**Wireless Distributed Air Quality Sensor Network for Measuring an Individual’s Exposure to Urban Air Pollution**

*SeungDoo Yang1,2, Barak Fishbain2*

*1Dvision of Engineering Science, Faculty of Applied Science and Engineering, University of Toronto; 2Division of Environmental, Water and Agricultural Engineering, Faculty of Civil and Environmental Engineering, Technion—Israel Institute of Technology*

July 29, 2017

**ABSTRACT**

It is important to measure air pollution level that an individual is exposed to in their daily life, especially if the one suffers from a respiratory disease, such as asthma. The collected measurement data could be very useful in identifying the cause of the individual’s illness. Also, it could be used to analyze the effect of air pollution levels on the patients. To effectively monitor several individuals’ exposure to air pollution, this project aimed to develop a wireless distributed sensor network consisting of air quality sensing stations and a central database server where the data is saved.

**TABLE OF CONTENTS**

**ABSTRACT**

**1. INTRODUCTION**

**2. REQUIRED COMPONENTS**

**2.1. Raspberry Pi 3 Model B**

**2.1.1. Raspbian Operating System**

**2.2. SDS021 Particulate Matter Sensor**

**2.2.1. Connection**

**2.2.2. Data Communication**

**2.3. PMS5003 Particulate Matter Sensor**

**2.3.1. Connection**

**2.3.2. Data Communication**

**2.4. tshark, Aircrack-ng, and WiPi**

**2.4.1. Installation**

**2.4.2. Enabling Monitor Mode on WiPi**

**2.5. MySQL**

**2.5.1. Installation**

**2.6. Complete Setup**

**3. DATABASE**

**3.1. Credentials**

**3.2 Requirements**

**3.3 Tables**

**3.3.1. Log Table**

**3.3.2. AQmeasurement Table**

**3.3.3. Sensor Table**

**4. CODE**

**4.1. Developing Environment**

**4.2. Sensor Reader Class**

**4.2.1. SDS021 Sensor Reader**

**4.2.2. PMS5003 Sensor Reader**

**4.3. Functions**

**4.3.1. Air Quality Measurement**

**4.3.2. Detecting Nearby Devices**

**4.3.3. Uploading Data to Database**

**4.3.4. Checking Internet Connection**

**4.4. Top Level Code**

**4.4.1. Initialization**

**4.4.2. Main Code**

**5. REFERENCES**

**6. APPENDIX**

**2**

**4**

**5**

**5**

**6**

**7**

**7**

**7**

**9**

**10**

**11**

**14**

**14**

**14**

**15**

**15**

**16**

**17**

**17**

**17**

**18**

**18**

**18**

**19**

**20**

**20**

**21**

**21**

**23**

**25**

**25**

**26**

**27**

**28**

**29**

**29**

**29**

**31**

**32**

**1. INTRODUCTION**

The first step of this project was designing the overall hardware and software architecture of the air quality measuring station. The station consists of air quality measuring sensors, a detector that identifies individuals in its proximity, and a wireless transmitter that sends collected data to the central database server. Software installation of MySQL, setup of Wireshark and putting a Wi-Fi dongle in the monitor mode using Aircrack-ng tools for packet sniffing were part of this step.

Currently the sensor network includes two particulate matter concentration measuring stations controlled by Raspberry Pi 3 [1]. One station measures PM 1.0 μm and PM 2.5 μm concentration in unit of μg/m3 with SDS021 particulate matter sensor [2] and other PMS5003 [3]. Comparison of data acquired by the two sensors was made. The station detects any Wi-Fi enabled device in its proximity after the measuring of the particulate matter concentration with a Wi-Fi dongle in the monitor mode.

The second was designing the MySQL database that effectively relates and organizes the date, MAC addresses, air quality data, sensor type, location of the station, etc. The final design of the database consists of three tables.

The third step was putting everything together and coding the functionality of the sensor network. Its operation is: (1) fetch PM 1.0 and PM 2.5 concentration data from a sensor, (2) detect Wi-Fi enabled devices and save their MAC addresses, and (3) Send the collected data to the central database server when there is internet connection. (4) Repeat the previous steps with a given time interval.

**2. REQUIRED COMPONENTS**

**2.1. Raspberry Pi 3 Model B**

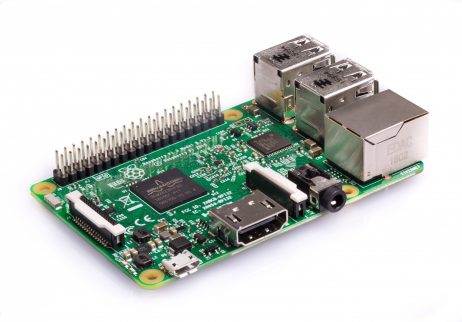
****

Figure 1. Raspberry Pi 3 Model B [1]

Table 1. Raspberry Pi 3 specifications [1]

|  |
| --- |
| **Specifications** |
| * Quad Core 1.2GHz Broadcom BCM2837 64bit CPU * 1GB RAM * BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board * 40-pin extended GPIO * 4 USB 2 ports * 4 Pole stereo output and composite video port * Full size HDMI * CSI camera port for connecting a Raspberry Pi camera * DSI display port for connecting a Raspberry Pi touchscreen display * Micro SD port for loading your operating system and storing data * Upgraded switched Micro USB power source up to 2.5A |

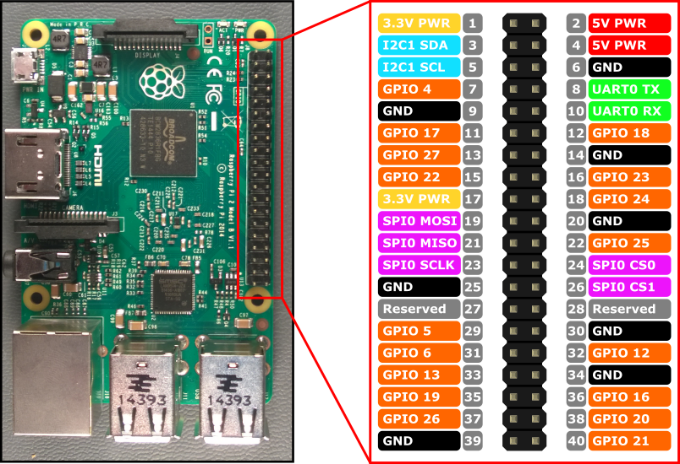


Figure 2. Raspberry Pi 3 GPIO pins [2]

What is shown in Figure 1 is **Raspberry Pi 3 Model B**, which is basically a small computer. It has four USB ports, an Ethernet port, HDMI port, integrated Wi-Fi dongle, GPIO pins shown in Figure 2, etc., as listed in Table 1 above. The air quality measuring station was built on Raspberry Pi. Raspberry Pi was selected as a controller for this project because it is capable of multitasking, has integrated WiFi for internet connection, and is able to use software necessary for this project (WireShark, Aircrack-ng and MySQL).

**2.1.1. Raspbian Operating System**

Raspbian operating system, one of the Linux distributions for Raspberry Pi, needs to be installed on the board. The operating system could be utilized by a graphical user interface by connecting a monitor through a HDMI cable, or by Terminal via SSH.

A tutorial for Installing Raspbian Operating System on Raspberry Pi could be easily found on Internet, such as from an official Raspberry Pi website:

[**https://www.raspberrypi.org/documentation/installation/installing-images/**](https://www.raspberrypi.org/documentation/installation/installing-images/)

After the installation, it is always a good idea to make sure the system is up to date. One can use these commands to check and update the system:

Update package lists that need upgrading, as well as new packages that have just come into the repositories:

sudo apt-get update

Upgrade the package lists:

sudo apt-get upgrade

**2.2. SDS021 Particulate Matter Sensor**



Figure 3. SDS021 PM Sensor [Appendix A]

Shown in Figure 3, SDS021 sensor uses laser scatter principle to the particle concentration between 0.3 and 10 μm in the air during its measurement. It sends digital output through the UART communication protocol. To see all of product specification, please refer to *Appendix A*. This sensor was selected as the first sensor of the air quality network system’s prototype because of its simple connection with Raspberry Pi through the USB interface and its reliable performance.

**2.2.1. Connection**

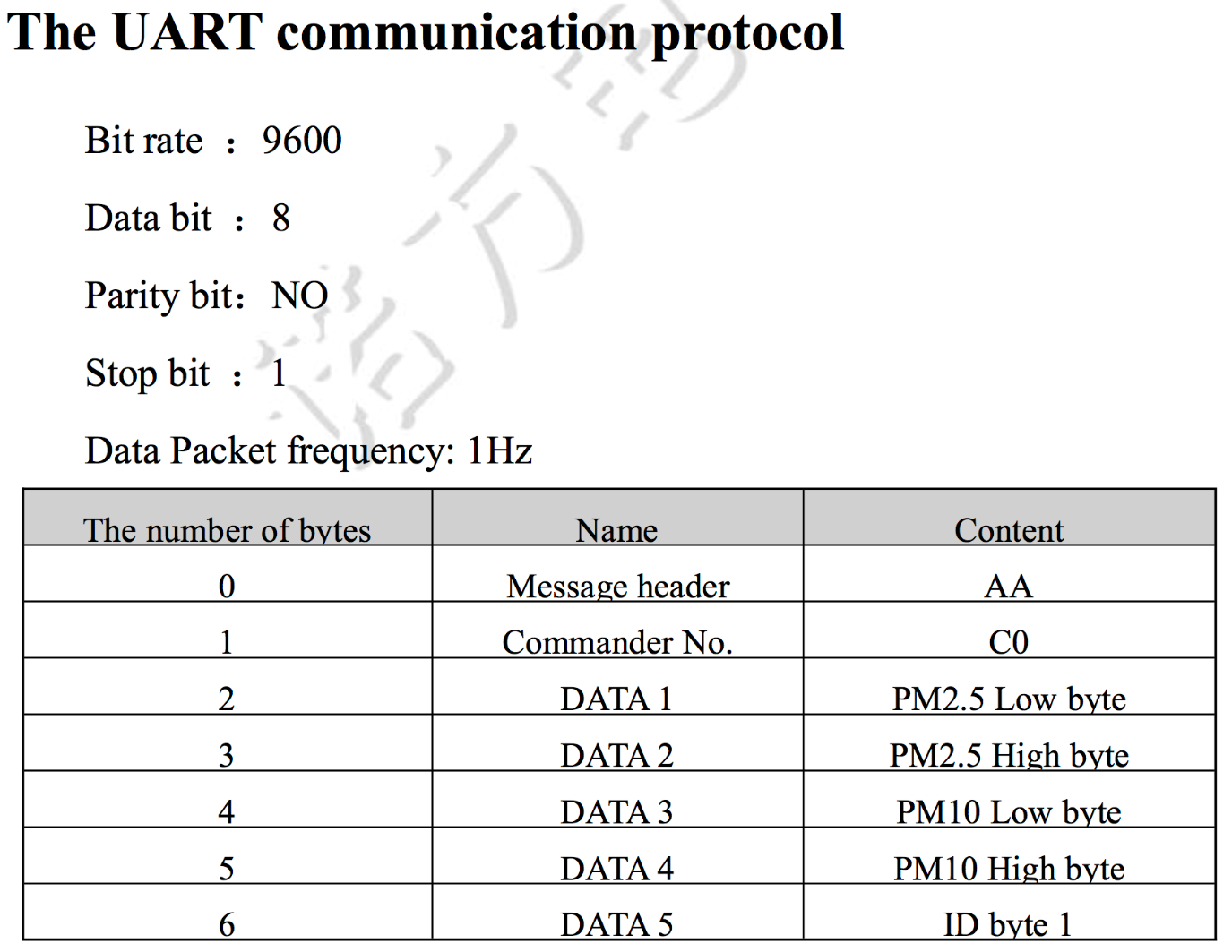


The connection between SDS021 sensor and Raspberry Pi was very simple. The sensor came with the USB interface shown in Figure 4 that simply is plugged into Raspberry Pi and the communication happens through **ttyUSB0** **port**. The purchase of the sensor was made here: <https://www.aliexpress.com/store/product/NOVA-PM2-5-Air-particle-dust-sensor-SDS021-laser-inside-digital-output-SDS021-Laser-PM2-5/1725971_32638192686.html>

Figure 4. The USB interface

**2.2.2. Data Communication**

Since connection is done by the USB interface and power supply to the sensor is also done by the interface, the only thing that needs paid attention in the sensor’s datasheet would be **UART communication protocol** shown in Figure 5 below. This information will be used in the code later to read the data values from the sensor.



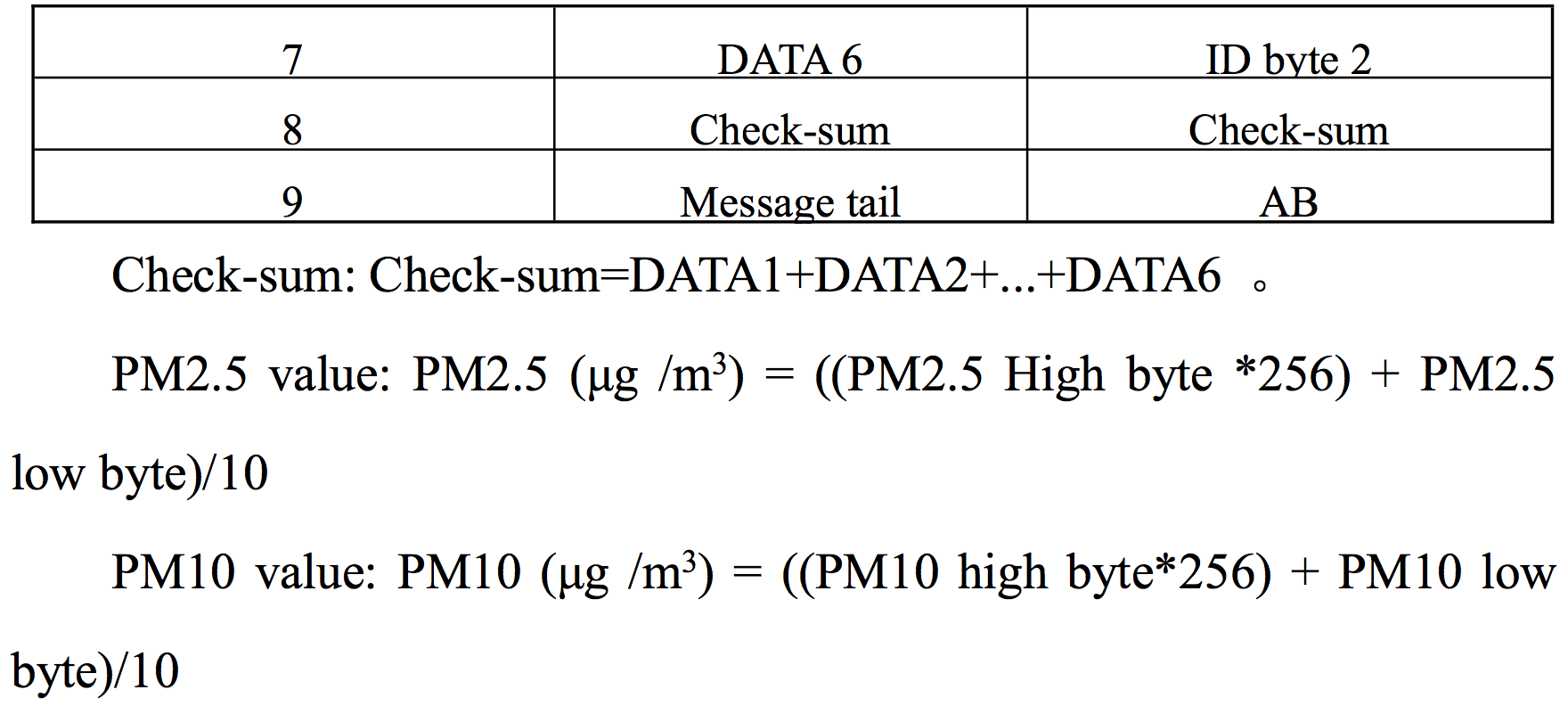


Figure 5. SDS021 UART Communication Protocol [Appendix A]

**2.3. PMS5003 Particulate Matter Sensor**



Figure 6. PMS5003 PM sensor [Appendix B]

Shown in Figure 6, PMS5003 also uses laser scattering principle to calculate the particle concentration in the air. It outputs digital signals through UART communication protocol. The communication works in the similar way as SDS021, but it has higher frame length for each data package because it can measure a lot more variety (12 different things). It is also consistent in performance within its working temperature even if it was running for a long time, such as 30 days [Appendix B], which is ideal for the air quality measuring station.

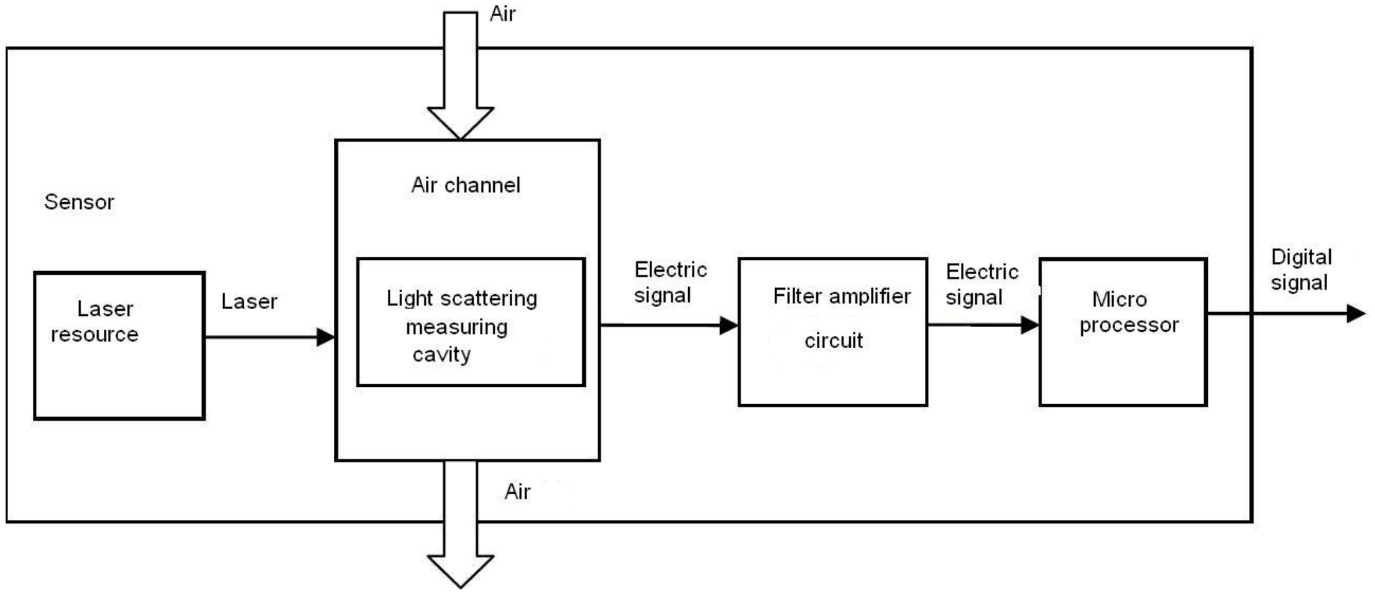


Figure 6. Functional Block Diagram of the sensor [Appendix B]

**2.3.1. Connection**

PMS5003 did not come with the USB interface so it had to be manually connected to Raspberry Pi’s GPIO pins. Figure 7 notes the pin assignments of PMS5003.

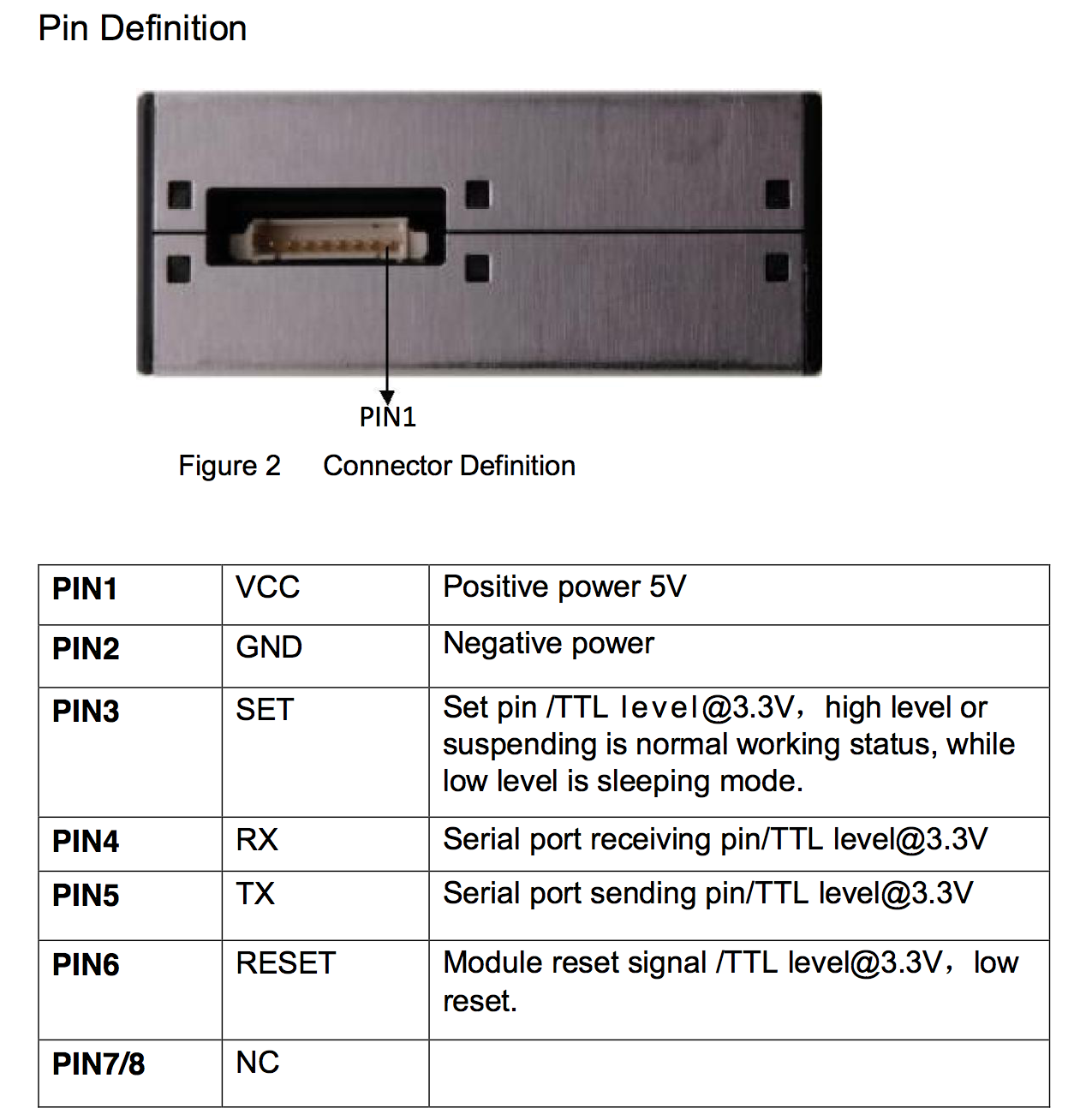
****

Figure 7. PMS5003 pin assignment [Appendix B]

The PMS5003 sensor was connected to Raspberry Pi as shown in Figure 9. Figure 8 is PMS5003’s typical circuit connection.



Figure 8. PMS5003 Typical Circuit Connection [Appendix B]

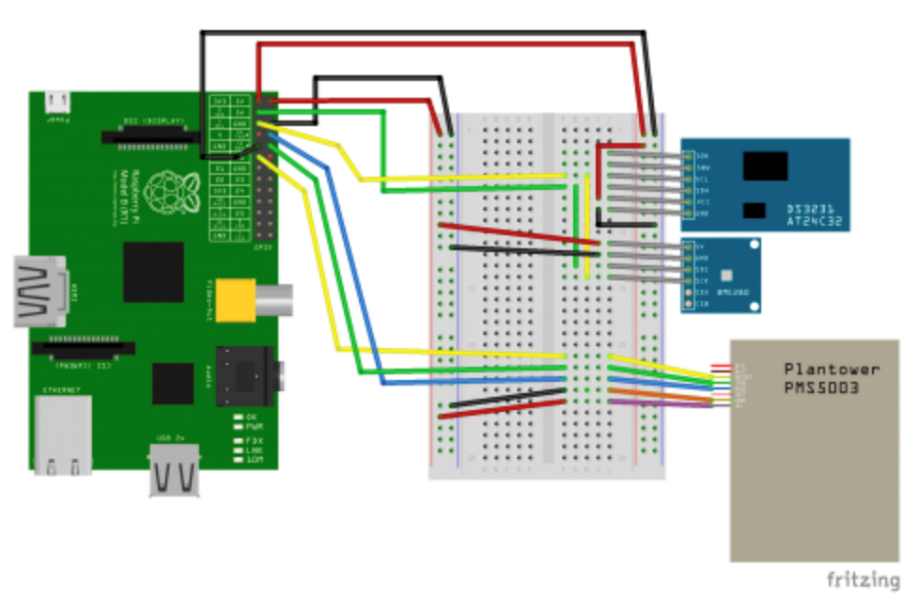
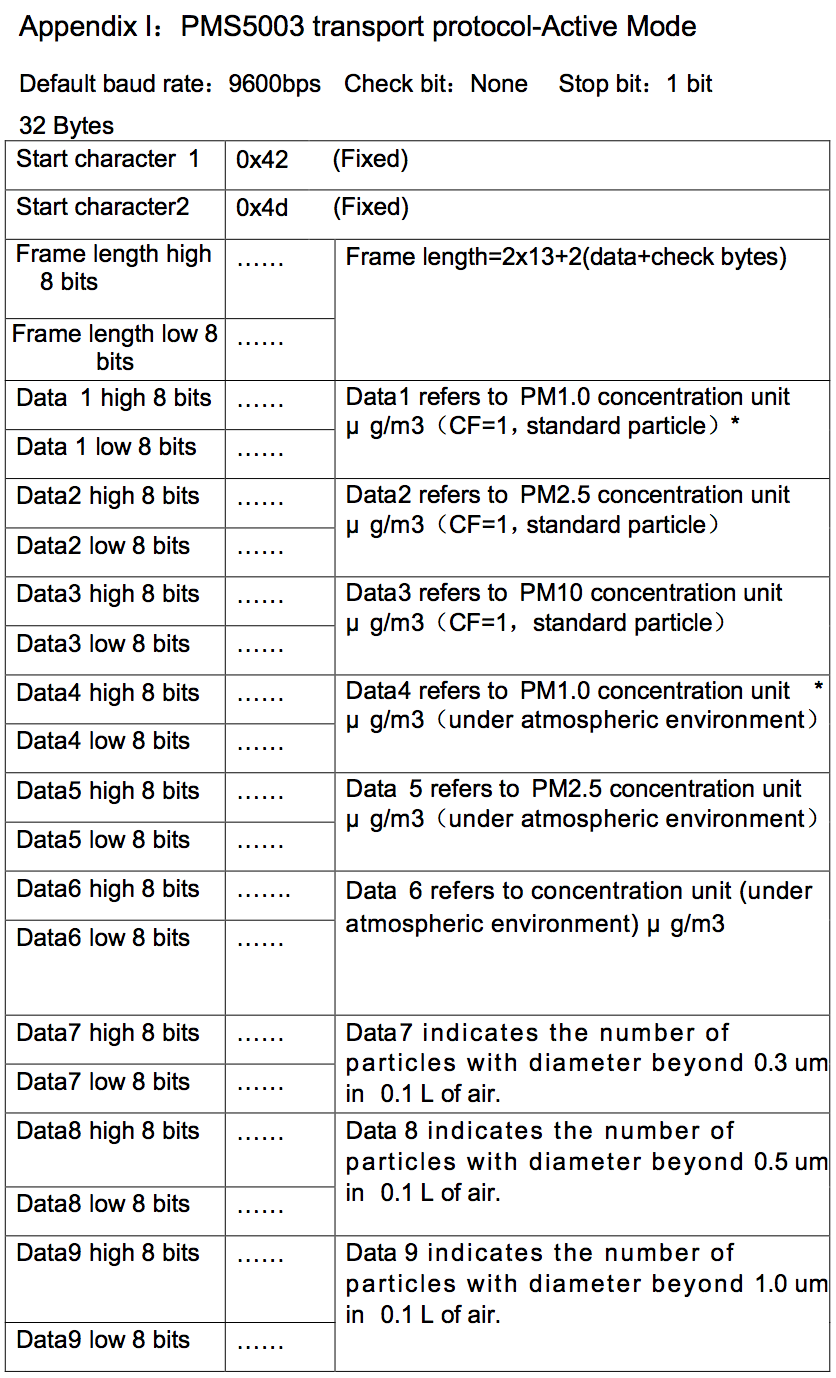


Figure 9. PMS5003 and RPi connection [3]

**2.3.2. Data Communication**

There are two types of protocols with PMS5003: active and passive. In this project, only active mode is used. Active mode sends serial data to the host automatically through **ttyS0 port**, which is what is desired for a stationary air quality measuring station. There are **12 different types of data** measured by the sensor and **each frame length is 32** **bytes**. Each data consists of 2 bytes, high and low bytes [Appendix B]. One simply needs to convert the 16 bits into an integer to obtain the measurement.



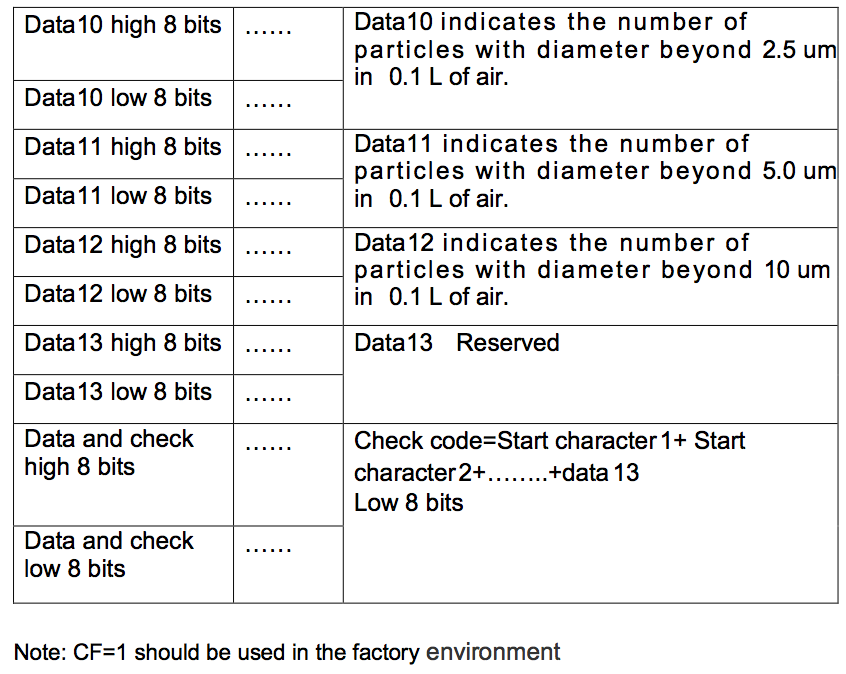


Figure 10. PMS5003 UART Communication Protocol [Appendix B]

**2.4. tshark, Aircrack-ng, and WiPi**

**tshark** is a widely-used network protocol analyzer. It can sniff packets in the air in is proximity if it is used with a WiFi dongle in the monitor mode. Using this, detecting nearby Wi-Fi enabled devices was possible.

**Aircrack-ng** is a set of hacking tools for network penetration tests. One of its tools involve putting a Wi-Fi dongle into the monitor mode. There are other ways to put the Wi-Fi dongle into the monitor mode but Aircrack-ng was selected because the Raspbian OS doesn’t automatically revert the change back into the normal mode if it was done with Aircrack-ng.

**2.4.1. Installation**

To install these two software, it just takes two simple and similar commands:

Install tshark:

sudo apt-get install tshark

Install Aircrack-ng:

sudo apt-get install aircrack-ng

**2.4.2. Enabling Monitor Mode on WiPi**

Monitor mode enables sniffing the packets in the air without connecting or associating with any access point. Not all Wi-Fi dongles in the market supports the monitor mode. **WiPi**, a Wi-Fi dongle for Raspberry Pi developed by *element 14,* supports the monitor mode (shown in Figure 10 below). The dongle is just plugged into one of the Raspberry Pi’s USB ports.



Figure 11. WiPi module [4]

To enable the monitor mode on WiPi, **Aircrack-ng**’s tools were used. To automatically put WiPi into the monitor mode at the boot, one can configure **rc.local** file. One can do so like this:

Open rc.local file with nano text editor:

sudo nano /etc/rc.local

After opening the rc.local file, simply type in the following two lines before the last line of the file, which is “exit 0.”

sudo airmon-ng start wlan0

sudo airmon-ng start wlan1

These two command lines try to enable the monitor mode of Wi-Fi dongles assigned to wlan0 and wlan1. If a Wi-Fi dongle supports the monitor mode, it is put into the monitor mode. If not, it just stays in the normal mode. As mentioned in the *Raspberry Pi 3 Model B* section above, the Raspberry Pi has an integrated Wi-Fi dongle. Sometimes after the boot, Wi-Pi and the integrated Wi-Fi dongle switches places between wlan0 and wlan1. To ensure that Wi-Pi will be put into the monitor mode after the boot, both lines are executed.

**2.5. MySQL**

MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL). It is used in this project to manage, query, and upload data to the database. SQL documentation could be viewed here: <https://www.w3schools.com/sql/>

To integrate MySQL into the running python script, MySQL python library was installed, which is named MySQLdb. Its documentation could be viewed here: <http://mysql-python.sourceforge.net/MySQLdb.html>

**2.5.1. Installation**

To install MySQL, type in the following command:

sudo apt-get install mysql-server

When prompter to enter a password for the root user, enter any password. It is not important for this project where database is located externally.

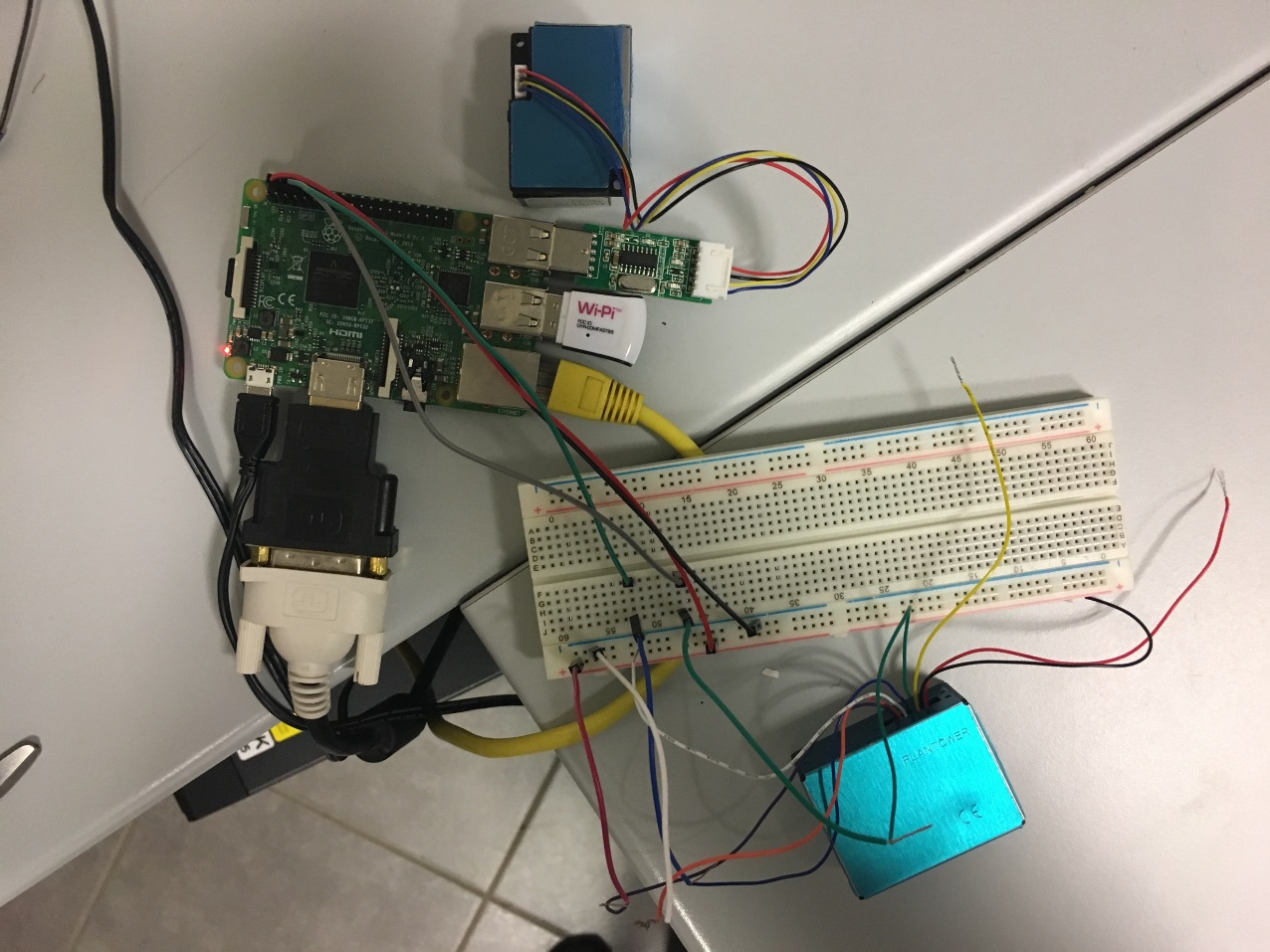
To install MySQLdb python library, type in the following command:

sudo apt-get install python-mysqldb

**2.6. Complete Setup**

Raspberry Pi 3

PMS5003 PM Sensor

****

WiPi

SDS021 PM Sensor

Figure 12. Overall Connection of the Station

**3. DATABASE**

Well organized data is important for any kind of future usage. The database for storing data from the air quality measuring station currently has three tables: (1) log, (2) AQmeasurement, and (3) sensors. The database was designed with consideration that when an individual would like to fetch data associated with their phone’s MAC address, the database searches for the data and returns it efficiently.

**3.1. Credentials**

DATABASE HOST NAME: "agTechELlqs.ag.technion.ac.il"

USER NAME: "AirSeung"

PASSWORD: "AirSeung"

DATABASE: "AirSeung"

One can log into the database by typing in following command in the Raspberry Pi terminal:

mysql -h agTechELlqs.ag.technion.ac.il -u AirSeung -p

where

mysql -h (hostname) -u (user name) -p

**3.2. Requirements**

The database needs to store:

(1) MAC addresses of various Wi-Fi enabled devices that were detected at different times

(2) Air quality sensor data [average PM 2.5 concentration, average PM 10 concentration, standard deviation PM 2.5 concentration, standard deviation PM 10 concentration, minimum PM 2.5 concentration, minimum PM 10 concentration, maximum PM 2.5 concentration, maximum PM 10 concentration]

(3) Date and time of data acquisition

(4) Hardware ID that identifies the individual air quality station

(5) Sensor ID that identifies an individual sensor

(6) The name of the sensor

(7) The air quality station type (Does it measure gas? PM?)

(8) The air quality station location, longitude, latitude

**3.3. Tables**

**3.3.1. Log Table**

Table 2. Columns of the log table

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Extra** |
| log\_id | Table id | int | primary key, auto increment |
| date | Date and time of data acquisition | datetime |  |
| mac\_address | Mac address | text |  |
| hardware\_id | Hardware ID | int |  |

Table 3. An example of the log table

|  |  |  |  |
| --- | --- | --- | --- |
| **log\_id** | **date** | **mac\_address** | **hardware\_id** |
| 1 | 2017-07-28 | 00:0b:0e:2b:7b:c2 | 2 |
| 2 | 2017-07-29 | 01:00:5e:00:00:16 | 1 |
| 3 | 2017-07-29 | c4:8e:8f:f4:95:4d | 2 |
| … | … | … | … |

**3.3.2. AQMeasurement Table**

Table 4. Columns of the AQmeasurement table

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Extra** |
| date | Date and time of data acquisition | datetime | primary key |
| hardware\_id | Hardware ID | int | primary key |
| avg\_pm25 | Average PM 2.5 concentration | float |  |
| avg\_pm10 | Average PM 10 concentration | float |  |
| std\_pm25 | Standard deviation of PM 2.5 concentration | float |  |
| std\_pm10 | Standard deviation of PM 10 concentration | float |  |
| min\_pm25 | Minimum PM 2.5 concentration | float |  |
| max\_pm25 | Minimum PM 10 concentration | float |  |
| min\_pm10 | Maximum PM 2.5 concentration | float |  |
| max\_pm10 | Maximum PM 10 concentration | float |  |

Table 5. An example of the AQmeasurement table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **date** | **hardware\_id** | **avg\_pm25** | **avg\_pm10** | **std\_pm25** | **std\_pm10** | **min\_pm25** | **max\_pm25** | **min\_pm10** | **max\_pm10** |
| 2017-07-28 | 1 | 5.446 | 6.001 | 0.600 | 0.954 | 4 | 6 | 3 | 8 |
| 2017-07-29 | 2 | 4.639 | 5.892 | 0.801 | 0.697 | 3 | 6 | 5 | 7 |
| 2017-07-29 | 1 | 5.213 | 4.522 | 0.702 | 0.445 | 5 | 6 | 5 | 8 |
| … | … | … | … | … | … | … | … | … | … |

**3.3.3. Sensor Table**

Unlike the previous tables, Sensors table is managed manually. Whenever there is a new air quality station built, its information is entered in this table. It has both Hardware ID and Sensor ID for each station in case that a new station is built with a same sensor from broken.

Table 6. Columns of the Sensor table

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Type** | **Extra** |
| hardware\_id | Hardware ID | int | primary key, auto increment |
| sensor\_id | Sensor ID | text |  |
| sensor\_name | Sensor Name | text |  |
| location | Location of the station | text |  |
| type | Type of the pollutant that the station measures | text |  |
| longitude | Longitude of the location | float |  |
| latitude | Latitude of the location | float |  |

Table 7. An example of the Sensor table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **hardware\_id** | **sensor\_id** | **sensor\_name** | **location** | **type** | **longitude** | **latitude** |
| 1 | 00000000 | SDS021 | bar\_lab | pm |  |  |
| 2 | 00000001 | PMS5003 | bar\_lab | pm |  |  |
| … | … | … | … | … | … | … |

Log table is the middle man that stands between AQmeasurement and sensors tables connects all three tables. The idea is, when some individual queries for the air quality data associated with their phone’s MAC address, the log table gives a list of relevant records that each contains **hardware\_id** and **date**. Hardware\_id and date are used to fetch the air-quality data from AQmeasurement table. Hardware\_id itself is used to see which sensor and where the data was collected from by looking at sensors table.

**4. CODE**

A python code was written to utilize the components and establish a flow of data connected to the database from the air quality measuring station.

**4.1. Developing Environment**

Raspbian Operating System has a pre-installed python integrated development environment. One can use this like a text-editor for python programming. The code can be run by typing in the following command in the terminal:

sudo python [directory where the code is located]

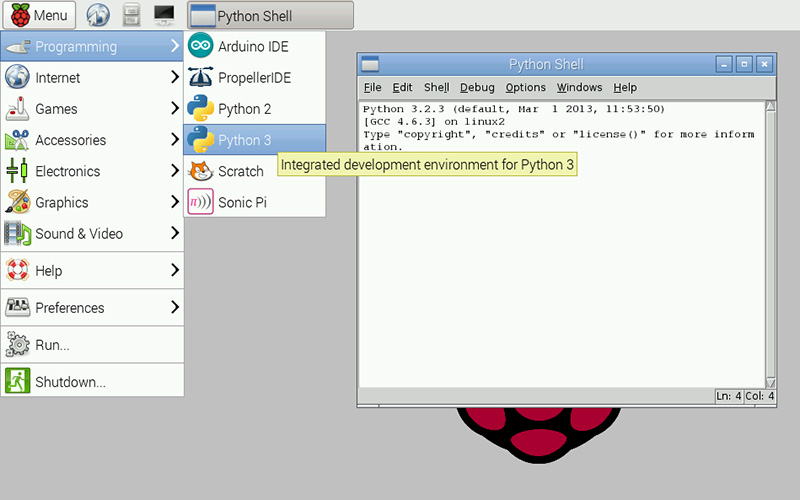


Figure 13. Python IDE on Raspbian

**4.2. Sensor Reader Class**

**4.2.1. SDS021 Sensor Reader**

#---------------------------------#

# filename: sensorSDS021.py       #

# author: SeungDoo (Charlie) Yang #

#---------------------------------#

import os

import sys

import time

import serial

import numpy as np

class SDS021Reader:

    def \_\_init\_\_(self, inport):

        #open port

        self.serial = serial.Serial(port=inport,baudrate=9600)

    def readValue(self):

        """function:

            Read and return a frame with length of 8 from the serial port

        """

        step = 0

        while True:

            while self.serial.inWaiting() != 0:

                #convert serial input of ASCII characters as an integer

                v = ord(self.serial.read())

                #total of 10 bit but only focus on 3rd to 6th DATA bits

                if step == 0:

                    #first bit is always 170

                    if v == 170:

                        values = [0,0,0,0,0,0,0]

                        step = 1

                elif step == 1:

                    #second bit is always 192

                    if v == 192:

                        step = 2

                    else:

                        step = 0

                elif step > 8:

                    step = 0

                    #low and high\*265 byte / 10... according to documentation

                    pm25 = (values[0] + values[1]\*256)/10

                    pm10 = (values[2] + values[3]\*256)/10

                    return [pm25,pm10]

                #start DATA bit acquisition

                elif step >= 2:

                    if v == 170: #on 4th measurement 170, 192 repeats

                        step = 1

                        continue

                    values[step-2] = v

                    step += 1

    def read(self, duration, file\_name, debug):

        """function:

Read the frames for a given duration and return PM 2.5 and PM 10 concentrations' average, standard deviation, min, and max

        """

        #initialization

        start = os.times()[4]

        count = 0

        species = [[],[]]

        result = []

        while os.times()[4]<start+duration:

            try:

                values = self.readValue()

                #pm2.5

                species[0].append(values[0])

                #pm10

                species[1].append(values[1])

                #elasped time

                dt = os.times()[4] - start

                time.sleep(1)

                count += 1

            except KeyboardInterrupt:

                sys.exit()

            except:

                e = sys.exc\_info()[0]

        #create debug file

        if debug:

            file\_name = "debug\_" + file\_name

            f = open(file\_name,"w")

            for pm in range(2):

                for value in species[pm]:

                    f.write(str(value) + "\n")

                f.write("-------------\n")

            f.close()

        #average, standard deviation, min and max computing

        for i in range(len(species)):

            result.append(np.average(species[i]))

            result.append(np.std(species[i]))

            result.append(min(species[i]))

            result.append(max(species[i]))

        return result

**4.2.2. PMS5003 Sensor Reader**

#---------------------------------#

# filename: sensorPMS5003.py      #

# author: SeungDoo (Charlie) Yang #

#---------------------------------#

import os

import sys

import time

import serial

import numpy as np

DATA\_FRAME\_LENGTH = 28

class PMS5003Reader:

    def \_\_init\_\_(self, inport):

        #open port

        self.serial = serial.Serial(port=inport,baudrate=9600)

    def readValue(self):

        """function:

            Read and return a frame with length of 28 from the serial port

        """

        while True:

            data = []

            #two start bytes

            start1 = self.serial.read()

            if start1 == b'\x42':

                start2 = self.serial.read()

                if start2 == b'\x4d':

                    #check if the frame length is correct

                    frame\_high = ord(self.serial.read())

                    frame\_low =  ord(self.serial.read())

                    frame\_length = frame\_high\*256 + frame\_low

                    if frame\_length == DATA\_FRAME\_LENGTH:

                        #data acquisition

                        for i in range(frame\_length):

                            data.append(ord(self.serial.read()))

                        #verify the data with checksum (last two bytes)

                        checksum\_expected = data[len(data)-2]\*256 + data[len(data)-1]

                        checksum\_received = 0

                        for i in range(len(data) - 2):

                            checksum\_received += data[i]

                        checksum\_received += frame\_high + frame\_low + ord(start1) + ord(start2)

                        if checksum\_expected == checksum\_received:

                            return data

                        else:

                            print "checksum error\n"

                            print "expected: {0}, received: {1} bytes".format(checksum\_expected, checksum\_received)

                            return None

                    else:

                        print "unexpected frame length"

                        return None

    def read(self, duration, file\_name, debug):

        """function:

Read the frames for a given duration and return PM 2.5 and PM 10 concentrations' average, standard deviation, min, and max

The frame also consists of 10 other data, such as a number of particles larger than 2.5 μm. But at this moment, only focusing on PM 2.5 and PM 10 concentrations

        """

        #initialization

        start = os.times()[4]

        count = 0

        species = [[] for i in range(12)]

        result = []

        while os.times()[4]<start+duration:

            try:

                values = self.readPacket()

                #convert each data to an integer value

                for i in range(len(species)):

                    species[i].append(values[i\*2]\*256 + values[i\*2 + 1])

                #elasped time

                dt = os.times()[4] - start

                time.sleep(1)

                count += 1

            except KeyboardInterrupt:

                print "keyboard interrupt"

                sys.exit()

            except:

                e = sys.exc\_info()[0]

                print ("ERROR: " + str(e))

        #create debug file

        if debug:

            file\_name = "debug\_" + file\_name

            f = open(file\_name,"w")

            for pm in range(len(species)):

                for value in species[pm]:

                    f.write(str(value) + "\n")

                f.write("-------------\n")

            f.close()

        #average, standard deviation, min and max computing

        for i in range(len(species)):

            result.append(np.average(species[i]))

            result.append(np.std(species[i]))

            result.append(min(species[i]))

            result.append(max(species[i]))

        #only return concentration of pm2.5 and pm10 (for the current purpose)

        return result[12:20]

**4.3. Functions**

**4.3.1. Air Quality Measurement**

#in file: AQstation\_mainCode.py

def doMeasurement(duration, debug):

    """function:

        This function calls one of the two sensor

        reader classes according to which station

        was selected. It receives average, max and min values

        of the PM 2.5 and PM 10 concentration after

        a measurement of a set duration of time.

        Then it creates a new log file and stores the data.

        It return the file name of the newly created log file.

    """

    print "START MEASUREMENT"

    count = 0

    #open a log file for current time

    file\_name = str(datetime.datetime.now()).split(".")[0]

    #measurements

    if SELECTED\_HARDWARE == 1:

        USBPORT  = "/dev/ttyUSB0"

        results = sensorSDS021.SDS021Reader(USBPORT).read(duration, file\_name, debug)

    elif SELECTED\_HARDWARE == 2:

        USBPORT  = "/dev/ttyS0"

        results = sensorPMS5003.PMS5003Reader(USBPORT).read(duration, file\_name, debug)

    file\_name = "log\_" + file\_name

    f = open(file\_name,"w")

    #if nothing is measured

    if len(results) == 0:

        for i in range(8):

            f.write("-1\n")

    #append results at the end of the log file

    else:

        for i in range(8):

            f.write(str(results[i]) + "\n")

    f.write("---END OF MEASUREMENTS---\n")

    f.close()

    return file\_name

**4.3.2. Detecting Nearby Devices**

#in file: AQstation\_mainCode.py

def detectDevices(file\_name, duration):

    """function:

        This function detects nearby Wi-Fi enabled devices

        and saves their MAC address in the log file from

        doMeasurement function.

    """

    print "START DETECTION"

    #initialization

    splited\_line = []

    mac\_list = []

    start\_append = 0

    #append mode

    f = open(file\_name,"a")

    #command line configuration

    #only display source, and destinaiton of the packet

    cmd = ("tshark -i mon0 -o column.format:"+'"src","%uhs","dst","%uhd"').split()

    #execute the command

    process = subprocess.Popen(cmd, stdout = subprocess.PIPE, stderr = subprocess.STDOUT)

    start\_time = time.time()

    #for every output line

    for line in iter(process.stdout.readline, ""):

        #acquire data for set duration of time

        if time.time()-start\_time > duration:

            process.terminate()

            break

        #start writing to the file when a MAC address is read

        if start\_append == 0:

            if "Capturing on" in str(line):

                start\_append = 1

                continue

        splited\_line = line.split(" ")

        for mac in splited\_line:

            if "\n" not in mac:

                mac = mac + "\n"

            #check if it is a valid MAC address

            if ("ff:ff:ff:ff:ff:ff" not in mac) and (len(mac) == 18):

                #write the address to file if it was not detected before

                if mac not in mac\_list:

                    mac\_list.append(mac)

                    f.write(mac\_list[len(mac\_list)-1])

    f.write("---END OF MAC ADDRESSES---")

    f.close()

    return

**4.3.3. Uploading Data to the Database**

#in file: AQstation\_mainCode.py

def uploadToDatabase(file\_name, selected\_hardware, host, user, password, database):

    """function:

        This function uploads the data in the log file to the database.

        It only uploads the data if the log file is complete.

        After uploading, it moves the log file to "uploaded\_logs" folder

        to indicated that the log file has been uploaded.

    """

    print "START UPLOADING"

    #upload to AQMeasurement if 0, to log if 1

    switch\_table = 0

    results = []

    #open data base connection

    db = mysql.connect(host, user, password, database)

    #prepare cursor object

    curs = db.cursor()

    f = open(file\_name,"r")

    lines = f.readlines()

    #if a log file is incomplete

    if "---END OF MAC ADDRESSES---" not in lines:

        incomplete\_file = "'" + file\_name + "'"

        cmd = "sudo rm /home/pi/logs/{0}/{1}".format(str(selected\_hardware), incomplete\_file)

        os.system(cmd)

        f.close()

        return

    for line in lines:

        #end of file

        if line == "---END OF MAC ADDRESSES---":

            continue

        if "\n" in line:

            line = line[:len(line)-1]

        if switch\_table == 0:

            #switch to uploading to AQMeasurement

            if line == "---END OF MEASUREMENTS---":

                switch\_table = 1

                continue

            results.append(line)

        else:

            #upload to log table

            try:

                curs.execute("""insert into log(date, mac\_address, hardware\_id) values(%s, %s, %s)""",

                             (file\_name[4:], line, selected\_hardware)) #file\_name[4:] to take out "log\_"

                db.commit()

            except:

                db.rollback()

    #upload to AQMeasurement table

    try:

        curs.execute("""insert into AQmeasurement(date, hardware\_id, avg\_pm25, avg\_pm10, std\_pm25, std\_pm10, min\_pm25, max\_pm25, min\_pm10, max\_pm10) values(%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)""",

                     (file\_name[4:], selected\_hardware, results[0], results[4], results[1], results[5], results[2], results[3], results[6], results[7]))

        db.commit()

    except:

        db.rollback()

    #move the file to uploaded logs

    file\_name = "'" + file\_name + "'"

    cmd = "sudo mv {0} /home/pi/logs/{1}/uploaded\_logs/".format(file\_name, str(SELECTED\_HARDWARE))

    os.system(cmd)

    print "END"

    f.close()

    db.close()

    return

**4.3.4. Checking Internet Connection**

#in file: AQstation\_mainCode.py

def checkInternetConnection(host):

    """function:

        This function checks if there is an internet connection

        to the database and uploading is possible. It returns True if yes.

    """

    count = 0

    #check if pinging goes through

    cmd = ("timeout 10 ping {0}".format(host)).split()

    process = subprocess.Popen(cmd, stdout = subprocess.PIPE, stderr = subprocess.STDOUT)

    for line in iter(process.stdout.readline, ""):

        count += 1

    #there is only one line of output if database cannot be reached

    if count > 1:

        process.terminate()

        return True

    process.terminate()

    return False

**4.4. Top Level Code**

**4.4.1. Initialization**

#---------------------------------#

# filename: AQstation\_mainCode.py #

# author: SeungDoo (Charlie) Yang #

#---------------------------------#

import subprocess

import datetime

import os

import sys

import time

import MySQLdb as mysql

import sensorSDS021

import sensorPMS5003

DURATION = 300 #5 min

DEBUG = False #True to also create a debug file

#credentials

HOST = "agTechELlqs.ag.technion.ac.il"

USER = "AirSeung"

PASSWORD = "AirSeung"

DATABASE = "AirSeung"

#hardware id

SELECTED\_HARDWARE = 2 #1 for SDS021, 2 for PMS5003

**4.4.2. Main Code**

#in file: AQstation\_mainCode.py

#wait for pi to boot up

time.sleep(60)

#check if there is a log folder; if not, create a new one

cmd = "/home/pi"

files = os.listdir(cmd)

if "logs" not in files:

    cmd = "sudo mkdir logs"

    os.system(cmd)

#check if there is a folder for a selected hardware; if not, create a new one

cmd = "/home/pi/logs"

files = os.listdir(cmd)

if str(SELECTED\_HARDWARE) not in files:

    cmd = "sudo mkdir -p logs/{0}/uploaded\_logs".format(str(SELECTED\_HARDWARE))

    os.system(cmd)

#set working directory

cmd = "/home/pi/logs/{0}".format(str(SELECTED\_HARDWARE))

os.chdir(cmd)

while True:

    #upload any not uploaded log

    if checkInternetConnection(HOST):

        cmd = "/home/pi/logs/{0}".format(str(SELECTED\_HARDWARE))

        files = os.listdir(cmd)

        for file in files:

            if "log\_" in file:

                try: uploadToDatabase(file, SELECTED\_HARDWARE, HOST, USER, PASSWORD, DATABASE)

                except: continue

    while True:

        #measurement

        upload\_info = doMeasurement(DURATION, DEBUG)

        #device detection

        detectDevices(upload\_info, DURATION)

        #time interval

        time.sleep(1000)

        if checkInternetConnection(HOST):

            break

**4.5. Automatically Running after Boot**

To automatically activate the air quality measuring station at the boot, this line of command line was added to the rc.local file before the last line of the file, “exit 0.”

sudo python /home/pi/AQstation\_mainCode.py &

“&” in the end runs the code in the background.

**5. References**

[1]Raspberry Pi. (2017). *Raspberry Pi 3 Model B - Raspberry Pi*. [online] Available at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed 30 Jul. 2017].

[2] Raspberrypi.stackexchange.com. (2017). *Help to build Mini PCI-E add-on card for Raspberry Pi 2*. [online] Available at: https://raspberrypi.stackexchange.com/questions/47088/help-to-build-mini-pci-e-add-on-card-for-raspberry-pi-2 [Accessed 1 Aug. 2017].

[3] Rigacci.org. (2017). *Rapsberry Pi Air Quality Station [rigacci.org]*. [online] Available at: http://www.rigacci.org/wiki/doku.php/doc/appunti/hardware/raspberrypi\_air [Accessed 1 Aug. 2017].

[4] Newark.com. (2017). *WIPI - ELEMENT14 - WLAN USB Module, IEEE 802.11n, Connect Raspberry Pi to WiFi Network, Built in Smart Antenna | Newark element14*. [online] Available at: http://www.newark.com/element14/wipi/wlan-module-for-the-raspberry/dp/07W8938 [Accessed 1 Aug. 2017].

**6. Appendix**

A. SDS021Particulate Matter Sensor Data Sheet

B. PMS5003 Particulate Matter Sensor Data Sheet