 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 1 of 37 Date: 28 October 2022
--	--	---

Final Subsystem Design **UAVPayload^{TAQ}** **Air Quality Sensor**

Revision Record

Prepared by	A.G. _____ Alex Gray, Air Quality Subsystem Lead	Date 28/10/2022
Checked by	A.S. _____ Alex Switala, Target Acquisition and Image Processing Co-Lead	Date 28/10/2022
Approved by	M.B. _____ Marissa Bowen, Project Manager and Enclosure Design Lead	Date 28/10/2022
Authorised for use by	_____ Dr. Felipe Gonzalez, Project Coordinator	Date 28/10/2022

Queensland University of Technology
 Gardens Point Campus
 Brisbane, Australia, 4001.

This document is Copyright 2022 by the QUT. The content of this document, except that information, which is in the public domain, is the proprietary property of the QUT and shall not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT



Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Initial Issue	28/10/2022	A.G

Table of Contents

Paragraph	Page No.
1 Introduction	6
1.1 Scope	6
1.2 Background	6
2 Reference Documents.....	7
2.1 QUT Avionics Documents	7
2.2 Non-QUT Documents	8
3 Subsystem Introduction	9
3.1 Subsystem Requirements	9
4 Subsystem Architecture.....	11
4.1 Interfaces	12
5 Design.....	14
5.1 Hardware Specifications.....	15
5.1.1 Raspberry Pi 3B+	15
5.1.2 Pimoroni Enviro+ Sensor	16
5.1.3 Physical connections	17
5.2 Software.....	18
5.2.1 Software Flow Diagram	18
5.2.2 Implemented Code	19
6 Conclusion.....	26
7 Appendix:	28
7.1 Appendix A Enviro+ connections:	28
7.2 Appendix B Enviro+ pin usage:	29
7.3 Appendix C Software and libraries used:	30
7.4 Appendix D MiCS-6814 (gas sensor) datasheet:	31
7.5 Appendix E Enviro+ code implementation:	32



List of Figures


Figure	Page No.
Figure 1: System Architecture with Air Quality system highlighted	11
Figure 2: Air Quality Interface Diagram	12
Figure 3: Full Mechanical Diagram of a Raspberry Pi 3B+ see Doc-1	15
Figure 4: Pimoroni Enviro+ Sensor.....	16
Figure 5: I2C Message Layout (Doc-3)	17
Figure 6: Software Flow Diagram.....	18
Figure 7: Calibration Code	20
Figure 8: Temperature Code.....	21
Figure 9: Pressure Code	22
Figure 10: Humidity Code.....	23
Figure 11: Light and Proximity Code.....	23
Figure 12: Gas Code.....	24
Figure 13: LCD Code	25

List of Tables

Table 1: Subsystem requirements.....	9
Table 2: Air Quality Subsystem Interfaces	13
Table 3: Key Dimensions of Enviro + Sensor.....	16
Table 4: Requirements Met by Preliminary Design.....	26

Definitions

ED	Enclosure Design
ST	Sampling Tube
AQS	Air Quality Sensor
GPIO	General Purpose Input Output
TAIP	Target Acquisition and Image Processing
WVI	Web Visualisation and Interfaces

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 6 of 37 Date: 28 October 2022
--	---	---

1 Introduction

The **final** design document has been created to help in the understanding of the subsystem to both the client and the other subsystem leads. It will be used to establish the initial design plan and software flow as well as an overview of the system and how it is integrated with the rest of the project. The final design document will also outline all requirements that relate to the subsystem created from the HLO's seen in document RD/1. The full requirements list can be seen in RD/2.

1.1 Scope

The purpose of this document is to outline the **final** design decisions of this subsystem and to confirm the requirements that relate to the AQS. This will be achieved though showing and describing where the subsystem sits in the system architecture and what software connections will be used to send data through to the other subsystems.

1.2 Background

The Queensland University of Technology's Airborne System Lab (ASL) has commissioned the group UAVPAYG19 to design and develop a payload capable in detecting specific objects, recording air quality data to be displayed on a web interface and to pierce a ground sample. This payload is to be attached to a S500 UAV which completed an automated flight path. The payload is mounted on the bottom of the UAV using a proved bracket. This payload must contain all components to complete its required tasks. These components are:

- Raspberry Pi 3b+
- Raspberry Pi Camera
- Pimoroni enviro sensor
- DF15RSMG 360 Degree Motor

The project is required to identify three targets, a valve (In open or closed position), a fire extinguisher and an ArUCO marker. The Pimoroni sensor is to be used to record air temperature, pressure humidity, light and potentially hazardous gas level data. This data along with a live feed of the Raspberry Pi Camera is to be visualized on a Web Interface. Lastly a soil sample must be obtained using a sampling mechanism.

2 Reference Documents

2.1 QUT Avionics Documents

RD/1	UAV - Customer Needs	Advanced Sensor Payload for UAV Target Detection and Air Quality Monitoring in GPS Denied Environments
RD/2	UAV - System Requirements	UAVPayloadTAQ System Requirements
RD/3	UAVPAYG19-PM-PMP-02	PMP
RD/1	UAVPAYG19-ICD-01	ICD
RD/6	UAVPAYG19-PD-ED-02	ED Preliminary Design
RD/7	UAVPAYG19-PD-AQS-02	AQS Preliminary Design
RD/8	UAVPAYG19-PD-TAIP-02	TAIP Preliminary Design
RD/9	UAVPAYG19-PD-ST-02	ST Preliminary Design
RD/10	UAVPAYG19-PD-WVI-02	WVI Preliminary Design
RD/21	UAVPAYG-19-AQS-TR-01	Air Quality Sensor Test
RD/25	UAVPAYG-19-AQS-TAIP-WVI-TR-01	Air Quality Sensor, Target Acquisition and Image Processing, Web Visualization and Interface Integration Report
RD/26	UAVPAYG-19-ED-AQS-TAIP-ST-TR-01	Enclosure, Air Quality Sensor, Target Acquisition and Image Processing, Sampling Tube Test Report
RD/27	UAVPAYG-19-ED-AQS-TAIP-WVI-ST-TR-01	Enclosure, Air Quality Sensor, Target Acquisition and Image Processing, Web Visualization and Interfaces, Sampling Tube Integration Report
RD/28	UAVPAYG-19-TR-AT-01	Acceptance Test Report
RD/29	UAVPAYG-19-VV-01	Verification and Validation Report
RD/32	UAVPAYG19-TAIP-FD-01	Target and image final report

2.2 Non-QUT Documents

Document number	Title	IEEE Reference
Doc-1	Raspberry Pi 3 Model B+	R. Pi, "Raspberry Pi 3 Model B+," Raspberry Pi, 25 August 2022. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/ . [Accessed 25 August 2022].
Doc-2	BASICS OF THE SPI COMMUNICATION PROTOCOL	S. Campbell, "BASICS OF THE SPI COMMUNICATION PROTOCOL," Circuit Basics, 13 February 2016. [Online]. Available: https://www.circuitbasics.com/basics-of-the-spi-communication-protocol . [Accessed 25 August 2022].
Doc-3	BASICS OF THE I2C COMMUNICATION PROTOCOL	S. Campbell, "BASICS OF THE I2C COMMUNICATION PROTOCOL," Circuit Basics, 14 February 2016. [Online]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/ . [Accessed 25 August 2022].
Doc-4	Calibration / correction of sensor readings	roscoe81, "Calibration / correction of sensor readings," github, 23 May 2020. [Online]. Available: https://github.com/pimoroni/enviroplus-python/issues/67 . [Accessed 12 September 2022].
Doc-5	Enviro+ readings unreliable	Roscoe27, "Enviro+ readings unreliable," Pimoroni, 3 February 2022. [Online]. Available: https://forums.pimoroni.com/t/enviro-readings-unreliable/12754/58 . [Accessed 12 September 2022].
Doc-6	Enviro+ / Ohms to PPM	Roscoe27, "Enviro+ / Ohms to PPM," Pimoroni, 8 March 2020. [Online]. Available: https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/6 . [Accessed 12 September 2022].
Doc-7	BME280 Datasheet	Bosch, "BME280 Humidity, Pressure & Temperature Sensor," 2022. [Online]. Available: https://static6.arrow.com/aropdfconversion/fab9aec0acec451f1e965292bf2ed54d3e7ce427/bst-bme280-ds002.pdf . [Accessed 12 September 2022].
Doc-8	roscoe81/enviro-monitor	Roscoe81, "roscoe81 / enviro-monitor," GitHub, 9 January 2022. [Online]. Available: https://github.com/roscoe81/enviro-monitor . [Accessed 5 September 2022].

3 Subsystem Introduction


The AQS subsystem is responsible for recording and sending air quality measurements to the web interface for the user to view. As per the requirements listed in Table 1 below the system must be able to take air quality measurements to determine the measurements of hazardous gasses. These requirements have been provided by the client and is imperative to the project and client that they are met before the project is considered a success. A full outline of the requirements for all subsystems along with a verification table can be seen in QUT document RD/2.

3.1 Subsystem Requirements

Table 1: Subsystem requirements

Requirement	Description	Verification
REQ-M-01	The UAVPayloadTAQ shall remain under the maximum weight of 320 g and comply with an IP41 rating. The air quality sensors must be exposed to the environment to allow for accurate reading.	Demonstration/ Measured
REQ-M-02	The UAVPayloadTAQ must measure hazardous gases, (these include Carbone monoxide, Nitrogen dioxide, Ethanol, Hydrogen, Ammonia, methane, and Iso-butane) humidity, pressure, temperature and light via on-board sensors.	Demonstration
REQ-M-03	The UAVPayloadTAQ shall communicate with a ground station computer to transmit video, target detection and air quality data.	Demonstration
REQ-M-05	The Web Interface is required to display real time air sampling data that is recorded directly from the UAVPayloadTAQ and updated dynamically throughout the duration of the flight.	Demonstrated
REQ-M-10	Preliminary designs shall be completed by week 7	Submitted Documentation
REQ-M-11	The payload should display its IP address via the integrated Enviro sensor LCD screen	Demonstration
REQ-M-12	The LCD screen should display live feed of target detection as well as temperature readings from the Pi and the Enviro sensor board.	Demonstration
REQ-M-13	The LCD screen shall be placed on the side of the payload in order for the user to easily see its operation during flight.	Demonstrated
REQ-M-14	Developed solution shall conform to the systems engineering approach.	Submitted documentation
REQ-M-15	The system shall have logged functioning operation for a minimal period of 10 minutes prior to acceptance test	Demonstration
REQ-M-19	Live data from the UAV must be made available through the web server within 10 seconds of capture.	Demonstration

To achieve all of the requirements listed in Table 1, an environment sensor has been provided to the group. This sensor (Pimoroni Enviro+) will collect the required data sending it to the web interface for the user to view. In addition to this, the temperature reading will be displayed in a rotation with a live stream of the imagery system and the raspberry pi's IP. This will all be displayed using the built in LCD screen. The demonstration flight will not commence until 10 minutes has passed to allow enough time for the sensors to be calibrated to the surroundings ([**REQ-M-15**]).

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 10 of 37 Date: 28 October 2022
--	--	--

The Enviro+ sensor board has been provided for this project and has all sensors required integrated into one board as well as a colour LCD screen. The advantages of all sensors integrated in this manner is a simplified integration with other subsystems and a decreased complexity of wiring. As this is a hat for a raspberry pi it occupies the GPIO pins without the need for additional wiring and the need for mounting options for multiple sensors.

The Enviro+ will be controlled through the raspberry pi using a variety of communication methods. These methods are I2C and SPI using 16 of the raspberry pi's GPIO pins. In addition to providing a communication method to collect the data, the raspberry pi also provides power and power pass through for other devices that may need it (power pass through will not be used for this project). As all pins of the raspberry pi will be covered by the hat the Enviro+ provides a single GPIO pin output that will be used by the sampling tube subsystem.

4 Subsystem Architecture

The entirety of the System Architecture can be seen in Figure 1 below with the components and connections that are relevant to the AQS highlighted in a green box. From this it can be seen that power is provided by the raspberry pi into the sensor. From here data is collected and sent back to the raspberry pi for processing and data storage in the database. At this stage the data is sent to the webserver for the WVI subsystem to handle and display. In addition to the data being sent to the WVI the temperature data will be sent to the LCD screen for users to view and will be in a rotation with the IP and live image processing.

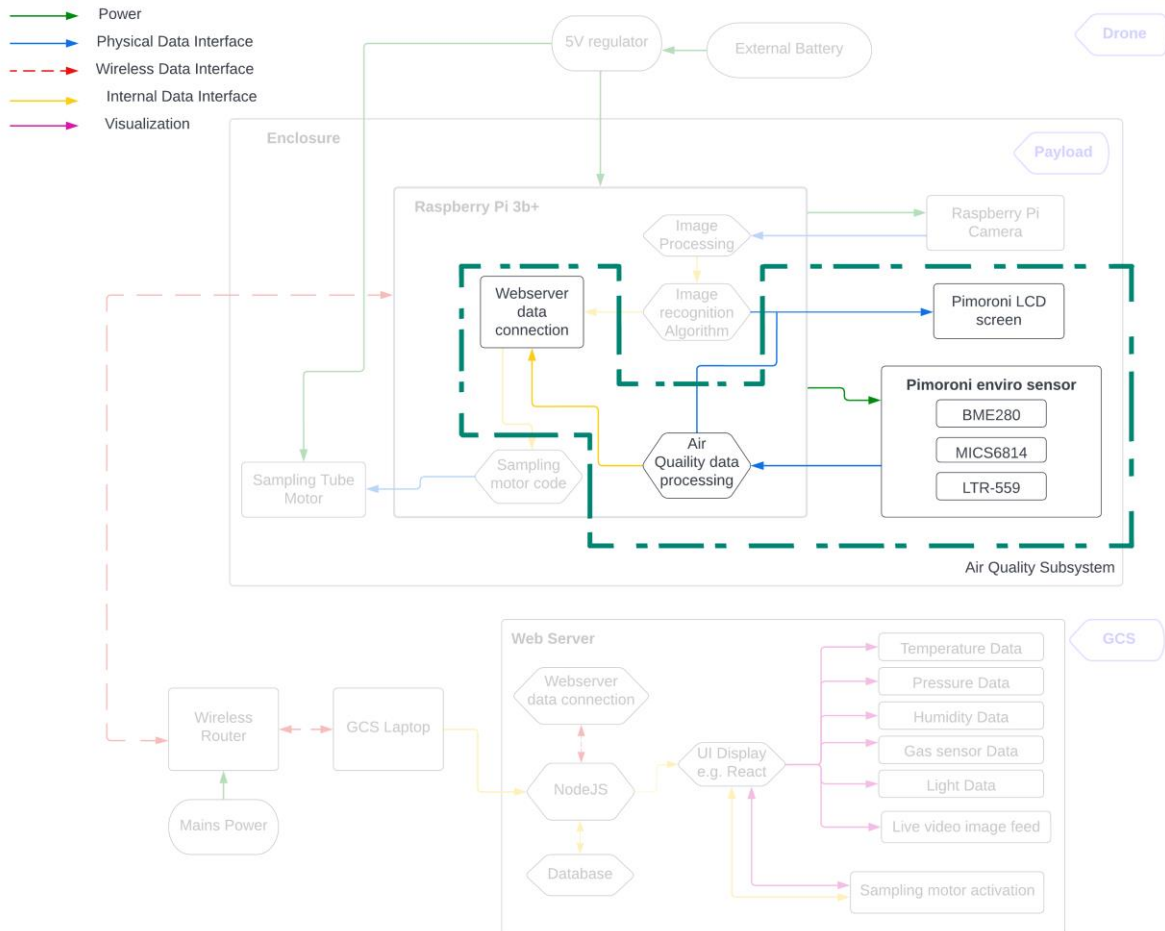


Figure 1: System Architecture with Air Quality system highlighted

4.1 Interfaces

For all requirements to be met for this project all subsystems must be interfaced together. The AQS has interfaces with imagery, the raspberry pi, the WVI and the ED. An overview of the connections and the type can be seen in Figure 2:

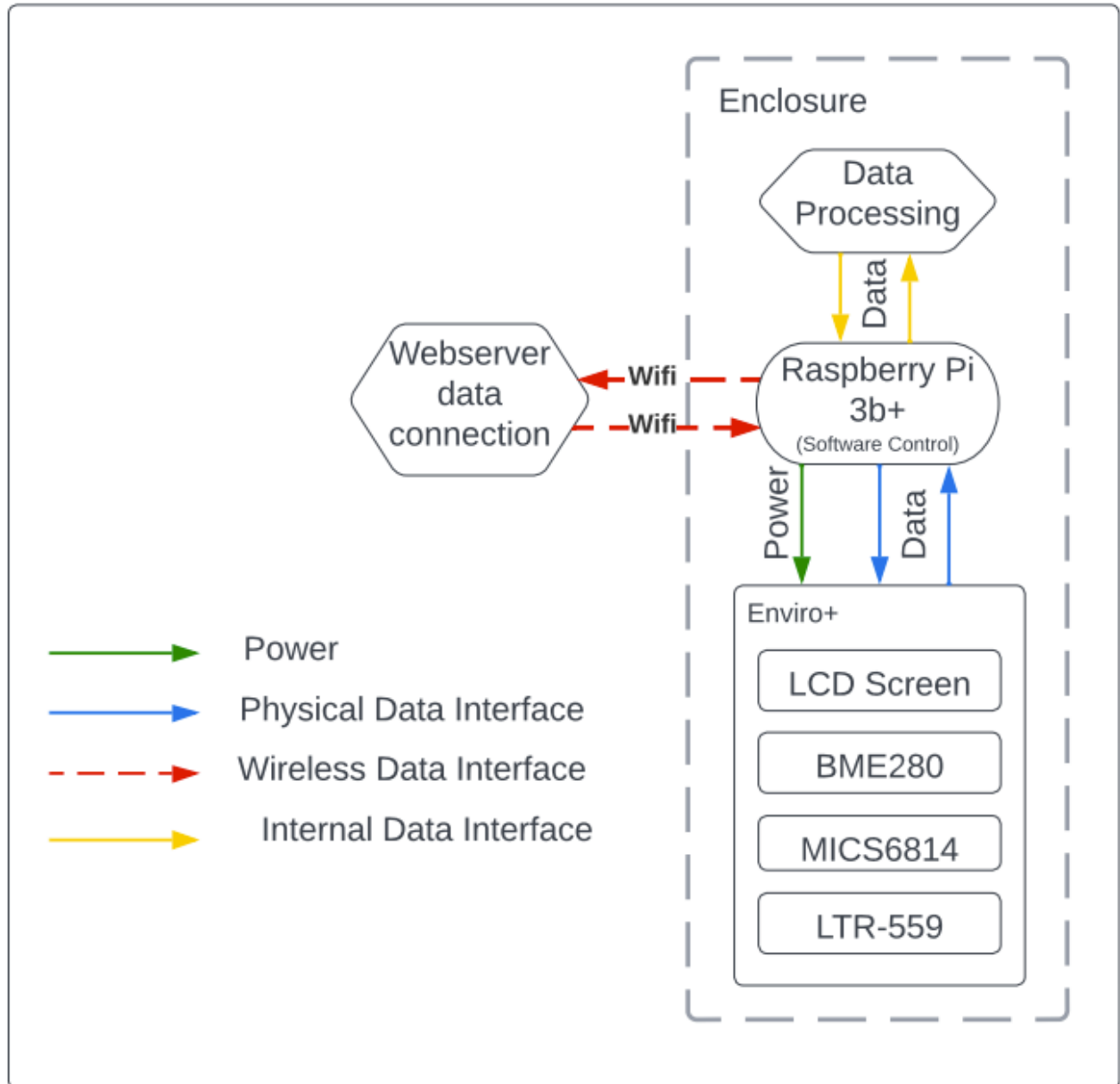


Figure 2: Air Quality Interface Diagram



 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 13 of 37 Date: 28 October 2022
--	--	--

Table 2 below provides a description of each interface in more detail then provided in figure 2 describing how each system is connected.

Table 2: Air Quality Subsystem Interfaces

Interface	Component/System	Interface Description
I1	Enclosure	The Enviro+ sensor will be connected to the side of the enclosure through 4 mounting screws. In addition to these holes there is a cut-out in the enclosure behind the sensor to allow for cables to be passed out providing data and power.
I2	Header extension	To allow for the sensor to sit 90 degrees to the raspberry pi a header extender will be used. This will be plugged into the raspberry pi on one end and ensuring that the pins are still in the same orientation (no cross over) the extender will be plugged into the enviro+ sensor.
I3	Servo connection	The servo will be plugged into the enviro+ sensor using the provided GPIO4 and GND pass through pins. This will allow for control of the servo motor
I4	Software connection	The raspberry pi will communicate with the sensor by instructing it to gather data as required by [REQ-M-02] . In addition to this, data will be passed from the WVI subsystem to the sensor's onboard LCD screen. This along with the temperature reading and IP of the raspberry pi will make up requirements [REQ-M-11 and REQ-M-12] .
I5	Internal Software	Data processing for the Air Quality samples will be handled on the onboard raspberry pi before being sent to either the LCD (temperature) or sent to the web API (all AQS data) where it will be displayed as part of the WVI.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 14 of 37 Date: 28 October 2022
--	---	--

5 Design

The system that **was implemented** was designed to meet the requirements described in Table 1. Using both hardware and software connections, the Enviro+ sensor was used to detect the presence of hazardous gasses in the air and report them with basic meteorological data to the user on the ground. This was achieved using the sensors that are integrated into the enviro+ sensor. These sensors include a temperature, pressure, and humidity sensor, a light and gas sensor and an analogue gas sensor. The temperature data, in addition to being displayed on the web interfaces, was also displayed on the LCD screen.

The data once collected **was** processed onboard the raspberry pi. This is required for a few reasons. **The first reason** is to ensure that the data is in the correct format. This is especially important for the analogue gas sensor as it returns data as a resistance value and not in the correct units. The datasheet for this sensor provides the curves for converting the data to the correct value. Using the datasheet **and online resources (Doc-6)**, a function **was implemented** that can convert the data into the correct format before being sent to the database. Once all data has been appropriately processed it **was** sent to the webserver as a JSON object. From the webserver the WVI **takes** the data to be stored in a database and displayed on the ground station. The **temperature function was also** accessed by the program managing the LCD and what is currently being displayed. This is due to the LCD needing to display one of three messages, these being: the IP address of the raspberry pi, the current feed from the TAIP system or the temperature.

In order to swap between the different displays, the light/touch sensor **was** used. By covering the sensor and decreasing its value to zero it is possible for this sensor to act as a button as well as reading the light levels of the environment. Using this button, a counter **was** incremented such that if the value of the counter is equal to a particular number, it will correlate to what the LCD should be displaying. **This is done by dividing it by three and keeping the remainder (modulus).**

5.1 Hardware Specifications

This section describes what hardware is used by this subsystem how these devices are connected to each other as well as a description of the components and the communication protocols being used.

5.1.1 Raspberry Pi 3B+

The raspberry pi 3B+ is a microcomputer that is used in many different applications. This is due to its relatively small size compared to its processing power. This device has the following specifications (a full list of the Specs can be found in Doc-1 [as well as an image with wiring connections which can be seen in Appendix A](#)):

- 64-bit processor
- 1GB of LPDDR2 RAM
- 2.4/5GHz wireless LAN
- Bluetooth 4.2
- Gigabit Ethernet
- 40-pin GPIO header
- 5V/2.5A power input
- Camera Support

This device is optimal for this project due to its low power requirements, decreasing the load on the UAV's power supply. In addition to these advantages the raspberry pi is capable of running multiple programs simultaneously thus removing the need to multi thread the programs in software and instead leaving the OS to handle this. The full mechanical diagram of the raspberry pi can be seen in Figure 3 below:

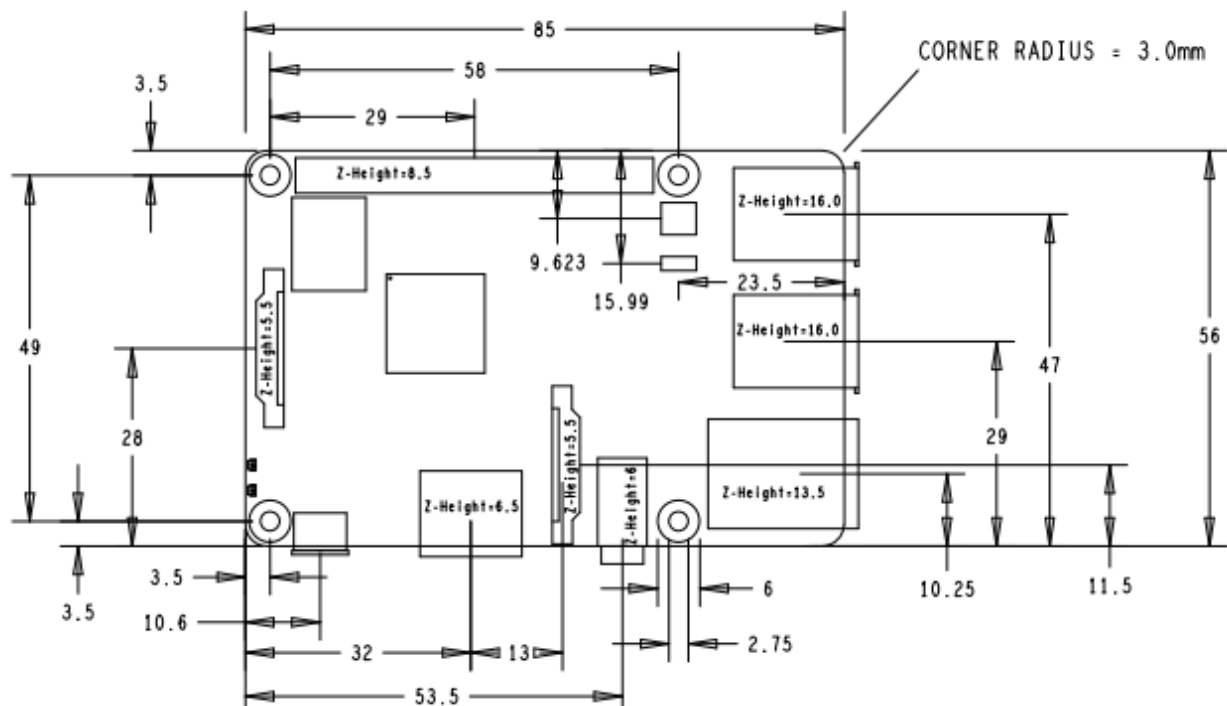


Figure 3: Full Mechanical Diagram of a Raspberry Pi 3B+ see Doc-1

5.1.2 Pimoroni Enviro+ Sensor

The Pimoroni Enviro+ Sensor is an environmental sensor capable of detecting hazardous gasses along with light, pressure, temperature, and humidity. It is designed as a hat for the raspberry pi and makes use of three different sensors to achieve these measurements these are listed below:

- BME280: Temperature, pressure, humidity Sensor
- LTR-559: Light and proximity Sensor
- MICS6814: Analog Gas sensor
- Connector for particulate matter sensor

In addition to these sensors, the Enviro+ features an LCD screen (160x80) capable of displaying graphs of the data collected or displaying other images at the user's discretion. This board makes use of 16 of the raspberry pi's header pins and has passthrough support for a few of them. The dimensions of the enviro+ can be seen below:



Length	65mm
Width	30mm
Height	8.5mm
Bolt size	M2.5

Table 3: Key Dimensions of Enviro + Sensor

Figure 4: Pimoroni Enviro+ Sensor

The board communicated with the raspberry pi on two different methods depending on which sensor is being used. These methods are detailed below:

5.1.2.1 Method 1 SPI

This communication protocol is used by the BME280 sensor as well as the I2C protocol. This protocol functions as a master-slave relationship with the master (microcontroller) providing commands to the slave device (typically the sensor). This communication protocol has the advantage over I2C in that the data can be sent between the master and slave continuously without needing to separate it into packets with separate start stop conditions (Doc-2). This communication method makes use of four wires on both devices. These are the Master output Slave input (MOSI), Slave Output Master Input (SOMI), Clock (SCLK) and the Slave Select Chip Select (SS/CS). Each of these lines allow for the operation of the protocol with the output input (IO) lines being used to send data, the clock is used to synchronise the IO of the data lines to ensure the data is timed correctly and finally the SS/CS line which is used to specify which master/slave the data is being sent to allowing for multiple slaves to be controlled from one master (Doc-2).

The primary advantage of this method over others is the ability to control multiple devices from the one master and being able to continuously send and receive data. These advantages have a disadvantage as well in that four wires are required for each connection decreasing the available space on the PCB. This could cause issues in design (for this project this is not an issue as the PCB has already been developed).

5.1.2.2 Method 2 I2C

The I2C communication protocol is used by all devices on the Enviro+ in some regard. This communication protocol similar to SPI is designed in a master-slave configuration with the capability to send/receive data to/from multiple slaves and masters. However, there are differences between these protocols. The largest of which is the method where data is sent. SPI is able to send data between the microprocessor and slave continuously whereas I2C is only able to send data in packets (Doc-3). A typical layout for the I2C protocol can be seen in Figure 5:

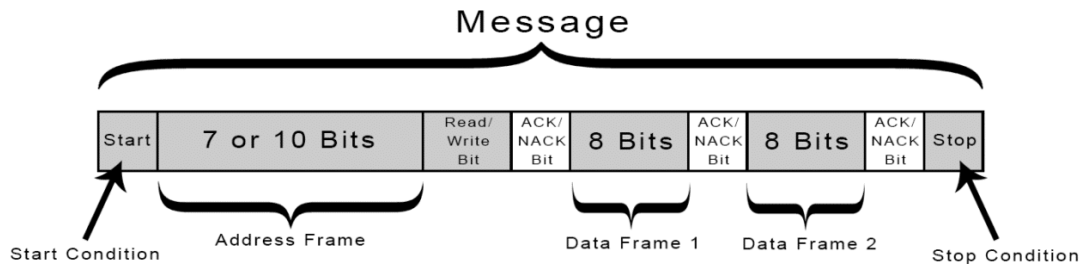


Figure 5: I2C Message Layout (Doc-3)

As can be seen the message or packet is lay out between a start and stop condition. The message starts with the address frame which is used by the master to identify which slave it is communicating with. This is followed by a read/write bit to specify if the master is sending data or requesting data. Before the data is sent the master or slave returns an acknowledgement (ACK) bit stating that they are ready for the data to be sent. If ACK is return the data is sent in 8-bit data frames separated by ACK requests before the stop condition is sent. Whilst this method is slower than the SPI protocol it has the advantage of ensuring that all data that is sent is received by the correct device which leads to a decrease in lost data (Doc-3). The other advantage that I2C has over SPI is a decrease in the required wires with only two needed. These wires are the Serial Data line (SDA) and the Serial Clock line (SCL). The SDA is used to communicate between master and slaves with the SCL being used to ensure the system remains synchronous (Doc-3).

5.1.3 Physical connections

In order for data to be passed between the raspberry pi and the enviro+ sensor a physical connection must be made as well. This is achieved through the use of 16 of the 40 header pins on the raspberry pi the exact pins used, and the purpose can be seen in Appendix B. These header pins allow for the communication protocols described in sections 5.1.2.1 and 5.1.2.2 of this report to function. In this case the raspberry pi is acting as the master with the enviro+ listening and responding to commands as the slave. It should be noted that the enviro+ connection takes up all 40 pins on the raspberry pi even through only 16 are used. This is to provide an added stability to the design but has the drawback of stopping the user from easy access to additional pins. To assist **this issue**, the enviro+ board also acts as a passthrough for a selection of pins. For this project only one of these pins **was** used being GPIO 4. This pin is used by the sampling tube subsystem to control the servo.

To meet **[REQ-M-01]** a header extension cable has been provided. **However, due to the space limitations of the enclosure this cable cannot be used. Instead, a shorter cable was used to better fit the Enviro+ in the enclosure.** In the final design this is used to mount the enviro+ sensor at a 90-degree angle to the raspberry pi. This cable acts as a passthrough of all the raspberry pi header pins and does not have any additional functionality. When connecting between the enviro+ and raspberry pi, **it was important** to ensure that the sensor will still be in the correct orientation as to not short out the pins of either device. An image showing this connection can be seen in appendix A. **Note that in this appendix the Enviro+ is flipped compared to the correct orientation. So, in this configuration the linking cable would be twisted.**

5.2 Software

This section describes the software process that will be implemented to collect and display the data. Due to the popularity of the sensor there are many different implementations of the sensor in current projects. In addition to this there are many forum pages which have been used when debugging code and inaccuracies in data recorded by the Enviro+ sensor cluster. The tests completed to confirm the functionality of this subsystem can be seen in the following test and integration test reports: RD/21, RD/25, RD/26, RD/27 and RD/28. Verification of these tests can be seen in document RD/29.

5.2.1 Software Flow Diagram

All programs for this subsystem **were** written in the python programming language. This has been selected due to its simplicity and the wide support available if there are any issues in implementing these sections of code. Multi-threading of programs **was** handled by the raspberry pi OS and code that is integrated with other subsystems were discussed and developed with those subsystem leads. Figure 6 describes the basic flow of the software indicating where the data is sent once collected and how it is used.

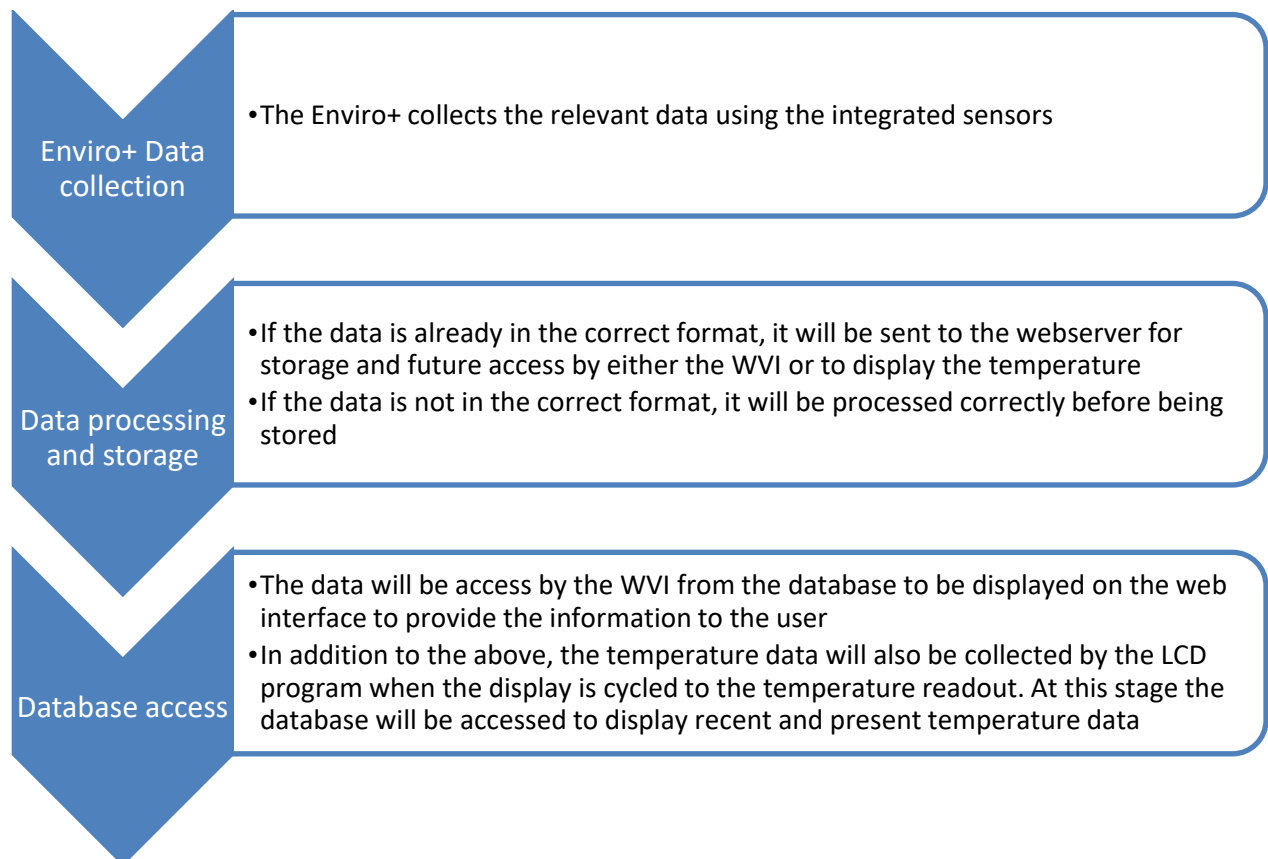



Figure 6: Software Flow Diagram

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 19 of 37 Date: 28 October 2022
--	---	--

5.2.2 Implemented Code

The sections below (from 5.2.2.1 to 5.2.2.7) describe the software that was implemented to collect the data from the sensor on the Enviro+. As well as the processes used to increase the data accuracy. Note that some sensors and code have been combined. During the development of the code, all functions were first written as a stand-alone script which tested the functionality of the subsystem. This code was later refactored to operate as a class allowing it to be accessed during the integration stage of the project simplifying this process. The full class implemented can be found in appendix E. It should be noted that many of the equations used to correct erroneous data values collected by the Enviro+ were sourced from forum pages and answers provided by user Roscoe27. The location the functions used were sourced from, is noted in the code (as will be seen in the code segments). When the primary GitHub for this user and his project was investigated (see Doc-8) and the licence for this code was read it states (the full licence can be seen at the top of the implemented code in appendix E):

“Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software”

This project has implemented only minor equations from this code in an attempt to improve the accuracy and quality of the data. As discussed in the sections below these sections had to be modified to better suit the environment of this project and to provide accurate data.

5.2.2.1 Sensor Calibration

The first stage of the AQS software is to provide baseline measurements to calibrate the sensors. These measurements are as follows: Expected room temperature (if a thermometer is handy this can be used to get an accurate value), current altitude (in meters) and whether the gas sensor requires calibration. The Calibration function can be seen in Figure 7:



```
55 # this function is run once at the start of
56 # the program to find the appropriate factor for the room
57 def calibration_setup(self):
58     global factor
59     global alt
60     global NO2_cal
61     global CO_cal
62     global NH3_cal
63     alt = 0
64     factor = 1
65     adj_temp = 0
66     cpu_temps = []
67
68     cpu_temp = get_cpu_temperature()
69     # Smooth out with some averaging to decrease jitter
70     cpu_temps = cpu_temps[1:] + [cpu_temp]
71     avg_cpu_temp = sum(cpu_temps) / float(len(cpu_temps))
72
73     calibration_temp = input("Enter the current room temperature in degrees celsius: ")
74     # https://en-us.topographic-map.com/maps/vj4v/Ferny-Grove/
75     alt = input("Enter the current altitude (above sea level) of your position (in meters): ")
76     gas_cal = input("Does the gas sensor require calibration? Type 'Yes' or 'No': ")
77     start_time = time.time()
78     finish_time = 0
79
80     if str(gas_cal) == "No":
81         try:
82             with open("Gas_calibration.txt") as params:
83                 content = params.read()
84                 print(content)
85                 content = content.split(" ")
86                 index = 1
87                 for x in content:
88                     if x == "no2_cal":
89                         NO2_cal = str(content[index])
90                     elif x == "co_cal":
91                         CO_cal = str(content[index])
92                     elif x == "nh3_cal":
93                         NH3_cal = str(content[index])
94                     index += 1
95         except:
96             print("Calibration file does not exist enter values manually if known:")
97             print()
98             NO2_cal = input("input the NO2 calibration value (in ohms): ")
99             CO_cal = input("input the CO calibration value (in ohms): ")
100             NH3_cal = input("input the NH3 calibration value (in ohms): ")
101     elif str(gas_cal) == "Yes":
102         # print("wah")
103         min_count = 0
104         timer = 0
105         while(timer < 300 ): #5 minutes
106             timer = finish_time - start_time
107             #print(timer)
108             sensor = gas.read_all()
109             NO2_cal = sensor.oxidising # find the NO2 in ohms
110             CO_cal = sensor.reducing # find the CO value in ohms
111             NH3_cal = sensor.nh3 # find the ethanol in ohms
112             finish_time = time.time()
113             with open("Gas_calibration.txt", 'w') as calibration:
114                 calibration.write("no2_cal " + str(NO2_cal) + " co_cal " + str(CO_cal) + " nh3_cal " + str(NH3_cal))
115
116             print("Calibrating the parameters: ")
117             start_time = time.time()
118             finish_time = 0
119             timer = 0
120             while(timer < 10 ):
121                 timer = finish_time - start_time
122                 cpu_temp = get_cpu_temperature()
123                 # Smooth out with some averaging to decrease jitter
124                 cpu_temps = cpu_temps[1:] + [cpu_temp]
125                 avg_cpu_temp = sum(cpu_temps) / float(len(cpu_temps))
126                 # print("this is the avg CPU temp " + str(avg_cpu_temp))
127                 finish_time = time.time()
128
129             while(adj_temp < float(calibration_temp)):
130                 raw_temp = bme280.get_temperature()
131                 adj_temp = raw_temp - ((avg_cpu_temp - raw_temp) / factor)
132                 factor += 0.01
```

Figure 7: Calibration Code

This function is called once during the initialisation of the Air Quality subsystem. Due to the inaccuracies associated with any sensor integrated in the BME280 chip baseline values were used to generate accurate data. This included an assumed room temperature for the temperature sensor and an altitude for the pressure sensor. Using the temperature provided and taking a ten second average of the raspberry pi's CPU temperature it was possible to find a scaling factor. This can be seen in lines 129 – 132 where while the adjusted temperature was less then the assumed temperature the factor would increase by 0.01. This provided a fast and robust way of setting the factor for temperature control. However, it is not expected that this factor will remain reliable over a long mission and should be reset at the start of each mission. The other value that was requested by the

program is the systems estimated altitude (in meters) this value will be used later to correct the pressure readings.

The final input requested by the user is an input for whether the gas sensors need to be calibrated. This process can be seen from line 76 to line 114. In this process the user is asked to input “yes” if the sensor requires re-calibration or “no” if it has been recently calibrated. In the event that the sensor has been recently calibrated a calibration file will be read to find the average resistor values for each gas group from the last calibration. If this file does not exist, the user can enter the resistor values manually if known. In the case where the gas sensors have not been calibrated the user can enter “no”. in this situation the system will run the gas sensor for five minutes to get a baseline reading of the gas readings in the room. It was suggested in Doc-6 that this process is run for 100 minutes to ensure the sensor is properly warmed up. However, as this application is a demonstration and there is no way to accurately calibrate the gas sensor, five minutes was deemed acceptable.

5.2.2.2 Temperature

The first of the reading collected by the BME280 chip is the chip temperature. Whilst the chip is advertised to be able to give an accurate temperature reading without any adjustment, the initial tests have shown that this was the case (see RD/21). After examining the example code provided by Pimoroni on GitHub the reason soon became clear. As this sensor is reading the temperature at the chip it is heavily affected by the temperature of surrounding components. This includes the Raspberry Pi’s CPU temperature. To attempt to alleviate this (as suggested by online users) the Enviro+ was separated from the Raspberry Pi and was mounted away from the CPU in the enclosure (see RD/6 for more information). This works for a short period, however, the temperature inside the enclosure still has an affect on the reading due to the small space, hence the scaling factor found in calibration. This heat effect on the BME280 effects all the readings of this chip as the temperature is used in compensation for both pressure and humidity (Doc-7). The code implemented to find the adjusted room temperature can be seen in Figure 8:

```

135     # gets the temperature in degrees celsius
136     def temp(self):
137         cpu_temps = []
138         unit = "C"
139         cpu_temp = get_cpu_temperature()
140         # Smooth out with some averaging to decrease jitter
141         cpu_temps = cpu_temps[1:] + [cpu_temp]
142         avg_cpu_temp = sum(cpu_temps) / float(len(cpu_temps))
143         # print("this is the avg CPU temp " + str(avg_cpu_temp))
144         raw_temp = bme280.get_temperature()
145         corrected_temp = raw_temp - ((avg_cpu_temp - raw_temp) / factor)
146         # print("this is the adjusted temperature " + str(corrected_temp))
147         return corrected_temp

```

Figure 8: Temperature Code

5.2.2.3 Pressure

The second sensor from the BME280 chip implemented was the pressure sensor. This sensor is capable of reading atmospheric pressure and reporting it to the Raspberry Pi. When first testing this sensor and comparing the values to data from the Bureau of Meteorology (BOM) it was found that the sensor consistently reported a pressure 10 hPa below the correct reading. This is well below the expected ± 1.5 hPa reported in the datasheet for this sensor (Doc-7) indicating that there was an error. This error was the temperature offset recorded by the device. This is a common issue reported by the Enviro+ community to the point where there are two different forum pages (both on GitHub (Doc-4) and the Pimoroni forums (Doc-5)) reporting errors in the Enviro+ reading and solutions to these problems. One such solution that was implemented and modified for this project was created by using a standard altitude and temperature compensation formula for pressure then multiplying this variable by the pressure reading output by the BME280. The user that suggested and implemented this method also noted that the temperature of the BME280 sensor did not greatly impact the reading, but altitude did. The code implementation of the pressure reading can be seen below. The modification made to this code for this project was changing how the altitude was collected. In the implementation seen in Doc-4 the altitude used was collected from a configuration file. To streamline the process and to allow the user more control, this system requests the sensors altitude. In the future, this could be updated based on imagery data or UAV onboard sensors to update the altitude while the UAV is in flight to provide a more accurate data reading then possible by this implementation. The function used to find the adjusted pressure can be seen in Figure 9:

```
149     # Gets the Pressure in hPa
150     def press(self):
151         temp = bme280.get_temperature()
152         adj = math.pow(1 - (0.0065 * float(alt)/(temp + 0.0065 * float(alt) + 273.15)), -5.257)
153         #calibration equation provided by Roscoe27 in this forum:
154         # https://github.com/pimoroni/enviroplus-python/issues/67
155         # The user used a standard formula for altitude and temperature compensation
156         # they seem to work within the error tollerances of the
157         # pressure sensor
158         pressure = bme280.get_pressure()
159         adj_pres = pressure * adj
160         # print("this is the sensor pressure " + str(pressure))
161         # print("this is the adjusted pressure " + str(adj_pres))
162         return adj_pres
```

Figure 9: Pressure Code



5.2.2.4 Humidity

The final sensor that is part of the BME280 chip is the humidity sensor. Once again due to the error in the sensor's temperature recording the initial reading was also incorrect without compensation. Similar to the pressure sensor this is a common issue reported in the Enviro+ forums. However, due to the regularity of these issues there are also solutions proposed and implemented. One of these solutions was adapted for this application to correct the humidity readings. In this case, the user undertook a regression analysis and applied a quadratic compensation to the humidity readings. When implementing this solution, (seen in Doc-4 and Doc-5) it was found that the reading was still inaccurate when compared to values recorded by BOM. As this is dealing with weather data this was expected as different factors affect the environmental conditions and compensation values depending on the location. Due to this, some modification of the variables (especially the humidity intercept) was required to better suit this application. The code implementation of this function to collect humidity readings can be seen in Figure 10:

```
164 def humid(self):
165     # calibration of humidity code created by user Roscoe27 in this forum
166     # https://forums.pimoroni.com/t/enviro-readings-unreliable/12754/58
167     humi_intercept = 13.686 # was originally at 15.686 this seems to work better
168     comp_hum_slope = 0.966
169     humi = bme280.get_humidity()
170     raw_temp = bme280.get_temperature()
171     dew_point = (243.04 * (math.log(humi / 100) + ((17.625 * raw_temp) / (243.04 + raw_temp)))) / (17.625 - math.log(humi / 100) - (17.625 * raw_temp / (243.04 + raw_temp)))
172     temp_adjusted_hum = 100 * (math.exp((17.625 * dew_point) / (243.04 + dew_point))) / math.exp((17.625 * self.temperature) / (243.04 + self.temperature))
173     comp_hum = comp_hum_slope * temp_adjusted_hum + humi_intercept
174     # print("This is the dew point " + str(dew_point) + "°C")
175     # print("This is the sensor humidity " + str(humi) + "%")
176     # print("This is the adjusted Humidity " + str(comp_hum) + "%")
177     return comp_hum
```

Figure 10: Humidity Code

5.2.2.5 Light and Proximity

Both of these data types are collected by the LTR-559 light and proximity sensor. As these readings are not affected by the PCB temperature there was no need to calibrate either of these sensors. In testing it was seen that the light sensor was able to react quickly and appropriately to light being shown on it. However, when testing the proximity sensor, it was interesting to see that when a finger was placed on the sensor it initially read zero, then as the finger got closer the value increased up to 2047. It was expected that the opposite response would occur. This did not ultimately effect anything but was a consideration when implementing the LCD display code seen in section 5.2.2.7. As seen in Figure 11 the code used to operate the light and proximity sensor did not require a separate function and is instead run as part of the update cycle. This was to improve efficiency and decrease the number of times it was called. The function calls in the update cycle are indicated by red arrows:

```
33 def update(self):
34     sensor = gas.read_all()
35     self.co = sensor.reducing
36     self.no2 = sensor.oxidising
37     self.nh3 = sensor.nh3
38     → self.light = ltr559.get_lux()
39     → self.proximity = ltr559.get_proximity()
40     self.co_ppm = co_ppm(self.co)
41     self.no2_ppm = no2_ppm(self.no2)
42     self.nh3_ppm = nh3_ppm(self.nh3)
43     self.temperature = self.temp()
44     self.pressure = self.press()
45     self.humidity = self.humid()
```

Figure 11: Light and Proximity Code

5.2.2.6 Gasses

As one of the requirements of this project was to report on detectable gasses the gas sensor on the Enviro+ was used to do this. This sensor is capable of detecting three primary gasses by reading the resistance values of the air. These primary gasses are carbon monoxide (reducing sensor), nitrogen dioxide (oxidising sensor) and ammonia. Even though these are the gasses targeted by the sensor it is also possible to detect other gasses that return similar resistance values. To see a full list of these as well as the resistance curves refer to appendix D. As stated, before part of the calibration stage for this subsystem included running the gas sensor for five minutes before the project. This was completed for two reasons. The first reason is that a baseline reading of the current air readings needs to be taken in order to find the correct concentration of gas. The second reason is, this sensor needs to be warmed up before use. This should be a longer process (around +100 minutes), however, due to time constraints this was not possible. The other point of note with the gas readings is that they are no way accurate. As the manufacturer have only provided the equation curves and not the equations themselves it is not possible to accurately calibrate these sensors to any particular gas or reading. It is possible to convert the resistance values into a parts per million (PPM) concentration using equations. However, this should only be used as a guide and not a truthful representation of the current environment. Similar to the humidity and pressure sensors, there is an active forum which assists with implementing a PPM conversion (see Doc-6). It was stated numerous times on this page that the data was not reliable, but it was possible to convert the main three gasses to PPM values using equations. When these equations were implemented in this project, the resulting gasses seemed to be accurate for the room environment the test was run in. These values were confirmed against average room data for CO, NO₂ and NH₃. The code implemented to calculate and collect the PPM values can be seen in Figure 12:

```
215 def co_ppm(Rs):
216     R0 = float(CO_cal)
217     CO_ppm = math.pow(10, -1.25 * math.log10(Rs/R0) + 0.64)
218     # equation provided by Roscoe27 sourced from
219     # https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/5
220     # print('this is the CO ppm ' + str(CO_ppm))
221
222     return CO_ppm
223
224 def no2_ppm(Rs):
225     R0 = float(NO2_cal)
226     NO2_ppm = math.pow(10, math.log10(Rs/R0) - 0.8129)
227     # equation provided by Roscoe27 sourced from
228     # https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/5
229     # print('this is the NO2 ppm ' + str(NO2_ppm))
230     return NO2_ppm
231
232 def nh3_ppm(Rs):
233     R0 = float(NH3_cal)
234     NH3_ppm = math.pow(10, -1.8 * math.log10(Rs/R0) - 0.163)
235     # equation provided by Roscoe27 sourced from
236     # https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/5
237     # print('this is the NH3 ppm ' + str(NH3_ppm))
238     return NH3_ppm
```

Figure 12: Gas Code

5.2.2.7 LCD Display

The last component of the Enviro+ and primary integrations of the AQS subsystem is the LCD display. This display can operate as a colour display capable of showing both images and video. For this application it was a requirement that this screen could display the Raspberry Pi IP address on boot as well as the current temperature reading and live video feed (with detection) from the TAIP subsystem. Each of these functions would be rotated through in a cycle by using the proximity sensor to detect when the user wants to change the screen. As the functions used to develop this code were all placed in classes this made the implementation of this system very simple in that when information is requested the LCD function can call the required class function. Similar to the light and proximity sensor discussed earlier the LCD was not affected by the temperature of the PCB with the only error occurring during testing due to a loose wire. In this case the wire in question inverted the colour display. The code implemented to run the LCD and swap between the different modes can be seen in Figure 13. Note that this code was taken from the G19-Integration.py. More of this program can be seen in RD/32:

```

176         if(UseAQ):
177             if AQSSubsystem.proximity > 1500 and time.time() - last_page > delay:
178                 print("Change")
179                 mode = (mode + 1) % 3
180
181             last_page = time.time()
182
183         if mode == 0:
184             lcd_text_dis(get_ip())
185
186         if mode == 1:
187             lcd_text_dis(str(round(AQSSubsystem.temperature,3)) + " C")
188
189         if mode == 2:
190             # This is the image display
191             #Display image
192             im_pil = Image.fromarray( frame )
193             # Display the resulting frame
194             # Resize the image
195             im_pil = im_pil.resize(( WIDTH , HEIGHT ) )
196             # display image on lcd
197             disp.display( im_pil )

```

Figure 13: LCD Code




6 Conclusion

Overall, this subsystem consists of three primary hardware components. The enclosure, raspberry pi, and enviro+ sensor. Using these the system will be able to display the required data and send environmental data such as the surrounding temperature, pressure, and light. This will meet the requirements and needs provided in RD/2 and RD/1. **There were no issues with the integration of this subsystem with the rest of the project. The only major change involved is the script which was updated to run as a class making it easy for any subsystem to access the functions and data as needed (the class code segments are seen in section 5.2.2 of this report.** Table 4 below shows the requirements that are relevant to this subsystem and what designs are in place presently to meet them.

Table 4: Requirements Met by Final Design

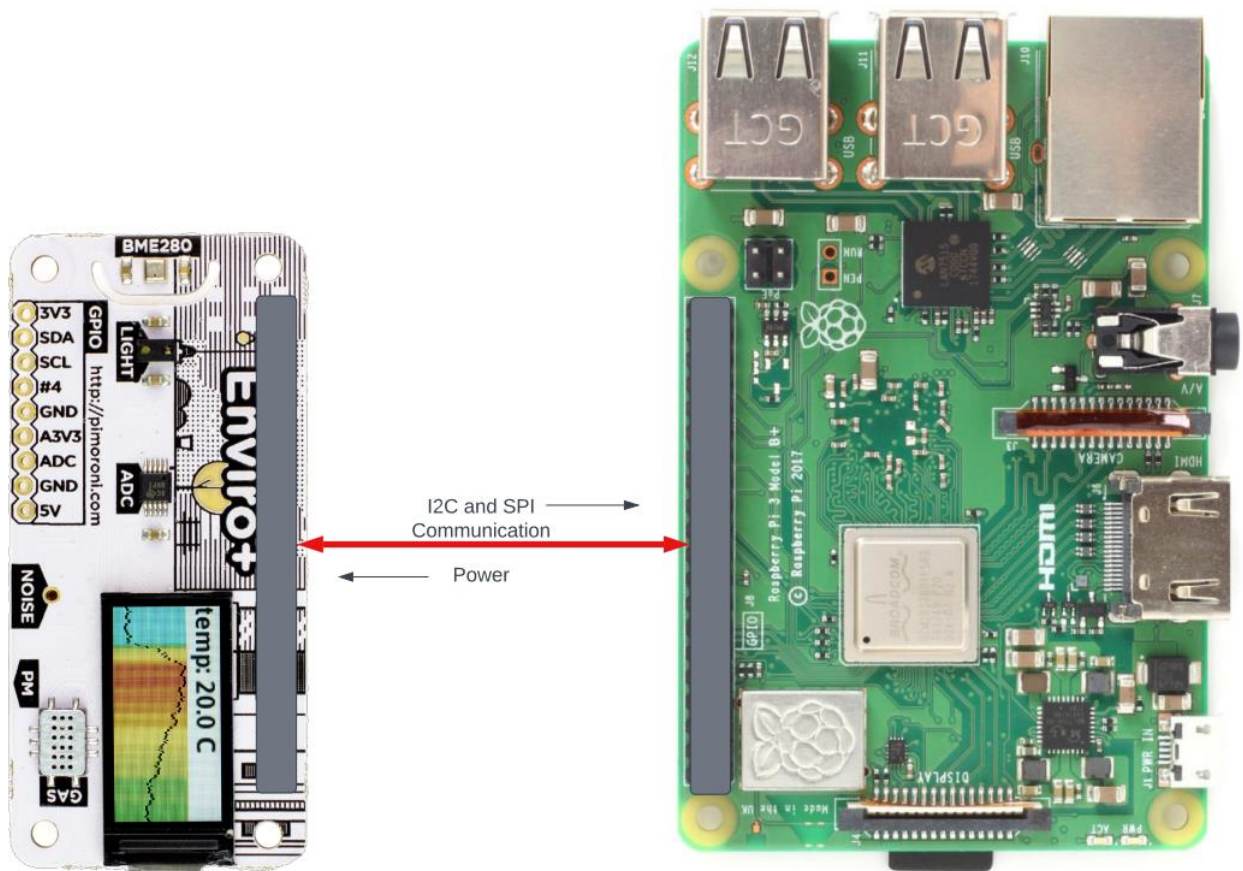
Requirement	Description	Requirements Met
REQ-M-01	The UAVPayloadTAQ shall remain under the maximum weight of 320 g and comply with an IP41 rating. The air quality sensors must be exposed to the environment to allow for accurate reading.	Met: - This requirement has been met as seen by the results of Hardware Test 1. The weight requirement has also been met but is discussed further in RD/20 and RD/28
REQ-M-02	The UAVPayloadTAQ must measure hazardous gases, (these include Carbone monoxide, Nitrogen dioxide, Ethanol, Hydrogen, Ammonia, methane, and Iso-butane) humidity, pressure, temperature and light via on-board sensors.	Met: - This has been met by all software tests completed. With the Enviro+ being used to accurately collect the required data.
REQ-M-03	The UAVPayloadTAQ shall communicate with a ground station computer to transmit video, target detection and air quality data.	Met: - This requirement has been met however, tests relating to it are described in the following test reports: RD/24, RD/25, RD/27 and RD/28
REQ-M-05	The Web Interface is required to display real time air sampling data that is recorded directly from the UAVPayloadTAQ and updated dynamically throughout the duration of the flight.	Met: - This requirement has been met however, tests relating to it are described in the following test reports: RD/24, RD/25, RD/27 and RD/28
REQ-M-10	Preliminary designs shall be completed by week 7	Met: - This has already been demonstrated through the submission of the assessment 1 reports
REQ-M-11	The payload should display its IP address via the integrated Enviro sensor LCD screen	Met: - This requirement has been met however, tests relating to it are described in the

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 27 of 37 Date: 28 October 2022
--	---	--

		following test reports: RD/25 and RD/28
REQ-M-12	The LCD screen should display live feed of target detection as well as temperature readings from the Pi and the Enviro sensor board.	Met: - This requirement has been met however, tests relating to it are described in the following test reports: RD/25 and RD/28
REQ-M-13	The LCD screen shall be placed on the side of the payload in order for the user to easily see its operation during flight.	Met: - This requirement has been met however, tests relating to it are described in the following test reports: RD/25 and RD/28
REQ-M-14	Developed solution shall conform to the systems engineering approach.	Met: - This has been demonstrated throughout the semester with reports showing this to be true
REQ-M-15	The system shall have logged functioning operation for a minimal period of 10 minutes prior to acceptance test	Met: - This requirement has been met however, tests relating to it are described in the following test reports: RD/24, RD/25, RD/27 and RD/28
REQ-M-19	Live data from the UAV must be made available through the web server within 10 seconds of capture.	Met: - This requirement has been met however, tests relating to it are described in the following test reports: RD/24, RD/25, RD/27 and RD/28

7 Appendix:


7.1 Appendix A Enviro+ connections:





7.2 Appendix B Enviro+ pin usage:

3v3 Power	1			2	5v Power
GPIO 2 (I2C1 SDA)	3			4	5v Power
GPIO 3 (I2C1 SCL)	5			6	Ground
GPIO 4 (GPCLK0)	7			8	GPIO 14 (PMS5003)
Ground	9			10	GPIO 15 (PMS5003)
GPIO 17	11			12	GPIO 18 (Mic i2s clk)
GPIO 27 (PMS5003 Reset)	13			14	Ground
GPIO 22 (PMS5003 Enable)	15			16	GPIO 23 (ADS1015 Alert)
3v3 Power	17			18	GPIO 24 (Gas Heater En)
GPIO 10 (SPI0 MOSI)	19			20	Ground
GPIO 9 (LCD D/C)	21			22	GPIO 25
GPIO 11 (SPI0 SCLK)	23			24	GPIO 8 (SPI0 CE0)
Ground	25			26	GPIO 7 (SPI CS)
GPIO 0 (EEPROM SDA)	27			28	GPIO 1 (EEPROM SCL)
GPIO 5	29			30	Ground
GPIO 6	31			32	GPIO 12 (Backlight)
GPIO 13 (PWM1)	33			34	Ground
GPIO 19 (Mic i2s fs)	35			36	GPIO 16
GPIO 26	37			38	GPIO 20 (Mic i2c data)
Ground	39			40	GPIO 21 (PCM DOUT)

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-AQS-FD-01 Issue: 1.0 Page: 30 of 37 Date: 28 October 2022
--	--	--

7.3 Appendix C Software and libraries used:

Software/Library	Link to information/home page
Python 3	https://www.python.org/
Enviro+ Library	https://github.com/pimoroni/enviropius-python

7.4 Appendix D MiCS-6814 (gas sensor) datasheet:

Data Sheet

MiCS-6814
1143 rev 8



The MiCS-6814 is a compact MOS sensor with three fully independent sensing elements on one package.

The MiCS-6814 is a robust MEMS sensor for the detection of pollution from automobile exhausts and for agricultural/industrial odors.



Features

- Smallest footprint for compact designs (5 x 7 x 1.55 mm)
- Robust MEMS sensor for harsh environments
- High-volume manufacturing for low-cost applications
- Short lead-times

Detectable gases

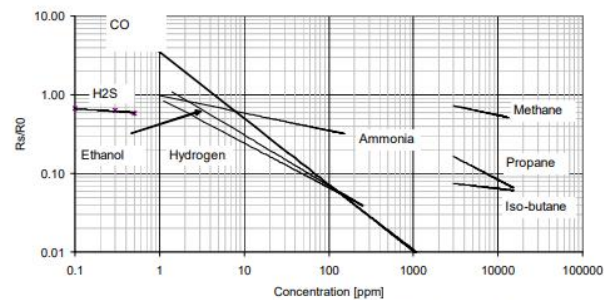
- | | | |
|--------------------|----------------------------------|--------------|
| • Carbon monoxide | CO | 1 – 1000ppm |
| • Nitrogen dioxide | NO ₂ | 0.05 – 10ppm |
| • Ethanol | C ₂ H ₅ OH | 10 – 500ppm |
| • Hydrogen | H ₂ | 1 – 1000ppm |
| • Ammonia | NH ₃ | 1 – 500ppm |
| • Methane | CH ₄ | >1000ppm |
| • Propane | C ₃ H ₈ | >1000ppm |
| • Iso-butane | C ₄ H ₁₀ | >1000ppm |

For more information please contact:

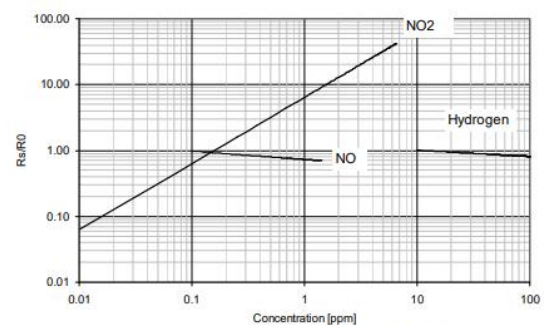
info.em@sgxsensortech.com

SGX Sensortech, Courtils 1
CH-2035 Corcelles-Cormondrèche
Switzerland

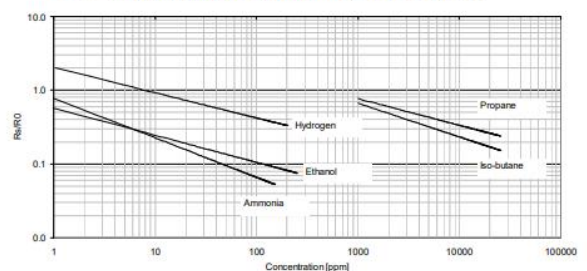
www.sgxsensortech.com



RED sensor, continuous power ON, 25°C, 50% RH



OX sensor, continuous power ON, 25°C, 50% RH



NH3 sensor, continuous power ON, 25°C, 50% RH



7.5 Appendix E Enviro+ code implementation:

```
# MIT License

# Copyright (c) 2020 Ross Fowler

# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

### please note that only individual equations were modified from the code this
licence relates to
# GitHub Link to full Licence: https://github.com/roscoe81/enviro-monitor/blob/master/LICENSE

import numpy as np
import sys
import cv2
import math
import time
from ltr559 import LTR559 #light and proximity
from enviroplus import gas #gas
import logging #for logging
import smbus2
from bme280 import BME280 #temp pressure and humidity
from datetime import datetime
import requests

bme280 = BME280()
ltr559 = LTR559()

delay = 0.5 # Debounce the proximity tap
mode = 0 # The starting mode
last_page = 0
light = 1
```




```
class AirQualitySensor:

    def __init__(self):
        self.calibration_setup()
        self.temperature = 0
        self.pressure = 0

    def start(self):
        print("Starting AQS")
        return

    def update(self):
        sensor = gas.read_all()
        self.co = sensor.reducing
        self.no2 = sensor.oxidising
        self.nh3 = sensor.nh3
        self.light = ltr559.get_lux()
        self.proximity = ltr559.get_proximity()
        self.co_ppm = co_ppm(self.co)
        self.no2_ppm = no2_ppm(self.no2)
        self.nh3_ppm = nh3_ppm(self.nh3)
        self.temperature = self.temp()
        self.pressure = self.press()
        self.humidity = self.humid()

    def read(self):
        return self.temperature, self.pressure

    def stop(self):
        return

    # this function is run once at the start of
    # the program to find the appropriate factor for the room
    def calibration_setup(self):
        global factor
        global alt
        global NO2_cal
        global CO_cal
        global NH3_cal
        alt = 0
        factor = 1
        adj_temp = 0
        cpu_temps = []

        cpu_temp = get_cpu_temperature()
        # Smooth out with some averaging to decrease jitter
        cpu_temps = cpu_temps[1:] + [cpu_temp]
        avg_cpu_temp = sum(cpu_temps) / float(len(cpu_temps))
```



```
        calibration_temp = input("Enter the current room temperature in degrees
celsius: ")
        # https://en-us.topographic-map.com/maps/vj4v/Ferny-Grove/
        alt = input("Enter the current altitude (above sea level) of your
position (in meters): ")
        gas_cal = input("Does the gas sensor require calibration? Type 'Yes' or
'No': ")
        start_time = time.time()
        finish_time = 0

        if str(gas_cal) == "No":
            try:
                with open("Gas_calibration.txt") as params:
                    content = params.read()
                    print(content)
                    content = content.split(" ")
                    index = 1
                    for x in content:
                        if x == "no2_cal":
                            NO2_cal = str(content[index])
                        elif x == "co_cal":
                            CO_cal = str(content[index])
                        elif x == "nh3_cal":
                            NH3_cal = str(content[index])
                        index += 1
            except:
                print("Calibration file does not exist enter values manually if
known:")

                print()
                NO2_cal = input("input the NO2 calibration value (in ohms): ")
                CO_cal = input("input the CO calibration value (in ohms): ")
                NH3_cal = input("input the NH3 calibration value (in ohms): ")
        elif str(gas_cal) == "Yes":
            # print("wah")
            min_count = 0
            timer = 0
            while(timer < 300 ): #5 minutes
                timer = finish_time - start_time
                #print(timer)
                sensor = gas.read_all()
                NO2_cal = sensor.oxidising # find the NO2 in ohms
                CO_cal = sensor.reducing # find the CO value in ohms
                NH3_cal = sensor.nh3 # find the ethanol in ohms
                finish_time = time.time()
                with open("Gas_calibration.txt", 'w') as calibration:
                    calibration.write("no2_cal " + str(NO2_cal) + " co_cal " +
str(CO_cal) + " nh3_cal " + str(NH3_cal))

            print("Calibrating the parameters: ")
```



```
start_time = time.time()
finish_time = 0
timer = 0
while(timer < 10 ):
    timer = finish_time - start_time
    cpu_temp = get_cpu_temperature()
    # Smooth out with some averaging to decrease jitter
    cpu_temps = cpu_temps[1:] + [cpu_temp]
    avg_cpu_temp = sum(cpu_temps) / float(len(cpu_temps))
    # print("this is the avg CPU temp " + str(avg_cpu_temp))
    finish_time = time.time()

while(adj_temp < float(calibration_temp)):
    raw_temp = bme280.get_temperature()
    adj_temp = raw_temp - ((avg_cpu_temp - raw_temp) / factor)
    factor += 0.01
    #print("this is the factor " + str(factor))

# gets the tempurature in degrees celsius
def temp(self):
    cpu_temps = []
    unit = "C"
    cpu_temp = get_cpu_temperature()
    # Smooth out with some averaging to decrease jitter
    cpu_temps = cpu_temps[1:] + [cpu_temp]
    avg_cpu_temp = sum(cpu_temps) / float(len(cpu_temps))
    # print("this is the avg CPU temp " + str(avg_cpu_temp))
    raw_temp = bme280.get_temperature()
    corrected_temp = raw_temp - ((avg_cpu_temp - raw_temp) / factor)
    # print("this is the adjusted tempurature " + str(corrected_temp))
    return corrected_temp

# Gets the Pressure in hPa
def press(self):
    temp = bme280.get_temperature()
    adj = math.pow(1 - (0.0065 * float(alt)/(temp + 0.0065 * float(alt) +
273.15))), -5.257)
    #calibration equation provided by Roscoe27 in this forum:
    # https://github.com/pimoroni/enviroplus-python/issues/67
    # The user used a standard formula for altitude and temperature
compensation
    # they seem to work within the error tollerances of the
    # pressure sensor
    pressure = bme280.get_pressure()
    adj_pres = pressure * adj
    # print("this is the sensor pressure " + str(pressure))
    # print("this is the adjusted pressure " + str(adj_pres))
    return adj_pres

def humid(self):
```



```
# calibration of humidity code created by user Roscoe27 in this forum
# https://forums.pimoroni.com/t/enviro-readings-unreliable/12754/58
humi_intercept = 13.686 # was originally at 15.686 this seems to work
better

comp_hum_slope = 0.966
humi = bme280.get_humidity()
raw_temp = bme280.get_temperature()
dew_point = (243.04 * (math.log(humi / 100) + ((17.625 * raw_temp) /
(243.04 + raw_temp)))) / (17.625 - math.log(humi / 100) - (17.625 * raw_temp /
(243.04 + raw_temp)))
temp_adjusted_hum = 100 * (math.exp((17.625 * dew_point)/(243.04 +
dew_point)) / math.exp((17.625 * self.temperature) / (243.04 + self.temperature)))
comp_hum = comp_hum_slope * temp_adjusted_hum + humi_intercept
# print("This is the dew point " + str(dew_point) + "C")
# print("This is the sensor humidity " + str(humi) + "%")
# print("This is the adjusted Humidity " + str(comp_hum) + "%")
return comp_hum

# converts the AQS data into a json object
def to_json(self):
    self.sensor = gas.read_all()
    current_time = datetime.now().strftime("%H:%M:%S")
    # print(current_time)
    AQS_data = {"Time": current_time, "Temperature C": float(self.temperature),
"Pressure kPa": float(self.pressure)/10, "Humidity %": float(self.humidity),
"Light Lux": float(self.light), "Proximity": float(self.proximity), "CO_ppm":
float(self.co_ppm), "NO2_ppm": float(self.no2_ppm), "NH3_ppm":
float(self.nh3_ppm)}

    return AQS_data

def co_ppm(Rs):
    R0 = float(CO_cal)
    CO_ppm = math.pow(10, -1.25 * math.log10(Rs/R0) + 0.64)
    # equation provided by Roscoe27 sourced from
    # https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/5
    # print('this is the CO ppm ' + str(CO_ppm))

    return CO_ppm

def no2_ppm(Rs):
    R0 = float(NO2_cal)
    NO2_ppm = math.pow(10, math.log10(Rs/R0) - 0.8129)
    # equation provided by Roscoe27 sourced from
    # https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/5
    # print('this is the NO2 ppm ' + str(NO2_ppm))
    return NO2_ppm
```



```
def nh3_ppm(Rs):
    R0 = float(NH3_cal)
    NH3_ppm = math.pow(10, -1.8 * math.log10(Rs/R0) - 0.163)
    # equation provided by Roscoe27 sourced from
    # https://forums.pimoroni.com/t/enviro-ohms-to-ppm/12207/5
    # print('this is the NH3 ppm ' + str(NH3_ppm))
    return NH3_ppm

def get_cpu_temperature():
    with open("/sys/class/thermal/thermal_zone0/temp", "r") as f:
        temp = f.read()
        temp = int(temp) / 1000.0
        # print('wah ' + str(temp))
    return temp

if __name__ == '__main__':
    AQS = AirQualitySensor()
    # Main loop for data collection
    while True:
        AQS.update()
        # print(AQS.to_json())
        r = requests.post(url = 'http://172.19.7.122:5001/aqs/aqs', json={"data":
AQS.to_json()})
        print(r)

        # time.sleep(1)

        # BME280 Sensor readings
        #temp() # corrected_temp
        #press() # pressure
        #humid() # comp_hum

        # Light and Proximity Sensor
        #light() # light
        #proximity() # prox

        #Gas Readings
        #co_ppm() # CO_ppm
        #no2_ppm() # NO2_ppm
        #nh3_ppm() # NH3_ppm
```