 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-DAT Issue: 1 Page: 2 of 19 Date: 23 October 2020
--	--	--

Revision Record

Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Preliminary Design Details	28 August 2020	Adrian Hiltunen
1.1	Functional Design Added	6 October 2020	Adrian Hiltunen
1.2	Final Design Details	23 October 2020	Adrian Hiltunen

Table of Contents


List of Figures and Tables	4
Definitions	5
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	7
3 Subsystem Introduction	8
4 Subsystem Architecture.....	9
4.1 Interfaces	9
5 Final Design Description.....	10
5.1 Software Driver Design.....	10
5.2 Gas Sensor Data Formatting.....	10
5.3 Temperature, Pressure & Humidity Sensor Data Formatting	13
5.4 Light & Noise Data Formatting.....	14
6 Conclusion.....	15
7 Appendix A – Driver Software	16
8 Appendix B – Gas Sensor Schematic	19

List of Figures and Tables

Figures	Page No.
Figure 1. <i>Data Acquisition Subsystem Architecture</i>	9
Figure 2. <i>Sensor Output Gradient</i>	10
Figure 3. <i>Top view and side view of additional heatsink without sensor hat attached</i>	13
 Tables	
Table 1. <i>Gradient and constant characteristics for the gas sensors</i>	12
Table 2. <i>Frequency bands chosen for analysis</i>	14

Definitions

UAV	Unmanned Aerial Vehicle
UAVPayload	Unmanned Aerial Vehicle Payload
GCS	Ground Control Station
CO2	Carbon Dioxide
WEB	Web Visualization Subsystem
DAT	Data Acquisition Subsystem
HLO	High Level Objective
ADC	Analog to Digital Conversion
PPM	Parts Per Million
V	Volt - Voltage
mA	Milliampere
ms	Millisecond
JSON	Java Script Object Notation
PPM	Part per million
FFT	Fast Fourier Transform
dBFS	Decibel below Full Scale

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-DAT Issue: 1 Page: 6 of 19 Date: 23 October 2020
--	--	--

1 Introduction

This document outlines a preliminary and final design for a data acquisition subsystem to be incorporated within a UAV Payload. The design process will follow a Systems Engineering process as a guide. This is an iterative process and much of the information in this document is expected to be updated as the subsystems integrate and as the project progresses. This document will aim to detail the preliminary design characteristics surrounding the data acquisition from a set of on-board sensors, and the communication of that acquired data to the networked WEB subsystem, outlined in RD/4.

1.1 Scope

The project includes a wide variety of deliverables and performs multiple tasks simultaneously. This document will outline the types of data and the information being gathered by the sensors as required by the system requirements in RD/2. This information will require communication between the DAT subsystem and WEB subsystem for access by the client, as requested in RD/1. Methodology and equipment will also be discussed. All the processing and acquisition of the data will be contained within the UAV Payload.

1.2 Background

The Queensland University of Technology (QUT) Airborne Sensing Lab have appointed Group 2 of the EGH455 (Advanced Systems Design) class to design a UAV Payload for indoor air quality to be installed on a S500 UAV designed for navigating in GPS denied environments. The UAVPayloadTAQ is required to conduct constant air quality sampling in a simulated underground mine. During monitoring, it must find and identify multiple markers placed by miners around the mine. Additionally, QUT Airborne Sensing Systems requires that the UAVPayloadTAQ is designed and developed using Systems Engineering to ensure QUT Airborne Sensing Systems requirements are met. Taken from RD/3.


2 Reference Documents

2.1 QUT Avionics Documents

RD/1	CN-UAVPayloadTAQ-01	UAV Payload Customer Needs
RD/2	SR-UAVPayloadTAQ-01	UAV Payload System Requirements
RD/3	PMP-UAVPayloadTAQ-01	UAV Payload Project Management Plan
RD/1	FD-UAVPayloadTAQ-G2-WEB-01	Preliminary Web Visualisation Subsystem Design
RD/2	FD-UAVPayloadTAQ-G2-DAT-01	Preliminary Sensor Data Subsystem Design
RD/6	FD-UAVPayloadTAQ-G2-IMG-01	Preliminary Image Processing Subsystem Design
RD/7	FD-UAVPayloadTAQ-G2-ENC-01	Preliminary Payload Enclosure Subsystem Design

2.2 Non-QUT Documents

RD/8	https://shop.pimoroni.com/products/-enviro?variant=31155658457171	Pimoroni Enviro+ Product Information
RD/9	https://learn.pimoroni.com/tutorial/sandyj/getting-started-with-enviro-plus	Pimoroni Enviro+ Getting Started
RD/10	https://www.sgxsensortech.com/-content/uploads/2015/02/1143_Datasheet-MiCS-6814-rev-8.pdf	Datasheet for Analog Gas Sensor
RD/11	https://github.com/pimoroni/enviropius-python	Driver Library for Sensor Hat

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-DAT Issue: 1 Page: 8 of 19 Date: 23 October 2020
--	--	--

3 Subsystem Introduction

A UAV Payload project required to perform complex tasks was broken down into separate subsystems. The subsystems must integrate and together provide the features provided in RD/2 to deliver a cohesive and functional system.

The data acquisition subsystem must provide features specified by the system requirements document RD/2. These are derived from HLOs in the customer needs document RD/1. Requirements identified that relate to the DAT subsystem are as follows.

- i. REQ-M-02/HLO-M-1: The UAV Payload device must measure and return readings of hazardous gases, air pressure, humidity, temperature, light, and noise levels via sensors on-board the device.
- ii. REQ-M-03, HLO-M-1: The UAV Payload must communicate wirelessly to a GCS via a standalone embedded system.
- iii. REQ-M-05, HLO-M-3: Real time air sampling data must be recorded and dynamically updated throughout the UAV's flight through a Web Interface.
- iv. REQ-M-11, HLO-M-1: A minimum period of 10 minutes' worth of logged data is mandatory pre-demonstration, known hereafter as an acceptance test.

The project team have been provided with the devices to be used therefore no alternative devices have been researched, fast tracking the development process. This is beneficial to the project timeline which is already constrained by the length of the project timeline and complexity of the project.

4 Subsystem Architecture

A graphical representation of the subsystem and the devices included are shown below. Outside of the sensors, the unit is expected to be in air, taking measurements of the surrounding environment.

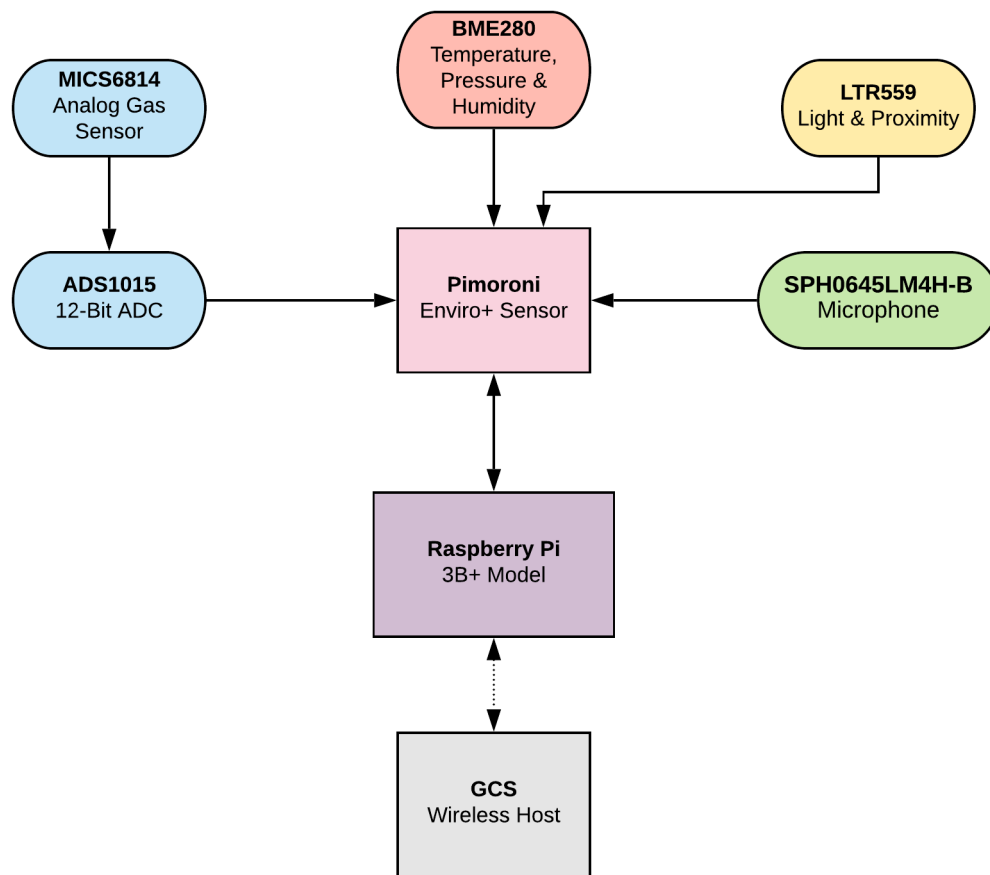



Figure 1. Data Acquisition Subsystem Architecture

4.1 Interfaces

All interfacing required for the DAT subsystem requires a physical connection to the Raspberry Pi that is the on-board device used for this project. All the sensors required for the project are located on one convenient package, the Pimoroni Enviro+ Sensor board, which was provided to the project team with the project brief. The board has been designed to connect directly into the header pins on the Raspberry Pi and communicate through an I2C protocol. The microphone is the only exception where it outputs digital audio through an I2S protocol, however it is seemingly unclear whether the Enviro+ sensor or the Raspberry Pi is converting the I2S data. The other exception is the Analog Gas Sensor. It has an analog output and requires an ADC which is supplied on the board and is used to interface the measurements to the device. The Raspberry Pi will then act as the communication device to transfer data over Wi-Fi to the host server detailed within the WEB subsystem.

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-DAT Issue: 1 Page: 10 of 19 Date: 23 October 2020
--	--	--

5 Final Design Description

The first HLO from RD/1 coincides with the second REQ provided in RD/2 is to measure air quality and report the data back to a remote station where a GPS denied environment may require assessment for safety concerns. The first item in the REQ-M-02 is the hazardous gases. The supplied Pimoroni Enviro+ Sensor is deemed unsuitable for the task and not recommended to be deployed in a safety requirement scenario and is noted in RD/8 & RD/9 that the device is unsuitable for ascertaining specific PPM values but is suitable for discovering changes in atmospheric gases. A safety feature must be noted that a requirement of the device operation requires start-up time in an environment where the hazardous content of the air quality is known, such as outside a mineshaft, away from the area to be measured.

5.1 Software Driver Design

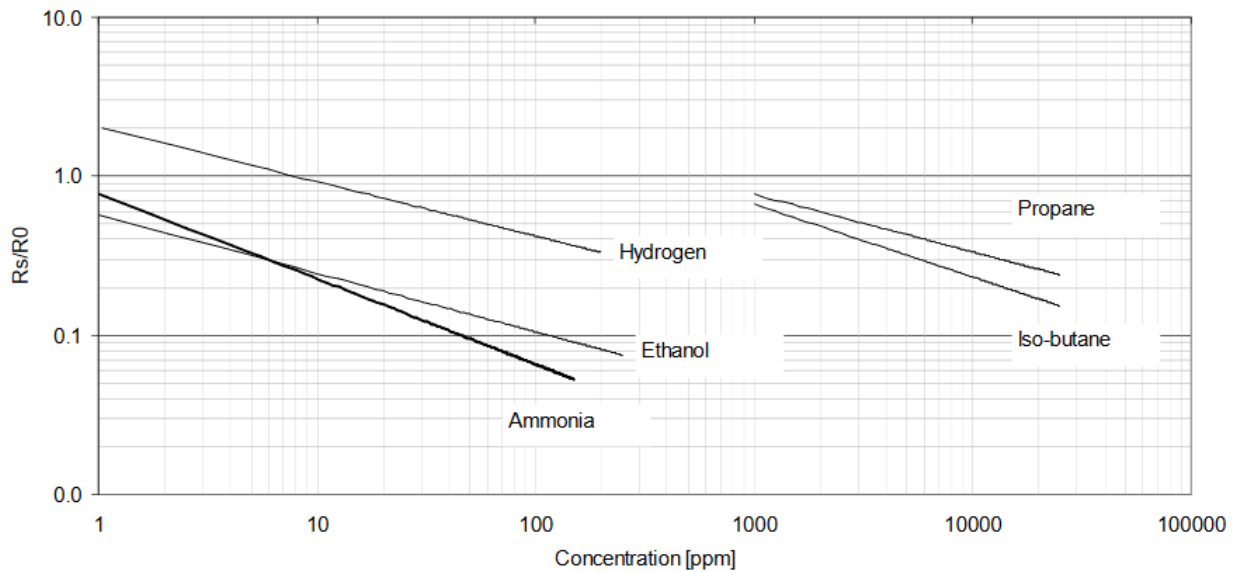
The developers of the Pimoroni Enviro+ Sensor hat have an open source driver library that demonstrates the functionality of the device and is available in RD/11. This reference document has been utilised as a skeleton file and for the initialisation of the device's individual sensors. The driver library supplied was only a fundamental beginning for further development, as the REQ-M-02 includes all the sensors returning data simultaneously and the library employed only a fraction of the sensors in concurrent operation.

Additionally, the driver library supplied in RD/11 did not output meaningful data in all cases. Careful consideration was taken to ensure the output of the DAT subsystem was meaningful and useful to the viewer. The subsequent data formatting will be explained in the following sections.

The script all-in-one.py was modified to include the missing sensors, namely the microphone and noise sensor information. This required an additional module from the driver library, noise.py to be imported. A dictionary element was created to conglomerate all the sensor values into one transportable object. This code section can be seen in the Appendix A.

5.2 Gas Sensor Data Formatting

The device used is the MICS6814 Analog Gas Sensor and communicates to the host via the Raspberry Pi, through a Python Library (provided) that reads the resistance values obtained through the 12-bit ADC inline. These resistance values can be converted to a value representing an approximation of the gas concentration in PPM, but first must be converted from a log-log graph provided in the datasheet, RD/10. This requires some algebraic manipulation, due to dependent and independent variables being opposite to the traditional format. An example is shown below.



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

Figure 2. Sensor Output Gradient. Image from RD/10

R_s shown in the image above on the vertical axis is a sense resistor. RD/10 and a schematic for a similar Pimoroni board utilising the same device make mention of 56 kilo-Ohm resistors being used, which are shown in Appendix B as R9, R10 and R15. This value is also found within the supplied Python library, where extraction of the resistance is being performed within the software. It should be noted that RD/9 mentions the three gas sensors only detect groups of gases, where a main reactive compound is shown on the graph in a bold line, however the mechanical sensor is sensitive to other types of gases, shown with thinner lines on the graph. The three main gases and supply characteristics of the sensors are:

- | | | | |
|---------------------|--------------------|-------|-------|
| 1. Carbon Monoxide | Reductional Sensor | 2.4 V | 32 mA |
| 2. Nitrogen Dioxide | Oxidising Sensor | 1.7 V | 26 mA |
| 3. Ammonia | NH3 Sensor | 2.2 V | 30 mA |

The three individual sensors require heating from a specific voltage applied across a heating element in the device. This device takes time to heat up and adds to the start-up time of the system. With a 5V supply to the heating element, the heating voltages for each element is shown above. With insufficient heating, data from the device will be inaccurate.

To extract a meaningful PPM value from the raw resistance data the following mathematical formulae were utilised. To find the gradient of a Log-Log graph, denoted as m , the graph was treated as a straight line and the following formula was used.

$$m = \frac{\log(F_2) - \log(F_1)}{\log(x_2) - \log(x_1)} = \frac{\log(F_2/F_1)}{\log(x_2/x_1)} \dots 1$$

Where F is the independent variable, the functions return value, or the y-axis. Picking two points on the graph provided in the datasheet provided the information required. However, it can be noted in figure 2 that the dependent and independent variables are displayed on the wrong axes, so I have flipped them respectively for the calculations that follow.

Manipulating equation 1 and inverting the logarithms simplifies the expression for the independent value to the following.

$$F(x) = \text{constant} \cdot x^m \dots 2$$

Further manipulation of equation 2 yields a value for the constant, denoted C in equation 3 below.

$$C = \frac{F(x)}{x^m} \dots 3$$

Using these formulae, the following table was produced for each of the three gas sensors using the most common and likely gases as values for the trendline. It should be noted that the x and y values are flipped to accommodate the dependent and independent variable format used in equation 2.

SENSOR	X1	X2	Y1	Y2	M	C
RED	3.5	0.01	1	1000	-1.179	4.382
OX	0.065	40	0.01	6	0.996	0.152
NH3	0.78	0.065	1	100	-1.853	0.631

Table 1: Gradient and constant characteristics for the gas sensors

With the values found in the table above, a PPM value could be displayed by entering the raw data into equation 2 before the data was populated as an element in a dictionary that was to be sent to the web server as a JSON object. The data formatting can be viewed in Appendix A

5.3 Temperature, Pressure & Humidity Sensor Data Formatting

The next group of items in the requirements are the temperature, pressure, and humidity sensors. A device made by Bosch has been integrated into the Enviro+ board, the BME280. The device has operating ranges between -40°C to 85°C, 300 hPa to 1100 hPa and 20-80% relative humidity. The device requires a 3.3V supply and has a maximum current draw of 0.74 mA while taking measurements. The device communicates via I2C.

The driver library previously mentioned included a formatted output and meaningful output value for both the pressure and humidity values. Humidity was displayed as a percentage and pressure readings in millibars.

The temperature value required modification as noted in the test report provided for this subsystem. To summarise, temperature calibration was required due to the addition of a heatsink on the Raspberry Pi CPU, shown below.



Figure 3. Top view and side view of additional heatsink without sensor hat attached

It can be noted that this was very close to the bottom of the sensor hat when attached. Being so close to the sensor hat required modification to the driver library where a compensation factor was being applied to the temperature readout. This modification to the temperature value was calibrated using a thermocouple and a multi-meter with a temperature reading. The CPU temperature was also used to compensate for the value read out, as shown in Appendix A.

5.4 Light & Noise Data Formatting

Light sensing is achieved using an opto-electric device, the LTR559. This device requires a 3.3 V supply and only draws 0.25 mA of supply current. The light sensor on board had a Lux value readout within the driver library and was calibrated within the initialisation of the device.

The last member of REQ-M-02 is noise level metering. This was achieved through a SPH0645LM4H-B requiring a 3.3 V supply and 0.6 mA supply current. This device is a transducer that outputs an analogue voltage and then converts the signal through a built in ADC with the help of a sigma-delta encoder, to digital audio output through I2S protocol. This digital audio was captured in 250 ms segments for analysis by a Fast Fourier Transform (FFT) on three audio bands considered to carry important information to humans. These were specified in the driver and include the three categories listed below.

FREQUENCY BAND	DESCRIPTION
100-200 Hz	Low frequency content, rumbling, explosion
500-600 Hz	Low-mid frequencies, grinding, crunching
1000-1200 Hz	Mid-high frequencies, whistling, crackling

Table 2. Frequency bands chosen for analysis

The documentation did not provide a maximum level output value which is why the amplitude of the FFT has been returned as a relative value. A dBFS value would have been ideal however the project team agreed that a relative value would be sufficient in noting any extreme noise event, alerting the user to possible danger in the output data stream. The 0 dBFS value would have required the possible destruction of the transducer to obtain the maximum value through experimentation and was deemed not a viable option. As the output graph in the final integrated product displayed the last 50 data points in each of the sensors data streams, all information would be relative and easily conspicuous if a sharp gradient was apparent, notifying the user of possible danger. This is regardless of units in the noise profile. Another point to consider, being mounted so close to the drone and the engines and air pressure turbulence provided by the thrust, any sharp increase in noise levels over this would be deemed as noteworthy.

6 Conclusion

This final design document outlines all the requirements pertaining to the DAT subsystem outlined in RD/2 have shaped the design of the system. The nature of this document serves as a final design overview of the air quality data and sensing subsystem.

The air quality data acquisition subsystem was designed to fulfill all the requirements as specified in RD/2 and outlined in section 3 of this report. The Test Report document TR-UAVPayloadTAQ-G2-DAT verifies the conditions being met and the successful operation of the subsystem pre and post integration with the other subsystems in RD/4, RD/6 and RD/7.

The software driver for the device shown in Appendix A shows the completed python script module “returnSensors” which returns a dictionary with the formatted and meaningful data types as requested by the project brief, in real time, for communication between the Raspberry Pi and the GCS.

The last loop was used for testing procedures outlined in the Test Report.



7 Appendix A – Driver Software

```
Set as interpreter
1  #!/usr/bin/python3
2  import numpy
3  import time
4  import colorsys
5  import sys
6
7  try:
8      # Transitional fix for breaking change in LTR559
9      from ltr559 import LTR559
10     ltr559 = LTR559()
11 except ImportError:
12     import ltr559
13
14 from bme280 import BME280
15 from pms5003 import PMS5003, ReadTimeoutError as pmsReadTimeoutError
16 from enviroplus import gas
17 #from enviroplus import noise
18 from enviroplus.noise import Noise
19 from subprocess import PIPE, Popen
20 import logging
21
22 logging.basicConfig(
23     format='%(asctime)s.%(msecs)03d %(levelname)-8s %(message)s',
24     level=logging.INFO,
25     datefmt='%Y-%m-%d %H:%M:%S')
26
27 logging.info("""DataSensing455.py - Displays readings from all of Enviro plus' sensors
28 Press Ctrl+C to exit!
29 """)
```

```
31 # BME280 temperature/pressure/humidity sensor
32 bme280 = BME280()
33
34 # PMS5003 particulate sensor
35 pms5003 = PMS5003()
36
37 # Microphone instance
38 noise = Noise()
39
40 # Get the temperature of the CPU for compensation
41 def get_cpu_temperature():
42     process = Popen(['vcgencmd', 'measure_temp'], stdout=PIPE, universal_newlines=True)
43     output, _error = process.communicate()
44     return float(output[output.index('=') + 1:output.rindex('"')])
45
46
47 # Tuning factor for compensation. Decrease this number to adjust the
48 # temperature down, and increase to adjust up
49 factor = 1.3
50
51 delay = 0.5 # Debounce the proximity tap
52 mode = 0    # The starting mode
53 last_page = 0
54 light = 1
```




```
56 # Initialise a values dict to store the data
57 sensors = ["oxidised",
58            "reduced",
59            "nh3",
60            "humidity",
61            "light",
62            "noise",
63            "pressure",
64            "temperature"]
65
66 values = {}
67
68 for sensor in sensors:
69     values[sensor] = 0
```

```
71 def updateSensors():
72
73     data = gas.read_all()
74     data_unraised = [56000.0/data.oxidising, 56000.0/data.reducing, 56000.0/data.nh3]
75     data_powers = [0.996, -1.179, -1.853]
76     data_raised = numpy.power(data_unraised, data_powers)
77
78     cpu_temp = get_cpu_temperature()
79     raw_temp = bme280.get_temperature()
80     compensated_temp = raw_temp - ((cpu_temp - raw_temp) / factor)
81
82     low, mid, high, amp = noise.get_noise_profile()
83
84     values["oxidised"] = data_raised[0]*0.152
85     values["reduced"] = data_raised[1]*4.382
86     values["nh3"] = data_raised[2]*0.631
87     values["humidity"] = bme280.get_humidity()
88     values["light"] = ltr559.get_lux()
89     values["noise"] = amp*64
90     values["pressure"] = bme280.get_pressure()
91     values["temperature"] = compensated_temp
```



```
93 def returnSensors():
94
95     data = gas.read_all()
96     data_unraised = [56000.0/data.oxidising, 56000.0/data.reducing, 56000.0/data.nh3]
97     data_powers = [0.996, -1.179, -1.853]
98     data_raised = numpy.power(data_unraised, data_powers)
99
100     cpu_temp = get_cpu_temperature()
101     raw_temp = bme280.get_temperature()
102     compensated_temp = raw_temp - ((cpu_temp - raw_temp) / factor)
103
104     low, mid, high, amp = noise.get_noise_profile()
105
106     values["oxidised"] = data_raised[0]*0.152
107     values["reduced"] = data_raised[1]*4.382
108     values["nh3"] = data_raised[2]*0.631
109     values["humidity"] = bme280.get_humidity()
110     values["light"] = ltr559.get_lux()
111     values["noise"] = amp*64
112     values["pressure"] = bme280.get_pressure()
113     values["temperature"] = compensated_temp
114
115     return values
```

```
117     # set number of iterations
118     loopVariable = 20
119
120     while loopVariable > 0:
121         loopVariable -= 1
122         updateSensors()
123
124         print("*-----*")
125         for sensor in sensors:
126             print(sensor)
127             print(values[sensor])
128
```



8 Appendix B – Gas Sensor Schematic

MICS-6814 Analog NO2 & CO & NH3 Sensor

OX (NO2): for 5V
R3 resistor target current: 26mA
R3 resistor selection: $(5V - 1.7V) / .026A = 126.9 \text{ ohm}$
R3 resistor power = $.026A * .026A * 126.9 \text{ ohm} = 86mW$
RED (CO):
R8 resistor target current: 32mA
R8 resistor selection: $(5V - 2.4V) / .032A = 81.3 \text{ ohm}$
R8 resistor power = $.032A * .032A * 81.3 \text{ ohm} = 83mW$
NH3:
R15 resistor target current: 30mA
R15 resistor selection: $(5V - 2.2V) / .03A = 93.3 \text{ ohm}$
R15 resistor power = $.03A * .03A * 93.3 \text{ ohm} = 84mW$

