approach allows for continuous course correction of the system if the conditions of the room change over time.

The model would be applied to the data in the cloud and the analytics can be accessed by the users, building managers from anywhere and the output can be linked to HVAC systems to

automatically adjust the settings to prevent the decline in air quality based on the expected change as well as maintain an efficient control on heating and cooling.

This solution would be unique as instead of being reactive it is preventive which means the air quality never declines below comfort levels whereas in current systems the quality has to be out of the comfort level for the system to adjust itself.

Market Research

Airthinkx [2] – is a product developed to offer professional level air monitoring in relatively small package with 3G/WIFI/Bluetooth connectivity to be able to form a mesh network and connect all devices to the cloud. It offers data visualisation and analytics tools. It is a compelling product offering and would be our closest competitor. The devices price is higher than our solution and its sole focus is not to maintain air quality but monitor it.

Awair 2 [3] – is a more consumer focused air monitoring system aimed at houses more than large buildings, but it works in both. It lacks the easy scalability of the AirQfy solution or the one found in *Airthnkx* (although it is cheaper than the latter).

Dyson Purifiers [4] – these purifiers are paired with cooling and heating function. They are an easy plug and set solution for small offices and homes and can be seen as a far competitor as it offers an all in one solution for air quality and comfort issues in a space, but it is expensive to scale and coordinate on a large level, so in the smart building space that our solution aims to enter they don't seem like a strong competition as office buildings, libraries and other large indoor spaces usually have ventilation systems built in. They just need a better control system to make them more effective.

Aeroqual Series 900 [5] – Indoor air quality monitoring companies that asses the health and safety standards of a workspace and then give design recommendations. These companies don't seem like a big threat, there is even a potential for a partnership.

All of the competing solutions fall in the categories of the above-mentioned examples. As mentioned, the strongest competition comes from the first company.

Table 1 Similar products comparison

	Airthinkx	Awair	Dyson	Aeroqual	AirQfy
Price	699£ or 49£/month	149£	300£ - 500£	500£ – 1200£	5£ - 10£ / month + 60£ hardware cost
Analytics	Available	Only data visualisation	Not available	Logging only	Available
Compatibility	High	Low	None	High	High
Expandability	High	Low	Low	Low	High
Scalability	High	Medium	Low	High	High

Based on Table 1 the solution seems to have a place in the market as it serves the need for an expandable air quality monitoring system and it doubles as a building occupancy monitoring solution.

Design Brief

Problem

The problem is the impact of bad indoor air quality on cognition, productivity, comfort levels and health. This is an issue faced by closed spaces where large numbers of people spend several hours, examples of this are open plan offices, libraries, lecture theatres.

Objectives

Develop a solution to easily monitor air quality in indoor spaces and predict the expected decline in air quality before it happens so it can be counter acted, and the air quality can always be maintained at an optimal level.

Key requirements of the solution:

- Feasible cost
- Low power consumption
- Easy access to information about all the rooms in a building
- Reliable for many years and non-intrusive

Target users

Our main target users are offices, universities, libraries and building management companies. In a wider context any indoor space that wants to improve its indoor air quality (gyms, bars, pubs, restaurants etc.) could be a potential customer.

Scope of the project

- Correlation of air quality with the number of people in a given room
- Live occupancy counting in rooms
- Live air quality estimates in rooms based on occupancy
- Cloud based data storage and analytics for universal access

Developing Ideas

Design Specification

Function

- Product must be able to present live (real time) air quality estimates and occupancy data
- Intuitive UI (User Interface)
- Standardised output that can be easily connected to HVAC systems. This means that in the future it would be easy to automate system even more and also to integrate and retrofit into already existing systems.
- Product must be able to offer remote access to facility managers
- Product must store data in the cloud

Performance

- Accurate in counting people (less than 2% error rate)
- Low power consumption enabling connection to any power line without much effect on the added consumption (0.5 Watts maximum). The low power consumption and potential for further power reduction (through optimization) can make the AirQfy system work on batteries where mains power is unavailable or this connection would be invasive.
- Latency of the data showing up in the user interface should be below 30 seconds

Cost

- Hardware must be cheap, so it is financially feasible to scale (below 100£/unit)
- Software and service provided must be priced low, to make it cheaper than developing one's own solution, and to make the service competitive. (below 25£/month)

Installation & Maintenance

- Simple installation: just supply power and enable connection to local WIFI network.
- Product should function as intended for 5+ years without regular hardware servicing
- General cleaning of the place shouldn't influence the functioning of the product
- In case of a power cut the product should reboot on its own

Appearance & Size

- Small: easy to hide in door frame (max. 20mm x 20 mm)
- Sleek design, possibility to cover with same material as the wall/doorframe it is placed in.

Feasible design ideas

The initial design idea was to use only air quality sensors. This was discarded after finding that reasonably priced sensors tend to saturate and become inaccurate after a few days, which is undesirable in our system.

Building onto this the next step was to corelate the number of people in the room to the air quality. This would allow to build a model for a room of how the air quality changes with occupancy. This is a more reliable solution as the expected change of air quality can be predicted before it starts occurring based on number of people in the room. And the air quality

sensors would only need to be used periodically to update any changes to the model. For the purpose of the update action the saturation is negligible.

For developing the people counting, the approach was to use sensors mounted on door frames that can detect if someone passes through the door. To do this, a 2-beam implementation is needed. The system would work as follows: with the beams next to each other a person passing through would interrupt either of them first, then both of them a short brief and finally the first beam would not be triggered anymore. The exact time constants used to detect a valid passthrough would then be obtained experimentally.

The first approach to 2-beam system was to use ultrasound. This set-up failed the people test, even after rigorous tuning and debugging. A different pair of sensors was used next: LASER TOF (time of flight). These proved to be reliable enough for the application.

Design Justification

Looking at the above requirements, an approach using time of flight sensors for people counting and the air quality sensor only for model creation seems to meet the design criteria. The solution is cheap, reliable and easily scales by adding additional components to the network.

The time of flight sensors performed significantly better than the ultrasound ones, effectively picking up directional passthroughs with a 98% success rate after thorough calibration. It was tested by moving slowly through the beams as well as at running speeds, different distances (within reasonable size of a doorframe i.e. 2 meters) and different outfits on people (that would potentially have different reflectivity). In all scenarios the sensors provided a good performance.

The chosen cloud platform was IBM Bluemix as it had the best pricing model for the intended application allowing for low costs. IBM also had convenient integration via MQTT with IOT devices in NodeRED. Other Cloud platforms we considered were offered by Microsoft, Amazon and Google. However, they lacked IOT integration being more database oriented.

Solution Development

As stated above, the initial design idea was to use air quality sensors to directly measure the air quality. This would enable direct control of ventilation. However, these air quality sensors either proved to be financially unfeasible or they did not provide sufficient accuracy because of noticeable sensor drifting. Therefore, an approach was selected where the air quality was determined indirectly by determining it from the room occupancy. This is an ideal solution for rooms that already have some pre-installed occupancy detection like entry gates in libraries or attendance recorders in offices. For most of the cases where the customer does not have any pre-installed occupancy monitoring or it is not accurate enough, we tried to develop a suitable device for this purpose.

Our first idea was to use two ultrasound sensors placed in line next to each other. If this sensor setup was placed into or close to a doorframe, it was able to detect whether people entered or left a room by determining which sensor (either left or right) was trigger first. The HC – SR04 ultrasound sensor was used for this purpose because it is a commonly used sensor with a very low cost (approx. £1.00 on Amazon – price from 23/04/19). Due to this attractive pricing, there is a lot of documentation available online. Both ultrasound sensors were then connected to an Arduino UNO. This microcontroller board was used because it is easily programmable, and it

provided the required input and output peripherals. Figure 3 shows the final hardware setup of the sensors.

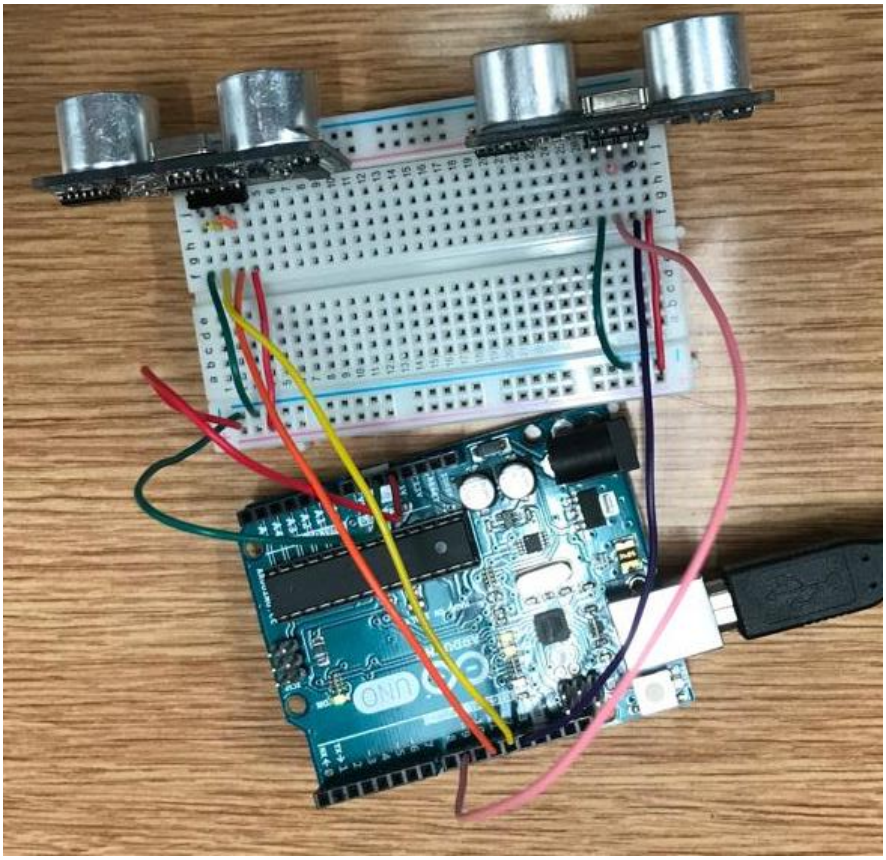

Figure 3 Ultrasound sensor (HC-SR04) setup while testing

The sensors were programmed in a way that they only registered a person entering or leaving if he passed both sensors within a specific timeframe. This helped improving accuracy and reduced wrong measurements. The Arduino code for the ultrasound sensors can be found in the appendix. The ultrasound sensor setup was able to register the direction of movement for objects with an acceptable accuracy. This was tested by setting up the sensors on a table while pushing a toy car through the ultrasound gate. Even with varying vehicle speed, the accuracy did not drop below a critical level.

However, when using the sensors to determine the direction of movement of people, the system accuracy significantly dropped. Even after extensive tuning and some modifications, the accuracy could not be improved to provide a sufficient performance (on average success rate was 60%). It is assumed that this was due to the combination of several reasons. First, the distance between the sensors and the moving object was now greater than with the toy car. Additionally, the reflectivity, the shape and the movement of the people created unwanted reflections that were interpreted in the wrong way by the sensors. This led to wrong measurements. As a result the ultrasound sensors were replaced by TOF (time of flight) sensors. It is expected that these sensors provide better performance because they have a harrower beam and they use a different frequency in the electromagnetic spectrum.

Creating and Demonstrating the Solution

Sensors

Time of Flight sensor **VL53L0X**

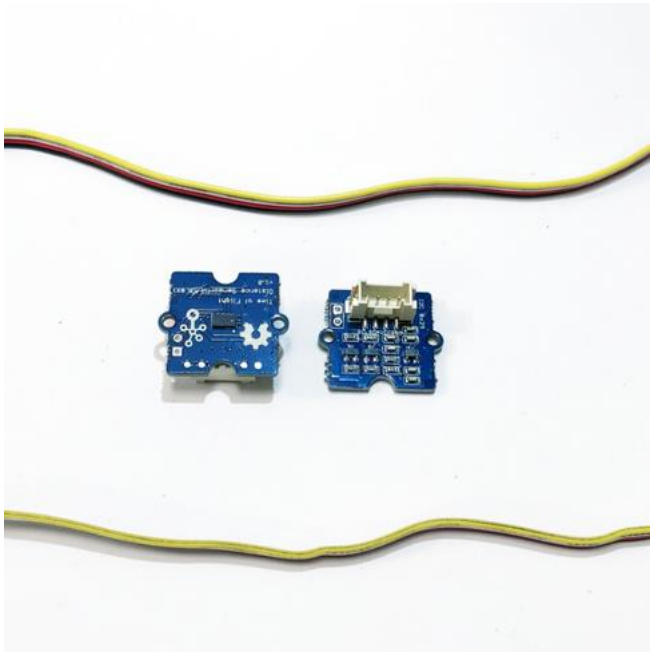

Figure 4 Time of Flight sensor (VL53L0X)

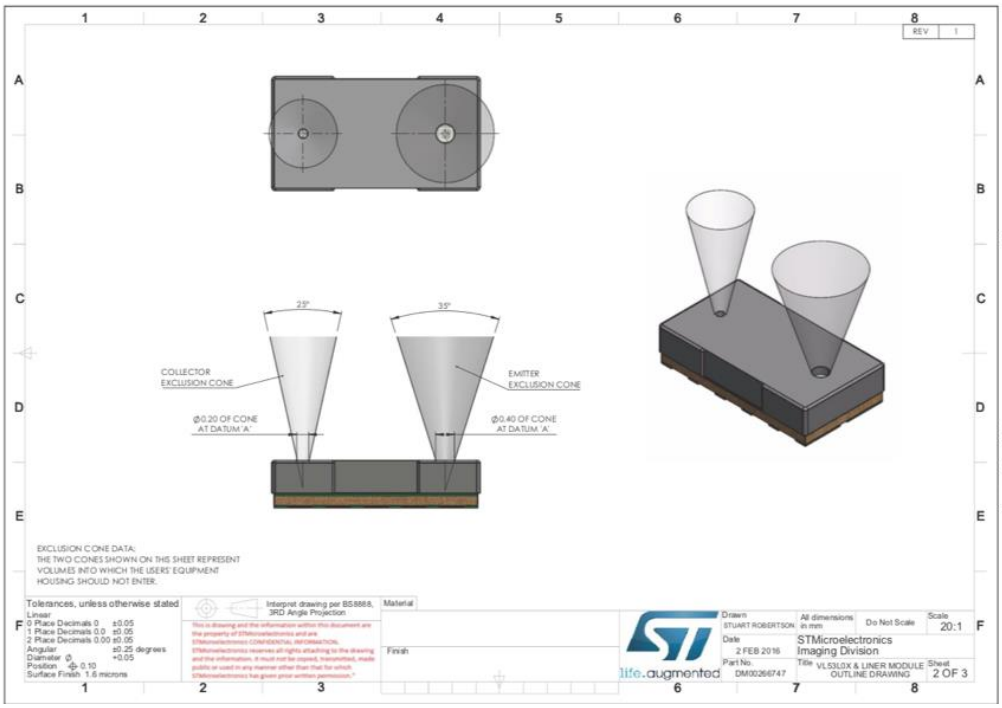

Figure 23. Outline drawing (page 2/3)

Figure 5 Illustration of sensor shape and set up.

The time of flight sensors where used in pairs connected to an ESP8266 microcontroller. They were placed in parallel with each other and this way the direction of the motion could be detected, by looking at which beam was interrupted first.

Figure 5 is a blown-up version of the sensor that can be seen in Figure 4 in the middle of the chip on the left side of the picture.

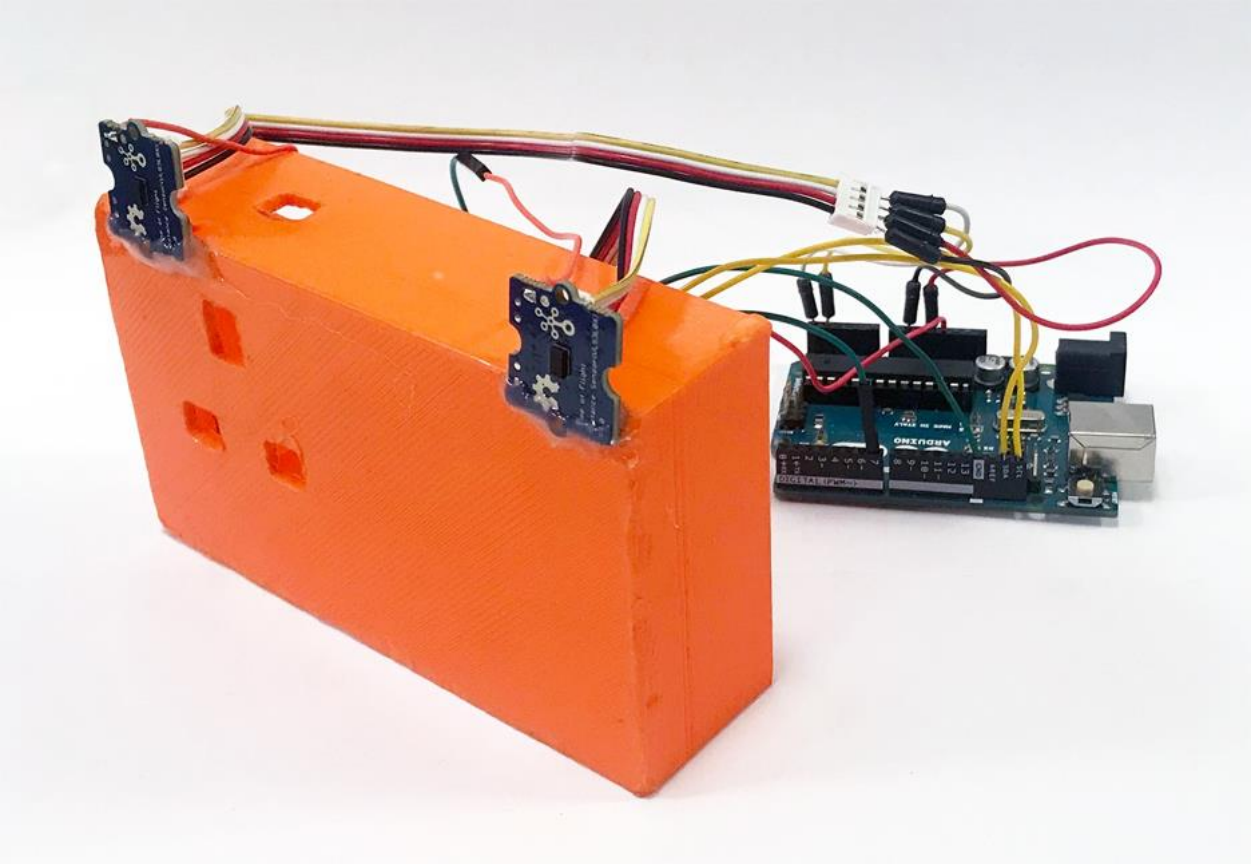

Figure 6 Time of Flight sensor application

The VL53L0X is currently the world's smallest commercial time of flight sensor which allows for a integrating it in a small unit for seamless deployment.

Key features utilised:

- Eye safe
- Fast, accurate distance ranging
  - Measures absolute range up to 2 m
  - Advanced embedded optical cross-talk compensation
- Easy integration
  - Single power supply
  - I2C interface for device control and data transfer

Based on these, the sensor fits the applications requirements stated in the specifications.

Table 2 Consumption at ambient temperature

Parameter	Min.	Typ.	Max.	Unit
HW STANDBY	3	5	7	uA
SW STANDBY (2V8 mode) (2)	4	6	9	uA
Timed ranging Inter measurement		16		uA
Active Ranging average consumption (including VCSEL) (3)(4)		19		mA
Average power consumption at 10Hz with 33ms ranging sequence			20	mW

1. All current consumption values include silicon process variations. Temperature and Voltage are at nominal conditions (23degC and 2.8V).

All values include AVDD and AVDDVCSEL.

2. In standard mode (1V8), pull-ups have to be modified, then SW STANDBY consumption is increased by +0.6uA.

3. Active ranging is an average value, measured using default API settings (33ms timing budget).

4. Peak current (including VCSEL) can reach 40mA.

It is low power as well based on Table 2, which is great as it meets the power consumption requirement as well.

Microcontroller **ESP8266**

The microcontroller has an integrated WIFI unit and 32bit CPU that can be used for running applications. This board was ideal for connecting up the time of flight sensors to the cloud and to run the decision algorithm to check if someone is entering or exiting the room. Then the microcontroller would send a signal to the cloud about either instance. [6]

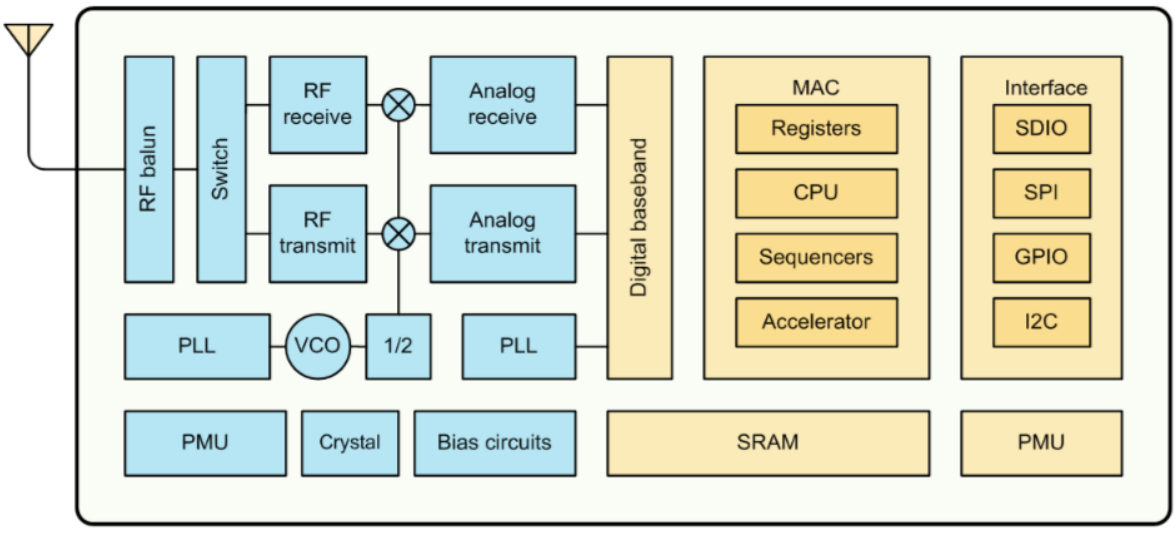

Figure 7 Block diagram of microcontroller

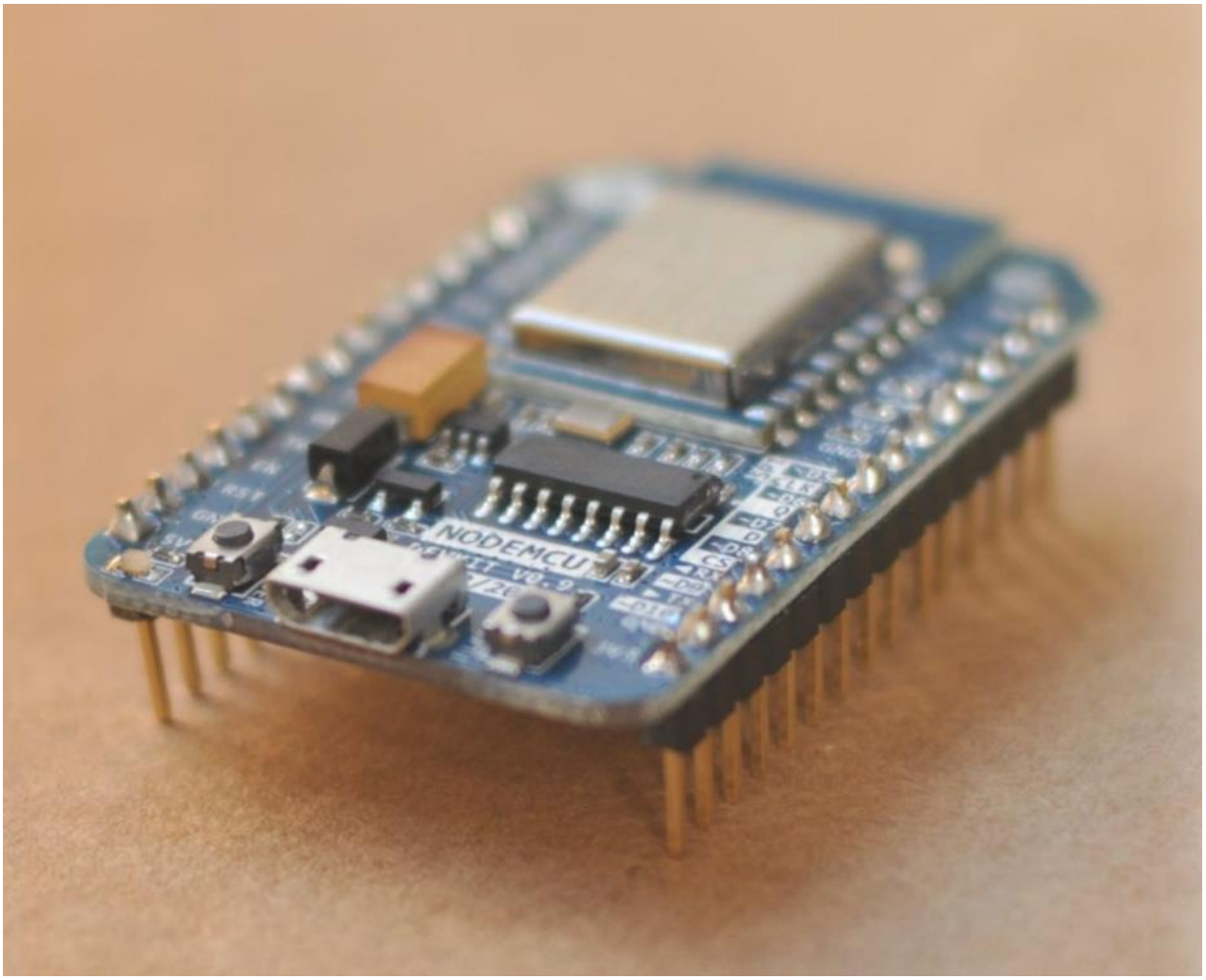

Figure 8 Node MCU board picture

Key Features utilised:

- Integrated low power 32 – bit MCU
- WIFI 2.4 GHz support WPA/WPA2
- Deep sleep power <10 uA, power down leakage current <5uA
- Wake up and transmitter packets < 2ms
- FCC, CE, TELEC, WIFI Alliance, and SRC certified
- SDIO 2.0, (H) SPI, UART, I2C, I2S, IR Remote Control, PWM, GPIO

Based on the above features the microcontroller was ideal for easy and reliable WIFI connection set up. The less than 2 milliseconds wake up time from the low power mode allows for low power consumption and to only wake up the controller for sending when the time of flight sensors transmits information through the I2C connection.

Air quality sensor **Grove v1.3**

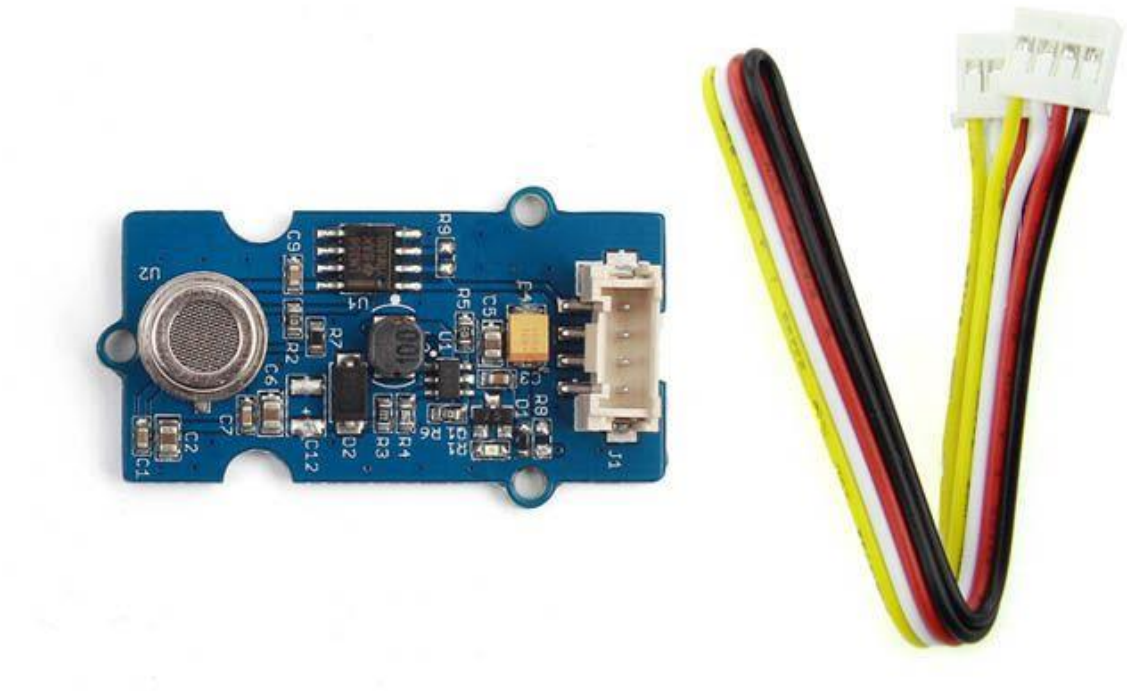

Figure 9 Air quality sensor and board, Grove v1.3

The air quality sensor chosen was a cheap device, with quick response to changes in air quality.

The key features of the device:

- Low power consumption
- High sensitivity
- Tiny outline
- Responsive to wide scope of target gasses
- Durable
- Compatible with 5V and 3.3V

The device was designed with indoor air quality monitoring in mind. Because of its cost (9£) and responsivity it was a good choice.

The only main drawback of this tiny sensor is that after about 2-3 days of use it starts to exhibit drifting in its readings as it tends to saturate if it is not turned off periodically. But this period of continuous data collection was good enough to develop a reliable model for a room about air quality and occupancy correlation.

Air quality data collection **ARDUINO UNO REV3**

Figure 10 Arduino Uno R3

The board to was chosen along the sensor because it is a well-documented easy to use board and allowed for quick set up. The air quality data read through the sensor was saved on an SD card connected to the board, this data was then used to develop the model.

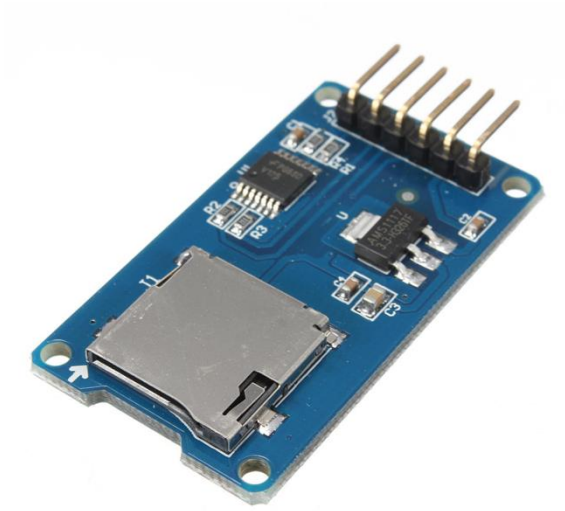

Figure 11 SD card SPI adapter for Arduino

The module (MicroSD Card Adapter) is a Micro SD card reader module, and the SPI interface via the file system driver, microcontroller system to complete the MicroSD card read and write files. Arduino users can directly use the Arduino IDE comes with an SD card to complete the library card initialization and read-write.

Thanks to the extensive Arduino libraries the deployment was quick.

It was crucial to get this part done as quick as possible because sufficient air quality data was crucial to the project and safety forms had to be completed before data collection.

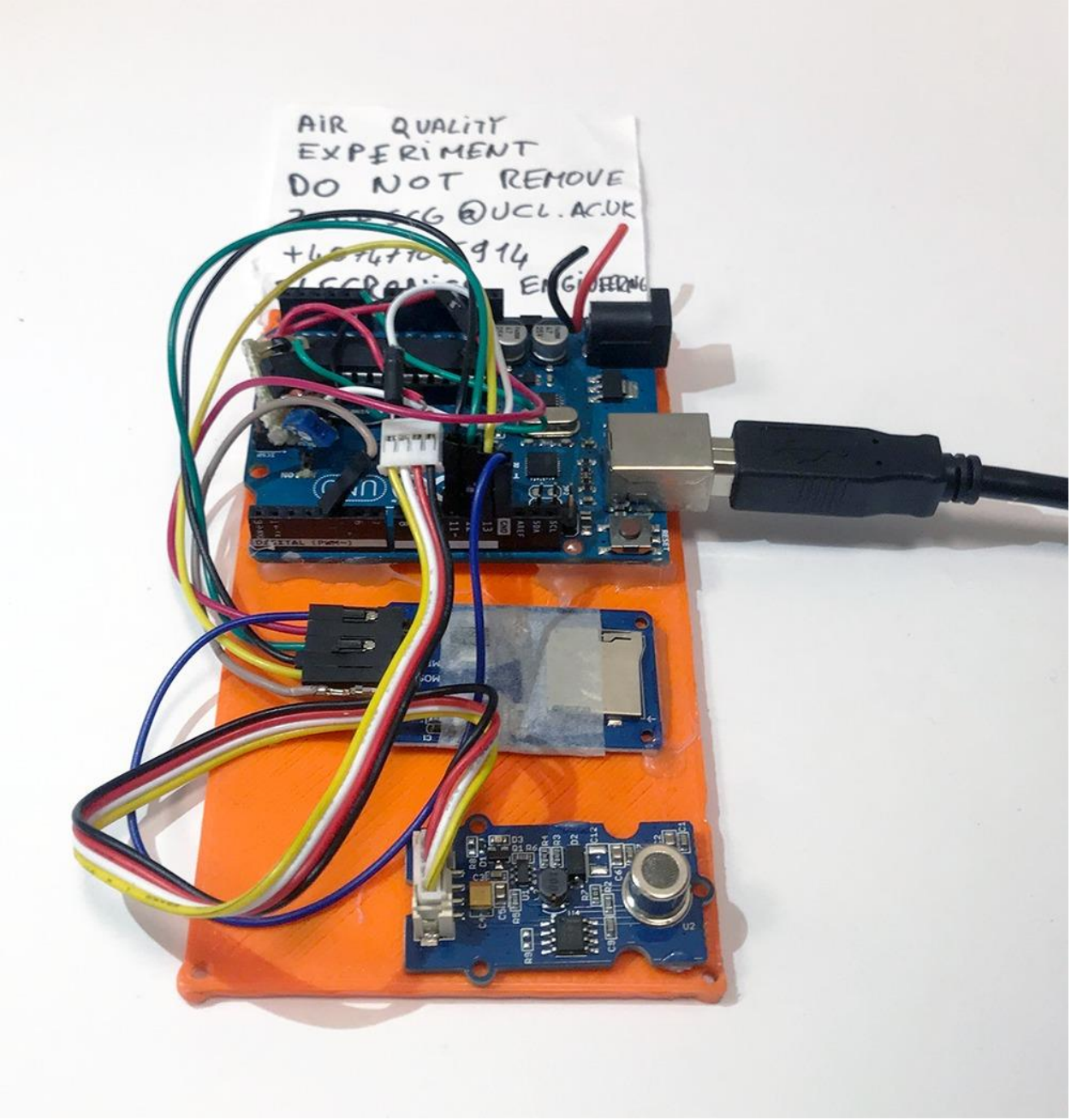

Figure 12 Arduino board connected to the air quality sensor and SD card reader

Cloud

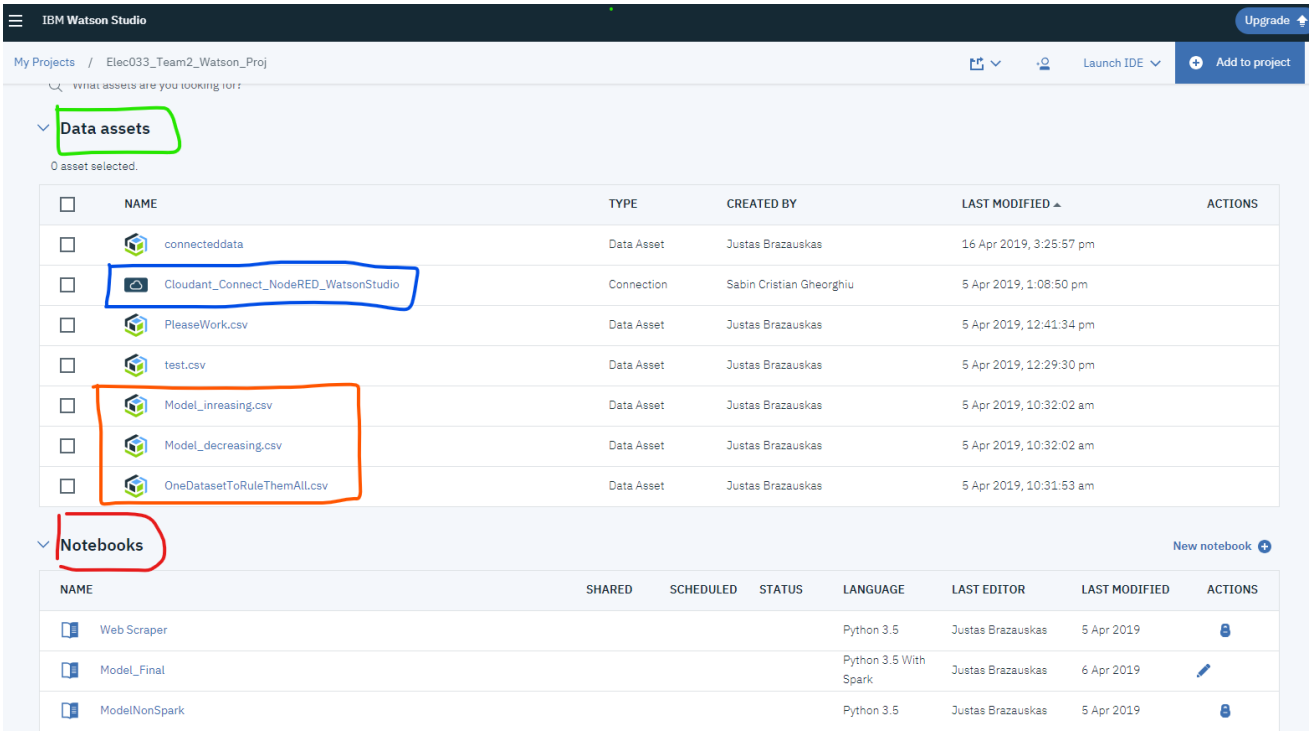

Figure 13 IBM Watson Studio presenting the Data Assets (in green square), Cloudant Database Object (in blue square), Created Databases in CSV format (in orange square). Python notebooks are presented below the assets (red square)

The Watson Studio (Figure 16) was used as the platform where all the raw data collected on air quality and number of people were further processed and later put together to obtained the regressed functions.

Presented below (Figure 14) is the final version of the UI for our system. It has gauges for estimation in air quality as well as for the real time people counter. Graphs are also shown for better visualisation. The button in the top left represents the state of the actuator (the fan). The fan will be automatically set to on when the system detects a tendency of air quality below a threshold. The fan can also be manually toggled by the system administrator (manual override). The UI also has a 'trend space' where the text updates real time in order to provide the user information about the air quality predictions in the near future. The rating is a value from 0-100 and represents how good the air quality is in relative terms. Please see table.

0-20	20-40	40-60	60-80	80-100
Very Poor	Poor	Good	Very Good	Excellent
A room that has not been aired in a very long time. (Extremely crowded library rooms=13)	A room in which working is becoming very hard (6^{th} floor after 3 hour lab yields 32)	A regular room on campus when it is half full (7^{th} floor lab average is 47)	A room that is properly ventilated (dorm room yields 77)	These levels represent the lowest level of particulates. Regent's Park level is 85

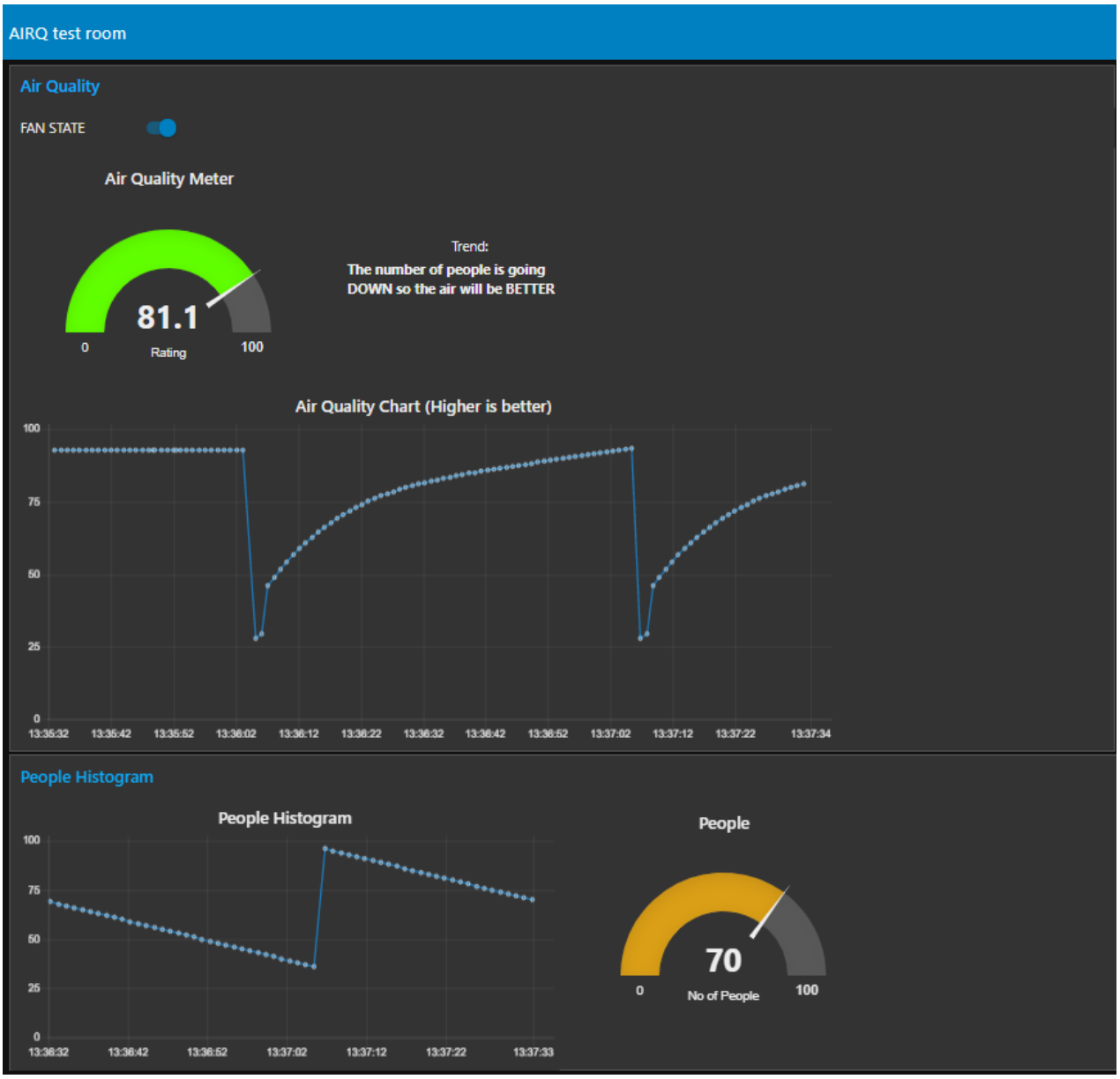

Figure 14 The final UI for AirQfy

IBM Bluemix and the NodeRED boilerplate (Figure 15 and Figure 16) were used in conjunction to enable connection and interaction in cloud with the hardware devices (light gate sensors and actuators). In the code presented 2 light gates send the number of people that have passed through (the total obtained in a set period of time (in this case 5 seconds) is given – each end device doing the intermediate calculations separately (top left of the image in the red squares). This is poured in a global variable of the flow. Another function that is triggered separately (“in room”) cumulates the information to obtain the number of people in the room. This number is compared to the average of the last 20 values obtained (last minute). According to this comparison the algorithm now chooses what path and estimation to calculate. The number of people can either be increasing or decreasing. There is a minimal threshold of difference

between average and value before the previous path of estimation is changed (avoiding jumps in estimation data). Once a path is chosen the text is updated and the estimation will output a variable that updates across the UI (gauges and charts). Different colours are set to suggest the occupancy and air quality. In the example given in Figure 14 the air quality is now good but the number of people is still quite high. The fan was triggered but as people have been leaving the room the air quality has been brought back to a good standard.

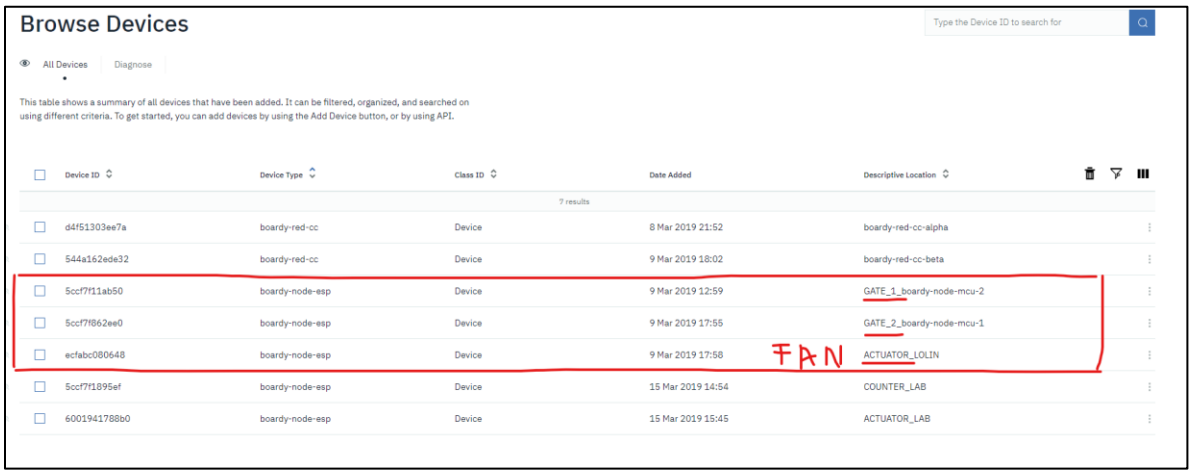

Figure 15 IBM Bluemix showing the registered devices. The 3 ESP8266 MCUs used in 2 light gates and the actuator (Fan) are outlined by the red square

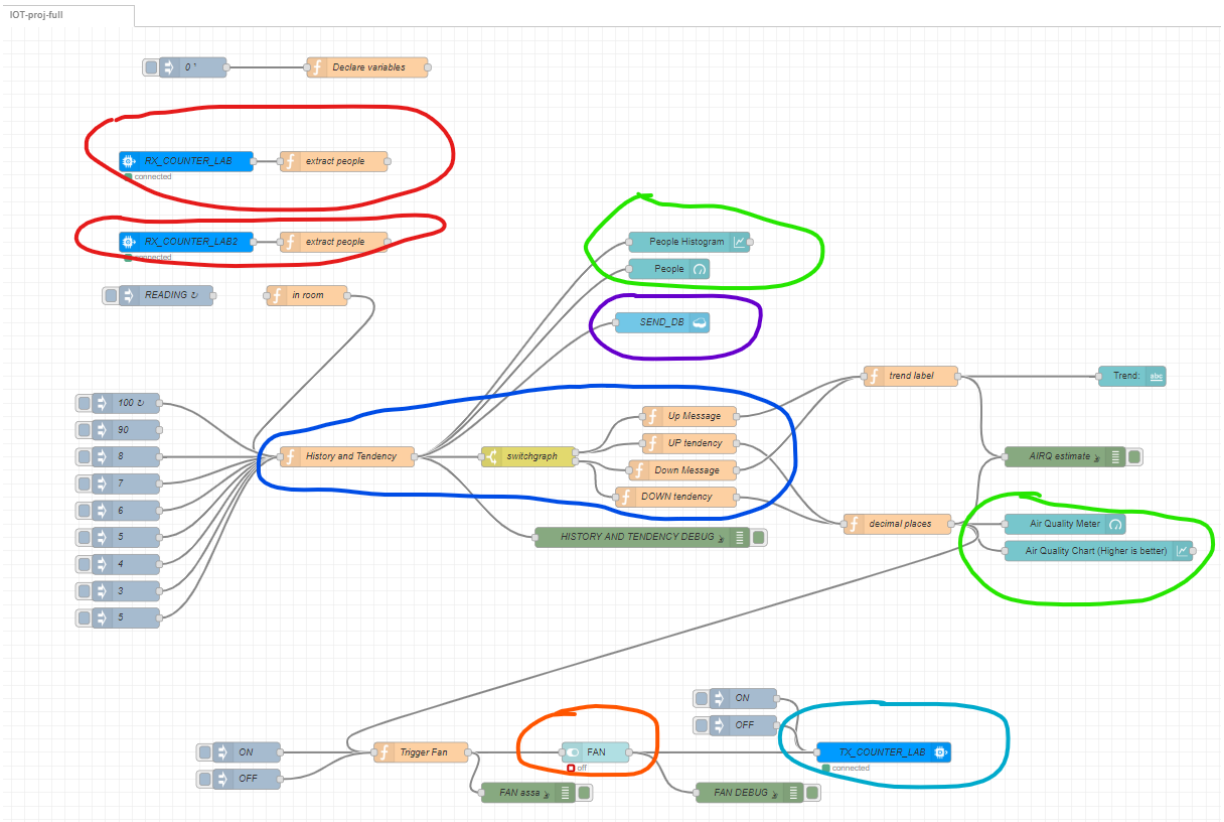

Figure 16 The NodeRED editor showing all the nodes used. IOT device interfaces are circled in red, UI interface nodes circled in green, Function calculation for the Air Quality approximation in blue.

Data Collection

Data Collection came in two phases. The first part was collecting data via the means of a web scraper. The second phase was creating a dataset from values obtained with hardware sensors.

Web Scraper Data

The first testing phase was conducted in the UCL Science Library (4th Floor). Because of technical difficulties with counting the number of people entering and leaving the study space, it was decided to first gather the air quality data and then correlate it with the information that is uploaded on UCL library seating availability website. To collect all the data from the website automatically a web scrapper coded in Python.

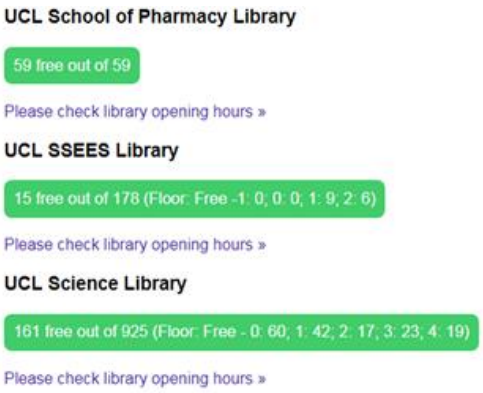

Figure 17 UCL Library Website presenting the available seats divided by zones. This website was scraped to obtain the values automatically. (https://www.ucl.ac.uk/library/libraries-and-study-spaces/available-study-spaces)

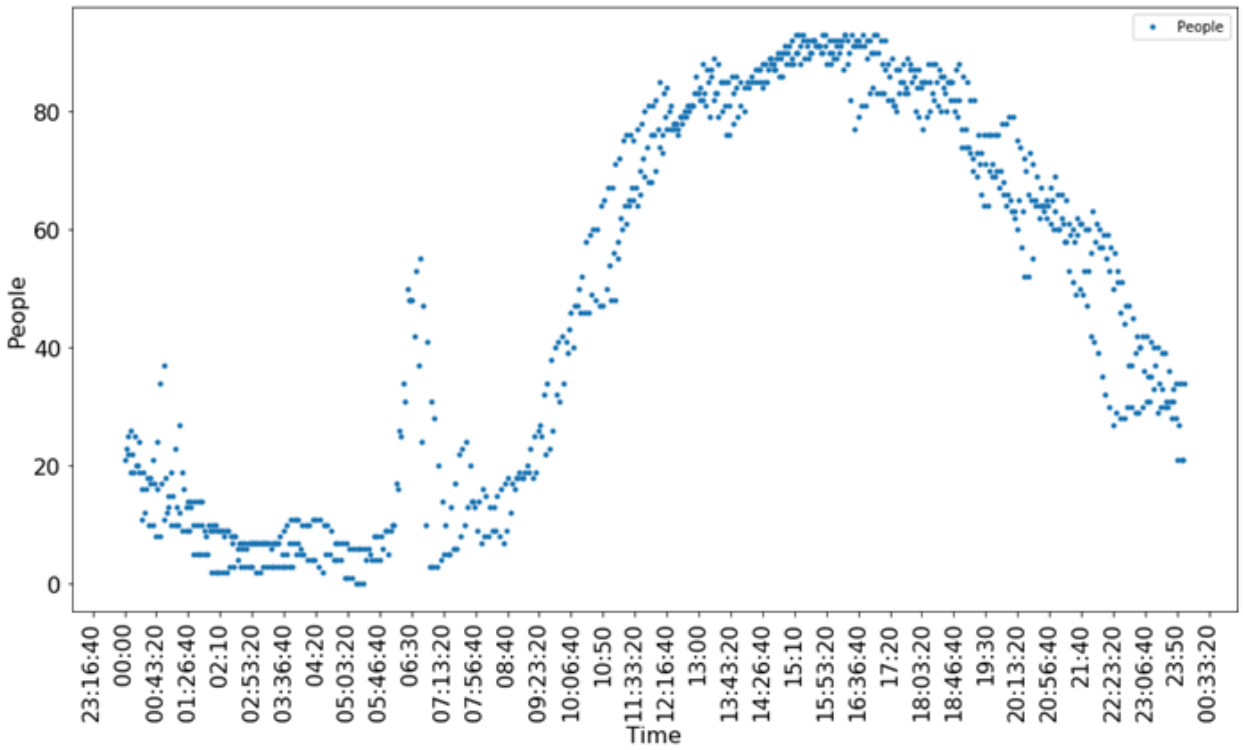

Figure 18 Data Superimposed from 72 hours of scraping. The data was scraped every 5 minutes.

Air Quality Data

The Gathered Air Quality had posed some unexpected challenges in the form of showing a clear drift pattern throughout several days red line on graph (Figure 19). To find the drifting trend, two points from the (red) dataset were taken – one from the first saddle point and one from the second one. These points represent the minimum values recorded in the period of 48 hours, most likely to have been sometime at night. Afterwards, every point on the red graph was shift multiplied by a negative of the linear function that that was passing by two points, to cancel the drift. The blue datapoints clearly show how the drift was completely cancelled, making the dataset usable.

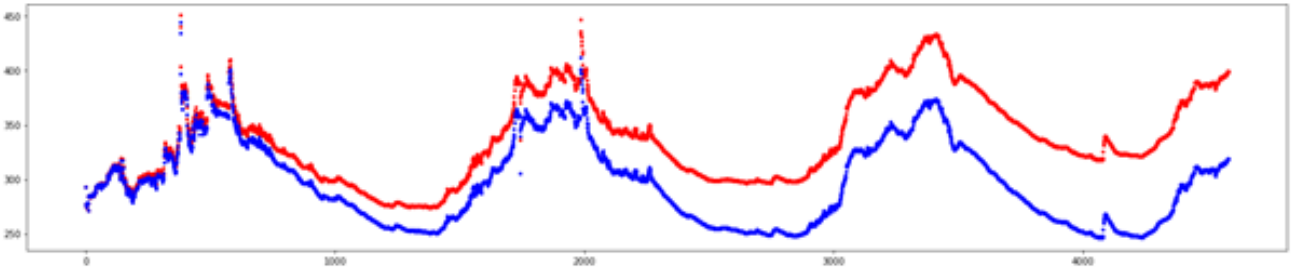

Figure 19 Graph showing 3 days of measurement - the initial sensor values shown in red present the air quality sensor drift. The blue is the shifted (normalized) graph

Lastly, the air quality data was divided into three parts each representing 24 hours of data collection. Afterwards, the three datasets were superimposed and plotted as a single 24-hour timeframe showing major outliners as well as the clear tendency of the graph. the rising slope is much more abrupt and presents harsher peaks. With the decrease of particulates, the peaks are no longer present.

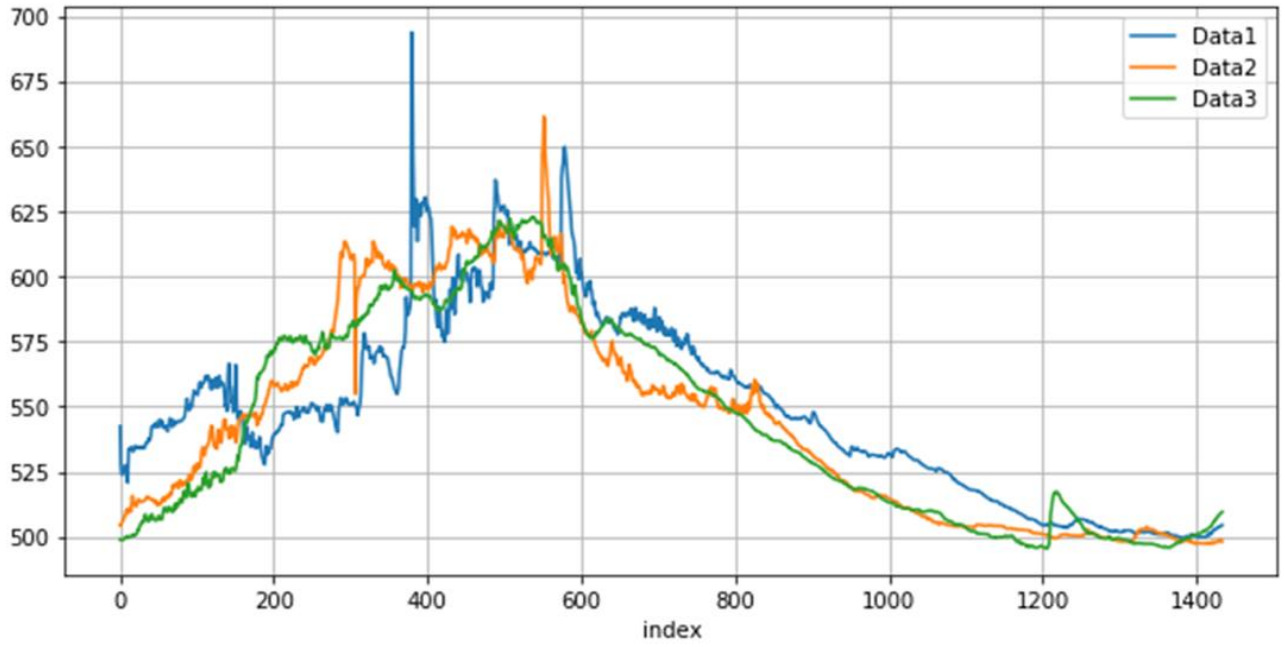

Figure 14 All the air quality data collected now superimposed.

The plot above illustrates all three days of collected air quality data superimposed. Even including the outlier-heavy data from day one did not seem to have drastic implications on the model building, as it will also be shown in the Machine Learning section.

Machine Learning Techniques

Machine learning models used polynomial regression. The code was implemented using Scikit-Learn and Python notebooks.

Fitting the “People in the library” dataset.

Different polynomial functions were tested to see which one was the best suited to fit the data.

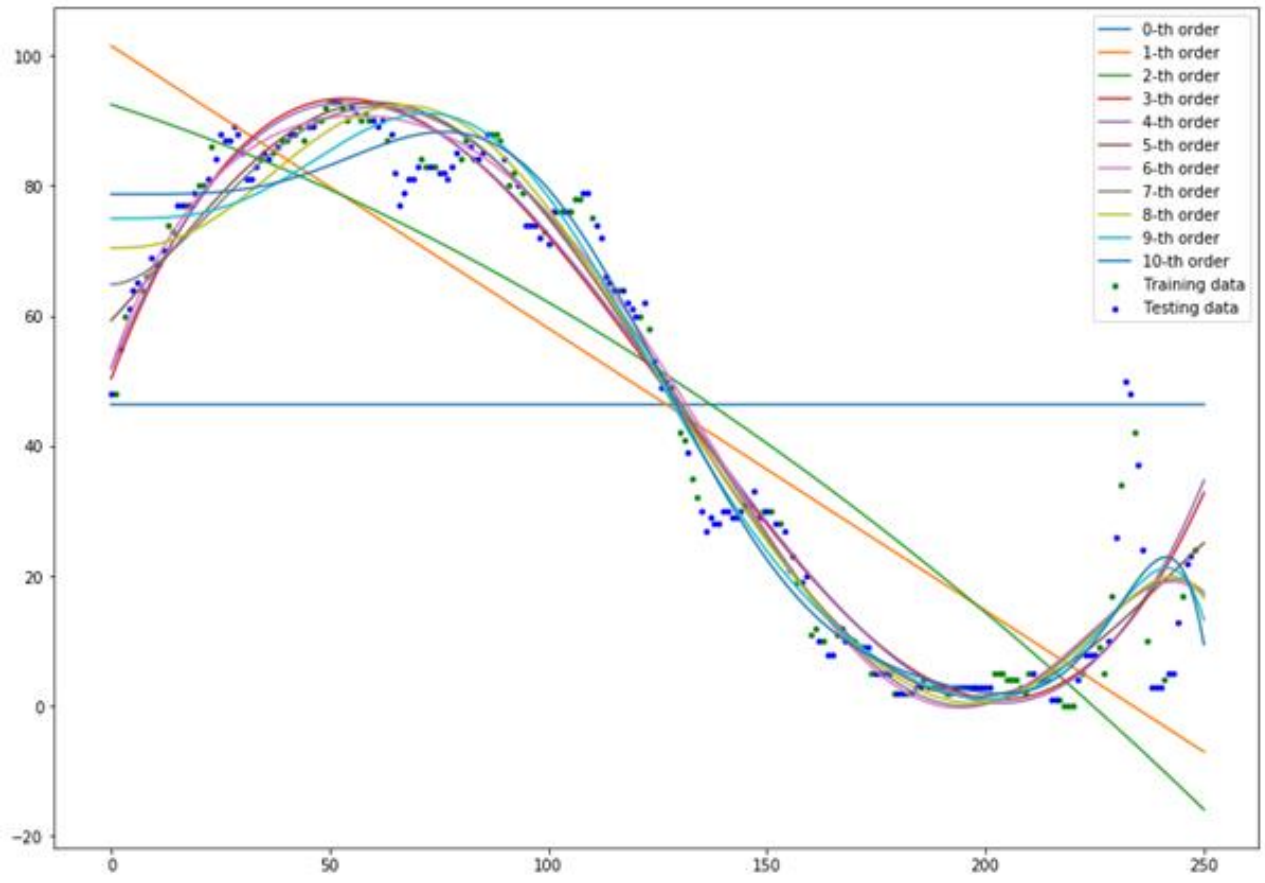

Figure 20 Different polynomial functions being tested for model fitting

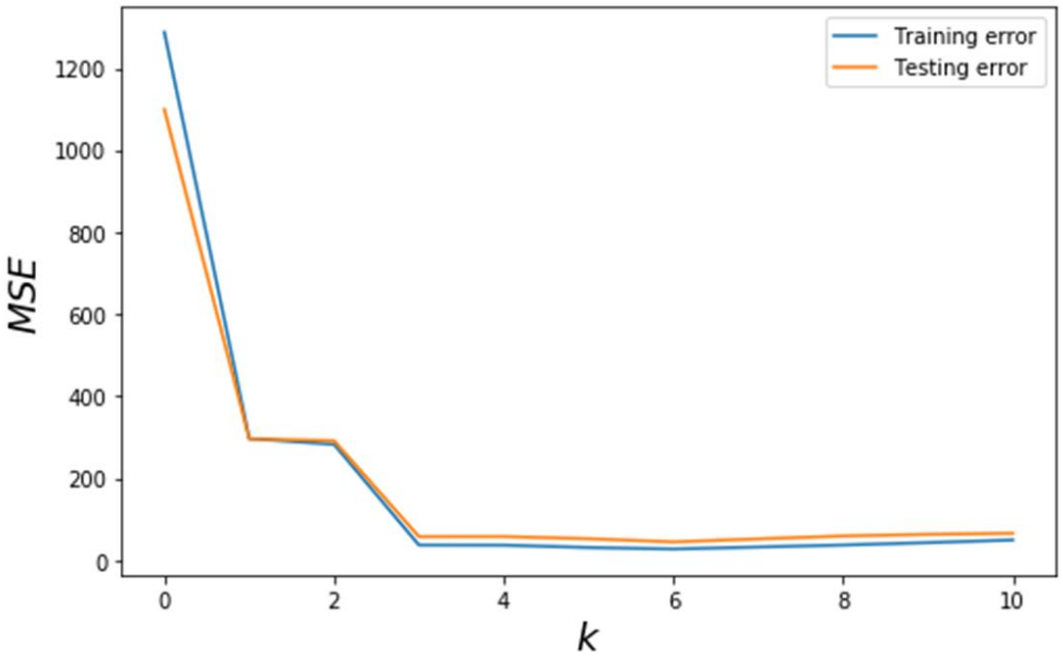

Figure 21 MSE error rates

MSE results have shown that the optimal result was around the sixth degree.

Fitting the air quality dataset.

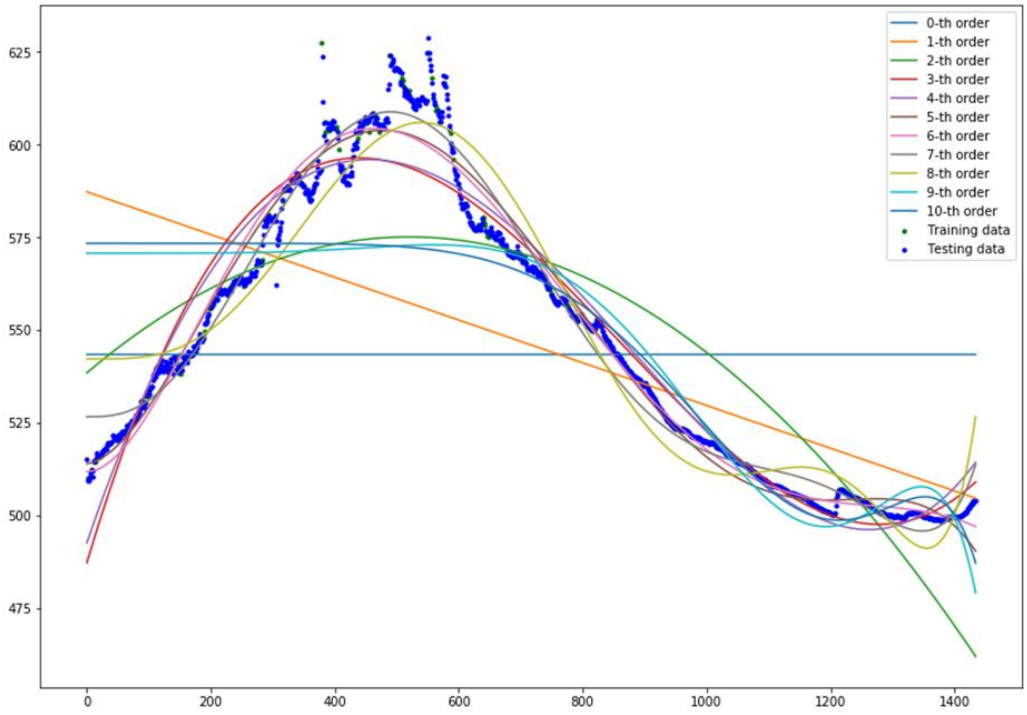

Figure 22 Different polynomial functions being tested for model fitting

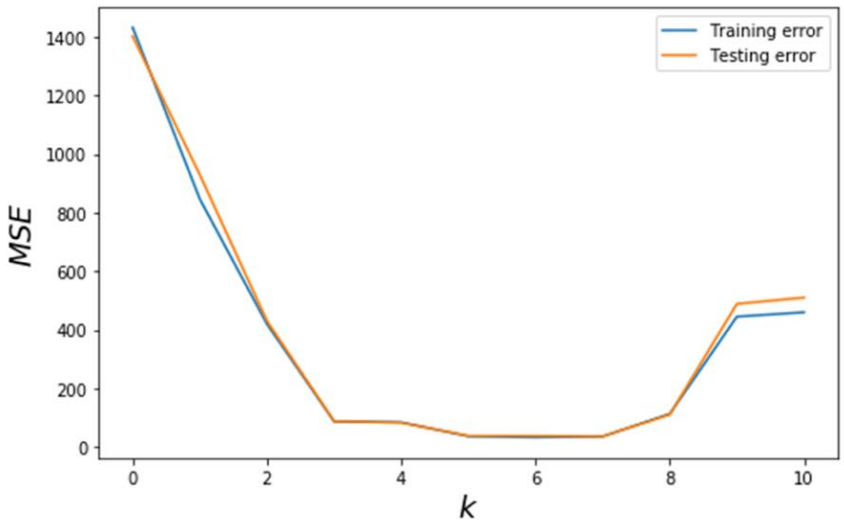

Figure 23 MSE error rates

MSE results for the air quality model also indicate that the optimal result was around the sixth-degree polynomial function.

The fitted graphs:

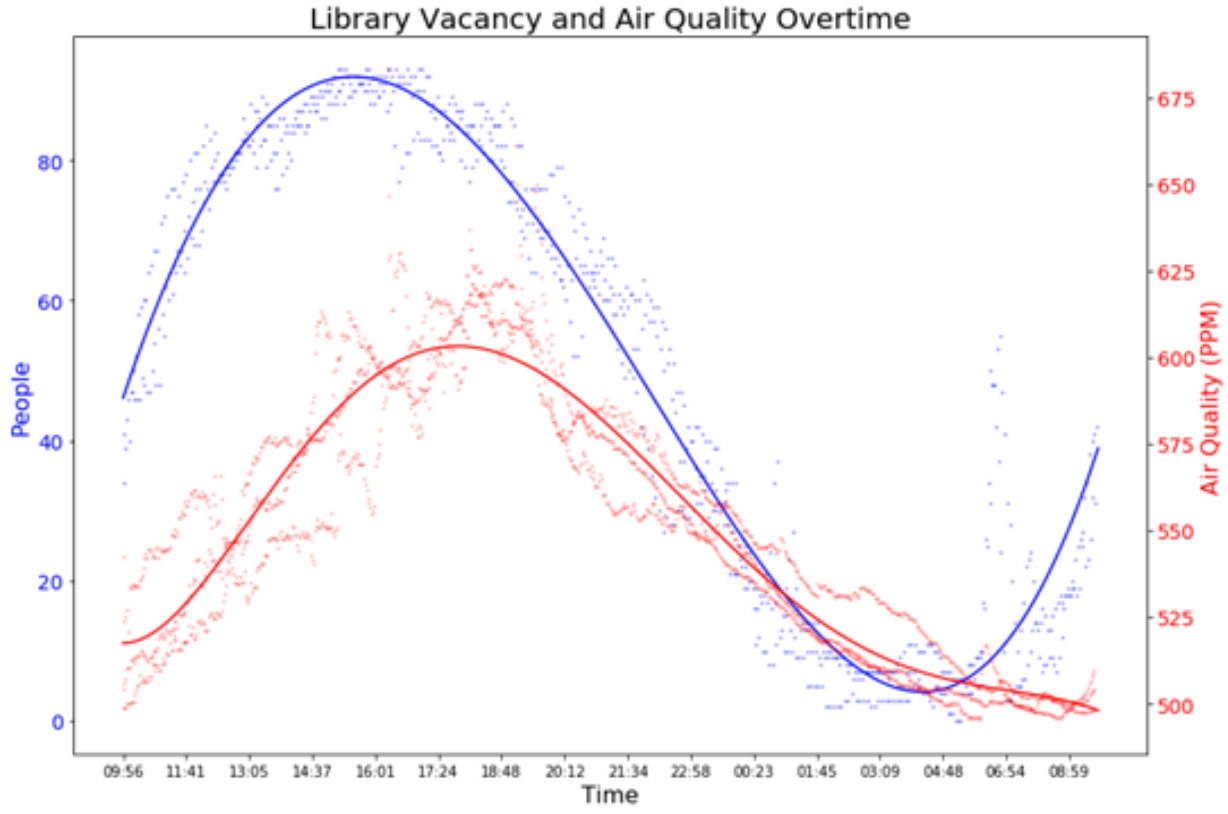

Figure 24 The number of people in the library and the correlation to the air quality. Blue dataset is the one for the rising number of people and Red dataset represents the air quality as people start leaving the rooms.

The models here represent two different datasets, both of which share the same x axis. Interestingly, the correlation between the number of people and a decrease in air quality is already visible here, as illustrated by the shift between the peaks.

However, for our project to work, we aimed at creating a model that can predict the air quality based on the number of people. Therefore, a third model had to be created, based on joining the two datasets together, eliminating the time factor.

Hysteresis plot

There was a particular reason why at the beginning of this documentation report some attention has been given to creating timestamps. Timestamps were a unifying variable that had helped to create our final model.

Both datasets were joined on unifying timestamps, creating a final dataset to be used for the creation of our model.

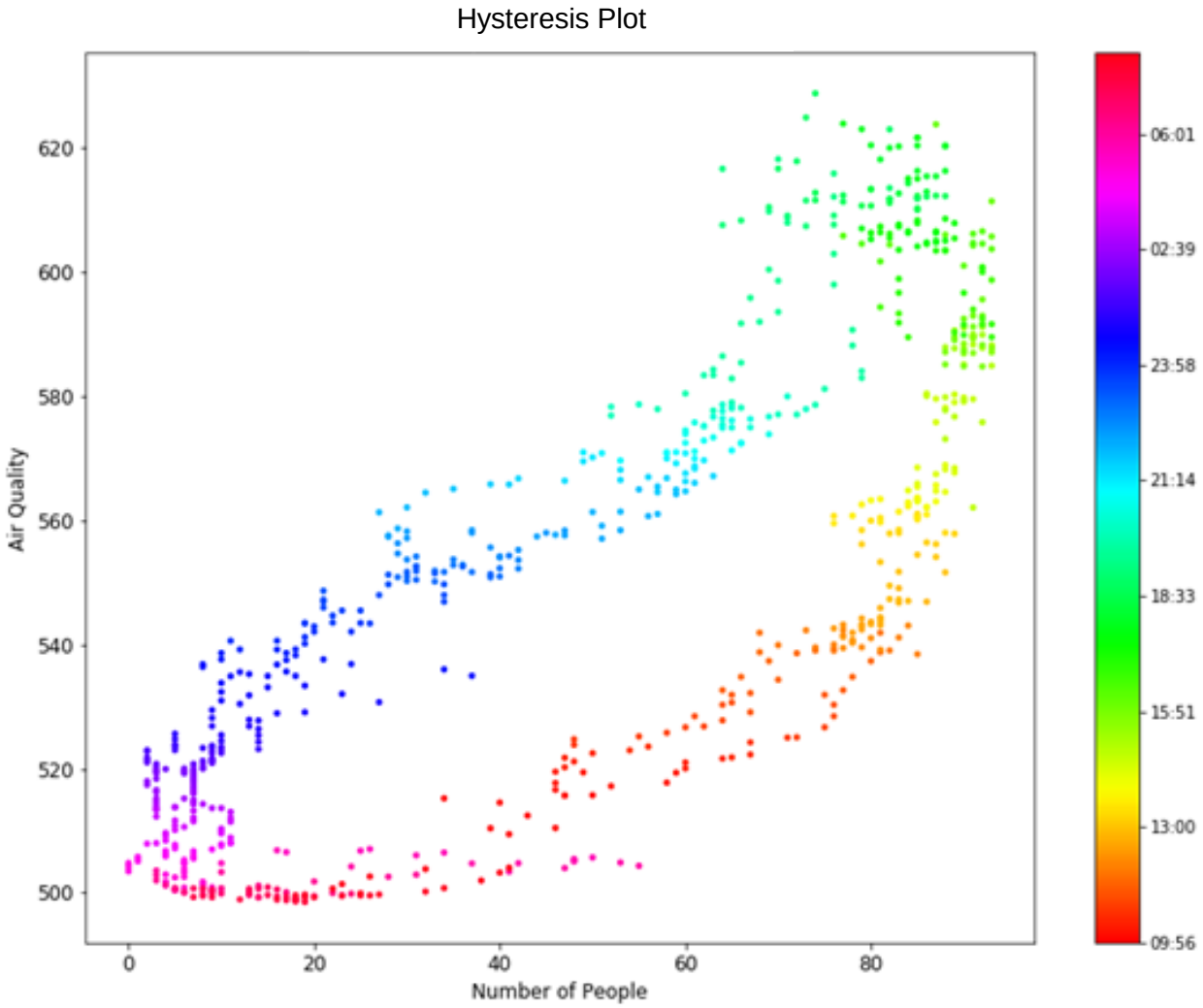

Figure 25 Both the datasets were joined on unifying timestamps and the hysteresis plot created.

Having this data, it was finally possible to directly plot the number of people against the air quality data to observe the correlation.

The resulting outcome is a hysteresis plot, displaying how the air quality circulates in a given space over the period of 24 hours.

Immediately, two clear trends can be observed – the bottom part represents late night and early morning, and the top part represents the day the afternoon and the first part of the night, letting us double check that it is in fact correct, when compared to the previous plots.

Lastly, the final thing left was to divide this new dataset into two parts (top and bottom). This was done by calculating the average number of people from the previous 90 measurements. This average was then compared to the current number of people. If the average was less or equal than the current measurement, it was classified as 'increasing occupancy'. The reason for including the case of equality (current number of people is equal to the average from the previous measurements) into the *increasing occupancy* category was that the air quality further degrades in such a situation. This also happens if the number of people increases, so the final model will be more accurate. The other measurements, where the average number of people is larger than the current number of people, were classified as *decreasing occupancy*.

The figure below (Figure 26), shows the *increasing occupancy* - category in blue while the *decreasing occupancy* - category is shown red.

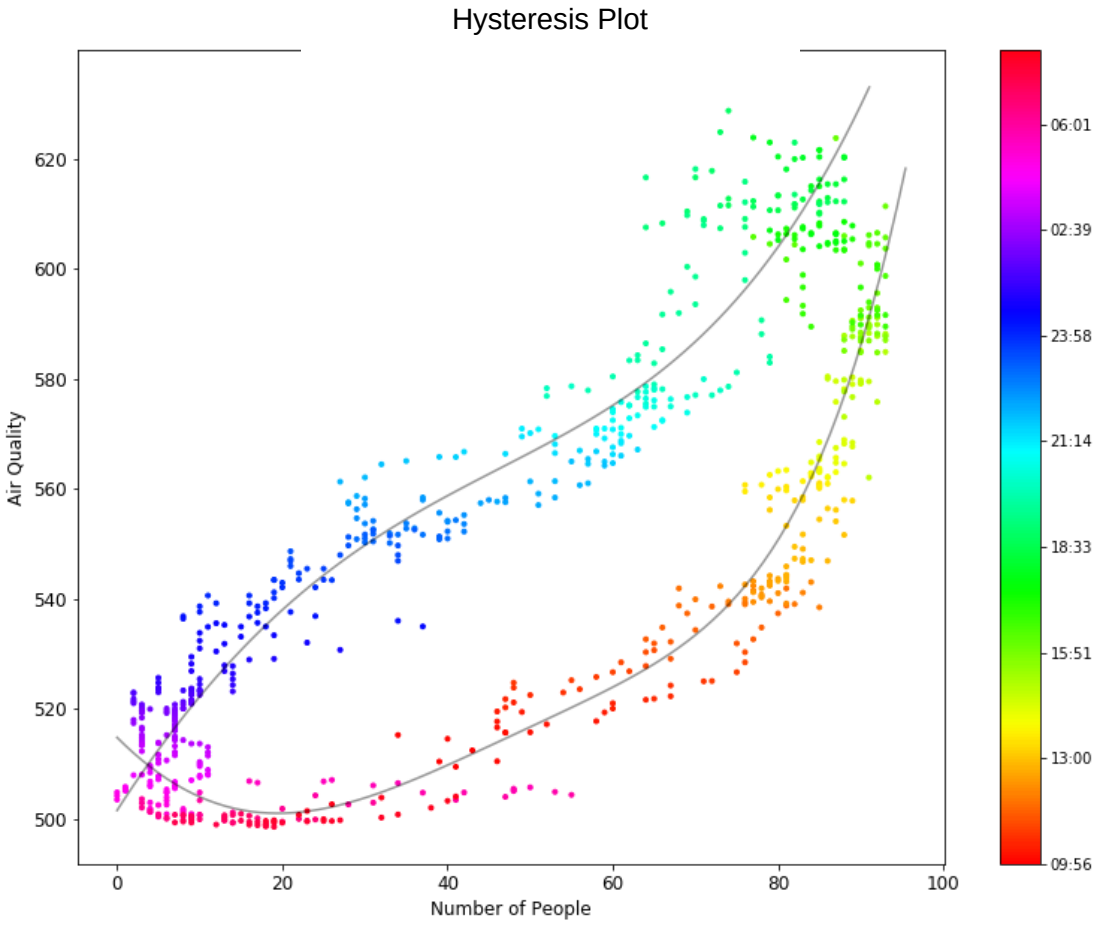

Figure 26 Hysteresis plot with the function approximations overlapped

The same polynomial regression fitting procedure was used as indicated above. Now it was finally possible to predict the quality of air given the number of people in a given space. To select which model we will be fitting, we first have to observe if the number of people is increasing or decreasing. Additionally, if we are monitoring a space with consistent daily patterns, we can also select it based on time.

Due to difficulties with the IBM Watson Studio platform, the generated model ended up being put on NodeRed.

The model was extracted using the following, function:

`print(lin_reg.coef_)`, where *lin_reg* is the employed polynomial regression model

The output is the corresponding coefficients of the polynomial line:

[[517.424 9.12730070e+00 -8.36829986e-01 3.23766962e-02

-5.90161058e-04 5.07454530e-06 -1.65093843e-08]]

The first number corresponds to the intercept and all of the following numbers are coefficients for corresponding polynomial degrees. The same procedure was repeated for both curves. The functions were recreated in the Cloud in the NodeRED system. Example of the first function below:

$$f(x) =$$

517.424+ (9.12730070e+00)***x**+(-8.36829986e-01)***x^2**+(3.23766962e-02)***x^3**+(-5.90161058e-04)***x^4**+(5.07454530e-06)***x^5**+(-1.65093843e-08)***x^6**

Evaluating

Testing methods

- Air quality sensor calibration and drift
- Counting people in the library “manually” (real life) to make sure the scraper system is precise enough
- Testing how the Science library website updates the number of people upon triggering the sensors
- Tested in a student dormitory room with a single door

The system was supposed to be tested on the 4th floor Science Library where the initial data collection for both the number of people vs time and air quality vs time models. Although an application for Risk Assessment was put together and forwarded, the lengthy process of approval made it impossible to achieve this extra step in due time.

Future work for the current system

The project touched upon the analysis for feasibility and also aimed to build a proof of concept. The perspective was in a way more experimental than a fully commercial solution. Because of time limitations, team size and limited budget, only hobby-electronics level parts were used. One future development of the system would be running the analysis again with more accurate sensors.

For the Cloud model the most important improvement would be making it adaptable. Because of an IBM Cloud Infrastructure bug (a report to IBM Support was sent) the team found it impossible to use the model in Watson Studio through NodeRED, instead the static functions. Future development has the choice of either keeping a NodeRED only approach and modifying the model functions using NodeRED templates. This approach is easier to code and adapt but it is impossible to maintain for more complex functions (given by more complex room with multiple entrances and windows). The only way to implement the feature in a very complex setting would be to manage to link the python notebook in Watson Studio with NodeRED to have the full power of the IBM data analytics and the flexibility of the NodeRED IoT data collection and device integration.

Conclusion

After identifying a market niche regarding predictive air quality control, several ways of achieving a system were compared. As a result, the team produced and tested a fully functional proof of concept design. A strong common desire was to test the system in the library to assess how well the system would perform in a more natural environment but for the purpose of the project, testing it in a student dorm room proved sufficient.

The system is very precise and considerably more robust than an ultrasound sensor approach. It is also relatively cheap, very easy to install and maintain and requires virtually no hardware maintenance. The running cost of the system would go down with larger implementations and large-scale production of kits to use in the system.

Bibliography

[1] Indoor enviroments Division, “Indoor air quality,” October 1997. [Online]. Available: https://www.epa.gov/indoor-air-quality-iaq/office-building-occupants-guide-indoor-air-quality.

[2] Airthinkx, “Airthink pricing,” [Online]. Available: https://airthinx.io/shop/.

[3] awair, “Awair 2,” 2019. [Online]. Available: https://getawair.co.uk/.

[4] Dyson, “Dyson Air Purifiers,” [Online]. Available: https://www.dyson.co.uk/air-treatment/purifiers.html?utm_campaign=uk_en__environmental__purifiers__do__dyson__na__na__text__all__range_exact&utm_source=google&utm_medium=paid_search&utm_term=dyson+purifier&utm_content=ds_na&ds_rl=1260330&ds_rl=1254550&ds_.

[5] Aeroqual, [Online]. Available: https://www.aeroqual.com/product/series-900-fixed-gas-monitor.

[6] NodeMCU, “ESP8266 homepage,” [Online]. Available: https://www.nodemcu.com/index_en.html#fr_54745c8bd775ef4b99000011.

[7] P. Kapoor, 2017. [Online]. Available: https://www.greenbiz.com/article/case-office-buildings-windows-open.

Appendix

Ultrasound People Counting Code

```
int trigPin_orange = 11;    // Trigger
int echoPin_yellow = 12;    // Echo
int trigPin_purple = 10;    // Trigger
int echoPin_pink = 9;    // Echo
```

```
int led_onboard = 13;
int threshold_mindist = 80; // minimum distance for sensor to detect object [cm]
long duration_pink, duration_yellow, distance_cm_pink, distance_cm_yellow;

void setup() {
  //Serial Port begin
  Serial.begin (9600);
  //Define inputs and outputs
  pinMode(trigPin_orange, OUTPUT);
  pinMode(echoPin_yellow, INPUT);
  pinMode(trigPin_purple, OUTPUT);
  pinMode(echoPin_pink, INPUT);
  pinMode(led_onboard, OUTPUT);
  digitalWrite(led_onboard, LOW);
}

void loop() {
  // Sensor Orange/Yellow: ////////////////////////////////////////
  // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  digitalWrite(trigPin_orange, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin_orange, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin_orange, LOW);

  // Read the signal from the sensor: a HIGH pulse whose
  // duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
  duration_yellow = pulseIn(echoPin_yellow, HIGH);
  distance_cm_yellow = (duration_yellow / 2) / 29.1;   // Divide by 29.1 or multiply by 0.0343

  // Sensor Purple/Pink: ////////////////////////////////////////
  digitalWrite(trigPin_purple, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin_purple, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin_purple, LOW);

  duration_pink = pulseIn(echoPin_pink, HIGH);
  distance_cm_pink = (duration_pink / 2) / 29.1;   // Divide by 29.1 or multiply by 0.0343


//  Serial.print(distance_cm_yellow);
//  Serial.println("cm (Gate 2)");
//  Serial.print(distance_cm_pink);
//  Serial.println("cm (Gate 1)");
```

```
//  Serial.println("------------------");
//  Serial.println();

  if (distance_cm_yellow < threshold_mindist && distance_cm_yellow > 3)
  {
    Serial.println(duration_pink);
    Serial.print(distance_cm_yellow);
    Serial.println("cm  (Gate 2)");

    double t = millis();
    while(millis() - t < 1000)
    {
      digitalWrite(trigPin_purple, LOW);
      delayMicroseconds(5);
      digitalWrite(trigPin_purple, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin_purple, LOW);

      duration_pink = pulseIn(echoPin_pink, HIGH);
      distance_cm_pink = (duration_pink / 2) / 29.1;   // Divide by 29.1 or multiply by 0.0343

      if(distance_cm_pink < threshold_mindist && distance_cm_pink > 3)
      {
        Serial.println("Gate 2 -> Gate 1");
        Serial.println("----------------");
        delay(1000);
      }
    }
  }
  else if(distance_cm_pink < threshold_mindist && distance_cm_pink > 3)
  {
    Serial.print(distance_cm_pink);
    Serial.println("cm  (Gate 1)");

    double t = millis();
    while(millis() - t < 1000)
    {
      digitalWrite(trigPin_orange, LOW);
      delayMicroseconds(5);
      digitalWrite(trigPin_orange, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin_orange, LOW);

      duration_yellow = pulseIn(echoPin_yellow, HIGH);
      distance_cm_yellow = (duration_yellow / 2) / 29.1;   // Divide by 29.1 or multiply by 0.0343

      if(distance_cm_yellow < threshold_mindist && distance_cm_yellow > 3)
      {
```

```
      Serial.println("Gate 1 -> Gate 2");
      Serial.println("----------------");
      delay(1000);
     }
    }
  }
  else
  {
   digitalWrite(led_onboard, LOW);
  }
}
```

Light Gate ESP8266 Code

```
 /* This example shows how to use continuous mode to take
range measurements with the VL53L0X. It is based on
vl53l0x_ContinuousRanging_Example.c from the VL53L0X API.

The range readings are in units of mm. */
//ADDRESS_DEFAULT 0b0101001 or 41
// address should be 0-127 most likely
//address
#include <Wire.h>
#include <VL53L0X.h>

#define XSHUT_pin2 7


VL53L0X sensor1;
VL53L0X sensor2;

int gate1=0, gate2=0;
int mindist=20, maxdist=2000;
int timeout_human=200; //takes max 200ms to cover both gates once in the beam of one of the
int timeout_pass= 800; // take max 800ms to go from both being covered to actually passing through
unsigned long triggertime=0;
int entering=14;

void setup()
{
pinMode(XSHUT_pin2, OUTPUT); //put this sensor in shutdown


  Serial.begin(9600);
  Serial.print("hello");
//  Wire.setClock(200000);
  Wire.begin();
  //Change address of sensor and power up next one
```

```
  sensor1.setAddress(46);
  pinMode(XSHUT_pin2, INPUT); // change it to alive
  delay(100);
  sensor2.setAddress(47);


  sensor1.init();
  sensor2.init();

  sensor1.setTimeout(500);
  sensor2.setTimeout(500);

  // Start continuous back-to-back mode (take readings as
  // fast as possible).  To use continuous timed mode
  // instead, provide a desired inter-measurement period in
  // ms (e.g. sensor.startContinuous(100)).
  sensor1.startContinuous();
  sensor2.startContinuous();

  Serial.println("Sensor 1 addres (should be 46):");
  Serial.println(sensor1.getAddress());
  Serial.println("Sensor 1 addres (should be 47):");
  Serial.println(sensor2.getAddress());
}

void loop()
{
  //Serial.println(sensor.getAddress());
//  Serial.print("sensor 1:");
//  Serial.println(sensor1.readRangeContinuousMillimeters());
//  if (sensor1.timeoutOccurred())
//  { Serial.print(" TIMEOUT"); }
//    Serial.print("sensor 2:");
//
//  Serial.println(sensor2.readRangeContinuousMillimeters());
//  if (sensor2.timeoutOccurred())
//  { Serial.print(" TIMEOUT"); }
//  Serial.println(".....................");
//  delay(500);

  gate1= sensor1.readRangeContinuousMillimeters();

  gate2= sensor2.readRangeContinuousMillimeters();
  Serial.println(gate1);
   Serial.println(gate2);
   Serial.println(".....................");
//    delay(20);
//    if(gate1<2000 || gate2<2000)
```

```
//    delay(1000);
 //person moves at 1-2 meters per second so distance between gates (10cm) is done in 100ms
 //it will take less than

 entering=14; //make sure it gets changed and we don't have previous value

 if(gate1<2000 || gate2<2000) //triggered
 {
  triggertime=millis();
  while((millis()-triggertime < timeout_human) && entering==14)
  {

    gate1= sensor1.readRangeContinuousMillimeters();

    gate2= sensor2.readRangeContinuousMillimeters();
    Serial.println("triggered");
    Serial.println(gate1);
    Serial.println(gate2);

    if(gate1<2000 && gate2<2000)
    {
     Serial.println("both triggered ");
     Serial.println(gate1);
     Serial.println(gate2);
     while((millis()-triggertime<timeout_pass) && entering==14)
     {

       gate1= sensor1.readRangeContinuousMillimeters();

       gate2= sensor2.readRangeContinuousMillimeters();
       Serial.println("one is released ");
        Serial.println(gate1);
        Serial.println(gate2);
       if(gate1> 8000)
       {
        entering=1;
        Serial.println("entering ");
        Serial.println(gate1);
        Serial.println(gate2);

       } //going from 1 to 2 (left to right)
       else if(gate2> 8000)
       {
        entering=0;
        Serial.println("exiting");
        Serial.println(gate1);
        Serial.println(gate2);
```

```
        }
      }
    }
  }
}
//if one of them gets lower than 2000mm than it is triggered and valid
//or if both - first one to be released is the back one
//move in from left to right or right to left
if(entering==1)
{
Serial.println("=============\\\\" );
Serial.println("=============\//");
Serial.println("          \//" );
Serial.println("         \//" );
delay(500);
}
else if(entering == 0)
{
Serial.println("   \// ");
Serial.println("  \// ");
Serial.println(" \//============= ");
delay(500);
}
```

Air quality collection code

```
//
// The circuit:
// * analog sensors on analog ins 0, 1, and 2
// * SD card attached to SPI bus as follows:
// ** MOSI - pin 11
// ** MISO - pin 12
// ** CLK - pin 13
// ** CS - pin 4 (for MKRZero SD: SDCARD_SS_PIN)

// read air quality every second/ logs the average of 60 seconds(1 entry on the log/everyminute)

#include <SPI.h>
#include <SD.h>

const int chipSelect = 4;
int analogPin=A0;
double sum=0,average=0;
int seconds = 0;
```

```
int minutes = 3;
int hours = 15;

int day = 20;
int month = 02;
int year = 2019;
unsigned long rtc=millis();

void setup()
{
  pinMode(analogPin, INPUT);
 // pinMode(13, OUTPUT);

  // Open serial communications and wait for port to open:
  Serial.begin(9600);
//  while (!Serial) {
//    ; // wait for serial port to connect. Needed for native USB port only
//  }
  Serial.print("Initializing SD card...");

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect))
  {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1);
  }
  Serial.println("card initialized.");
  delay(100);

   File dataFile = SD.open("airqlog.txt", FILE_WRITE);
   String dataString="------------------------------START OF LOG----------------------";

  // if the file is available, write to it:
  if (dataFile)
  {
    Serial.println (dataString);
    dataFile.println(dataString);
    delay(100); //allow to write
    dataFile.close();   ////close the file
  }
  else
  {
    Serial.println("failed to open comms with SD");
  }

}
```

```
void loop()
{
  // make a string for assembling the data to log:
  String dataString = "";
  // read three sensors and append to the string:
  sum=0;


  for (int i = 0; i< 60; i++)  //60 seconds
  {
    int sensor = analogRead(analogPin);
    sum= sum + sensor;
    Serial.print("the particulate value : ");
    //digitalWrite(13,HIGH);
    delay(499); //basically do a reading every 500ms and do an avergage every second.
    Serial.println (sensor);
   // digitalWrite(13,LOW);
    delay(499);// delays are broken apart so that there is continuous power draw and power bank does
not stop
  }

  ////////////////////////////////////////////////////////////////////////////////////////the time part
//  if(millis()-rtc>=1000) //one second has passed
//  {seconds++;
//  rtc=millis();}
  seconds=60; //becuase the loop above takes 60 seconds
  if(seconds==60)
  {
    seconds=0;
    minutes++;
    if(minutes==60)
    {
      minutes=0;
      hours++;
      if(hours==24)
      {
        hours=0;
        day++;
      }
    }
  }
    dataString += String(day);

  dataString +="FEB=> ";

  dataString += String(hours);
  dataString +=":";
```

```
   dataString += String(minutes);
   dataString +=" ";
//    dataString += String(seconds);
//    dataString +=" ";

  average = sum/60;
  dataString += String(average); //add to the string

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  File dataFile = SD.open("airqlog.txt", FILE_WRITE);

  // if the file is available, write to it:
  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();   ////close the file
    Serial.println(dataString); //also have it on the serial
    // print to the serial port too:
    //Serial.println(sum);
  }
  // if the file isn't open, pop up an error:
  else {
    Serial.println("error opening airqlog.txt");
  }
}
```