

# **Final Document**

# **Automated Beehive – PlanBee**

## **Team:**

Abhinandan Shetty Rathnakar  
Chaitanyananda Buruganahalli Sreedhar  
Dilip Maharjan  
Nilesh Solanki  
Shrikanth Singh Balaji Singh

# Contents

- I. Introduction
- II. Hardware and Software Requirements
- III. System Architecture
  - 1. Block diagram
  - 2. Circuit connection diagram
- IV. Sensor Implementation
  - 1. MQ-135 (Gas sensor)
  - 2. BME 280 (Outside Temperature, Humidity and Air Pressure)
  - 3. Load cells
  - 4. DHT-22 ( Inside Humidity and Temperature)
  - 5. Neo-6M (GPS location)
  - 6. kwmobile (Light sensor)
  - 7. USB microphone (Volume and Frequency)
  - 8. MPU 6050 (Gyroscope and Accelerometer)
  - 9. Night vision camera (Image)
- V. Software Integration
  - 1. Database
  - 2. Scheduling and Logging
  - 3. Adding the main program to boot file
- VI. Installation steps
- VII. Results
- VIII. Challenges and Solutions
- IX. Limitations
- X. Future Scope
- XI. Extended Work
- XII. Conclusion
- References

## **I. Introduction (*Team*)**

This document gives in-depth knowledge about the various sensors' basic functionality, its pin configuration, interface with the Raspberry Pi 3b+, and integration steps for the sensor data with the Firebase cloud database, installation steps and the challenges faced during the prototype implementation for the better understanding of the user. The purpose of the sensors is as per the requirement document.

There are a few changes done on the prototype version from what was mentioned in the requirement analysis document. They are as follows:

1. KY-037 sound sensor has been replaced with the USB mic as the output of initially selected KY-037 sensor output was erroneous and was hard to understand the output generated by the sensor. The calculation of the frequency value estimation was tough to carry out. Hence we changed to USB mic.
2. Gyroscope and Accelerometer sensor – MPU-6050 was not in the scope of the initial requirement document, it is implemented in the prototype version to get the data about the orientation of the beehive.
3. Connection of sensors from bread-board to matrix board to ensure the robustness of the connection.

## II. Hardware and Software Requirements (*Team*)

For our automated beehive project, we are mainly using different sensors to collect data using Raspberry Pi. Following are the list of hardware and software components we used in our project:

### 1. Hardware Requirements

- a. Raspberry Pi 3B+
- b. MQ135 (Gas sensor)
- c. JY-MCU (level shifter)
- d. MCP 3008 (ADC)
- e. DHT 22 (Inside Temperature and Humidity sensor)
- f. Neo 6M (GPS sensor)
- g. MPU 6050 (Acceleration and Orientation)
- h. USB microphone (Volume and Frequency)
- i. Night vision camera ( Image)
- j. BME 280 (Outside Temperature, Humidity and Air Pressure)
- k. Load cells
- l. HX711 (ADC)

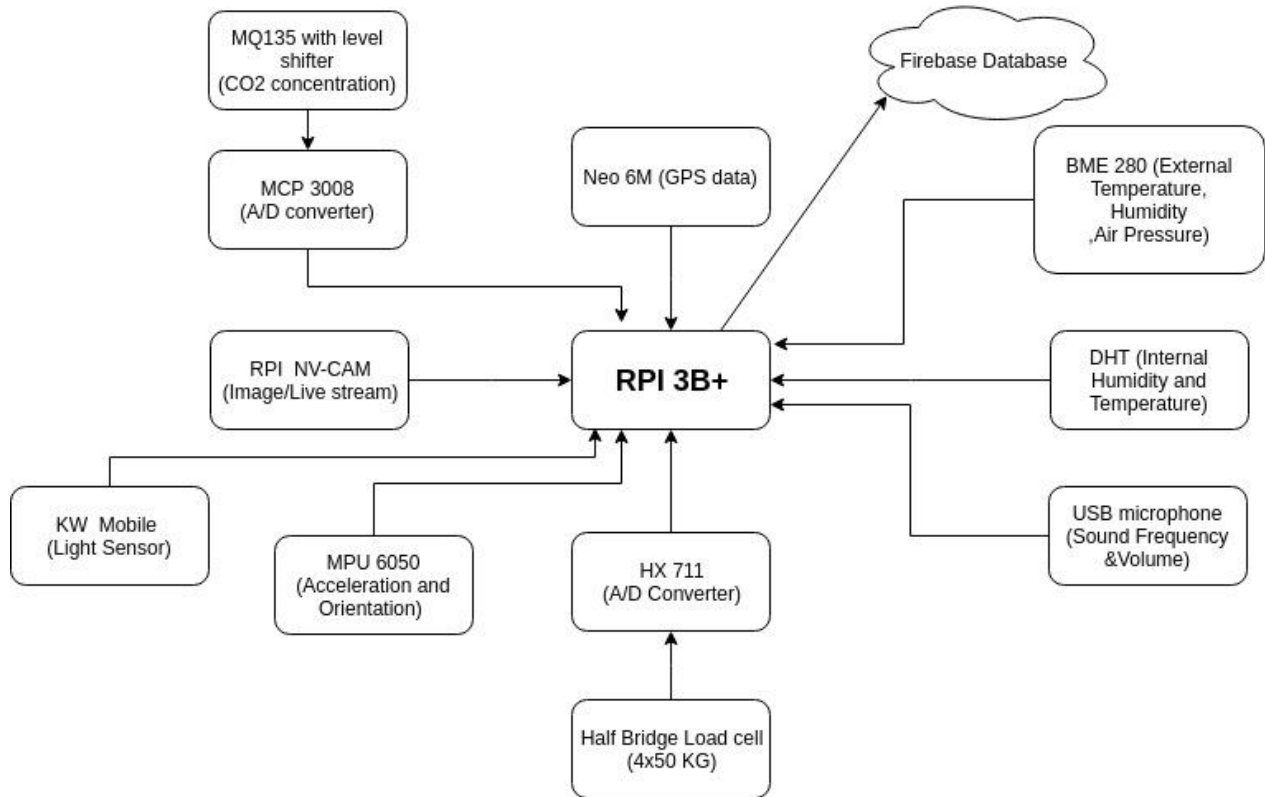
### 2. Software Requirements

- a. Operating System: Raspbian Buster
- b. Fritzing tool to draw schematics of Raspberry Pi connection with all the sensors
- c. Programming Language: Python (Python 3)
- d. Database: Firebase - Cloud Firestore

### III. System Architecture (*Team*)

## 1. Block Diagram

The following diagram shows the interface between the sensors and Raspberry Pi and the transfer of data to the database:



### Fig 1: System Architecture

The type of interface between the sensor and Raspberry Pi is given in the table below:

| Sensor name         | Interface          |
|---------------------|--------------------|
| MQ-135 (Gas sensor) | SPI Interface      |
| BME 280             | I2C Interface      |
| MPU 6050            | I2C Interface      |
| Neo 6M              | UART communication |
| DHT22               | GPIO               |
| Load Cells          | GPIO               |

|                         |                         |
|-------------------------|-------------------------|
| KW mobile               | GPIO                    |
| USB microphone          | USB Interface           |
| RPI Night vision camera | Camera Serial Interface |

## 2. Circuit Connection Diagram

As the major portion of our project relies on the accurate hardware connections, a clear picturization of the pin to pin connections between the various sensors and the Raspberry Pi is required. The circuit diagram is designed using the tool called Fritzing, as it provides us with all the pin diagrams of the sensors used in the project.

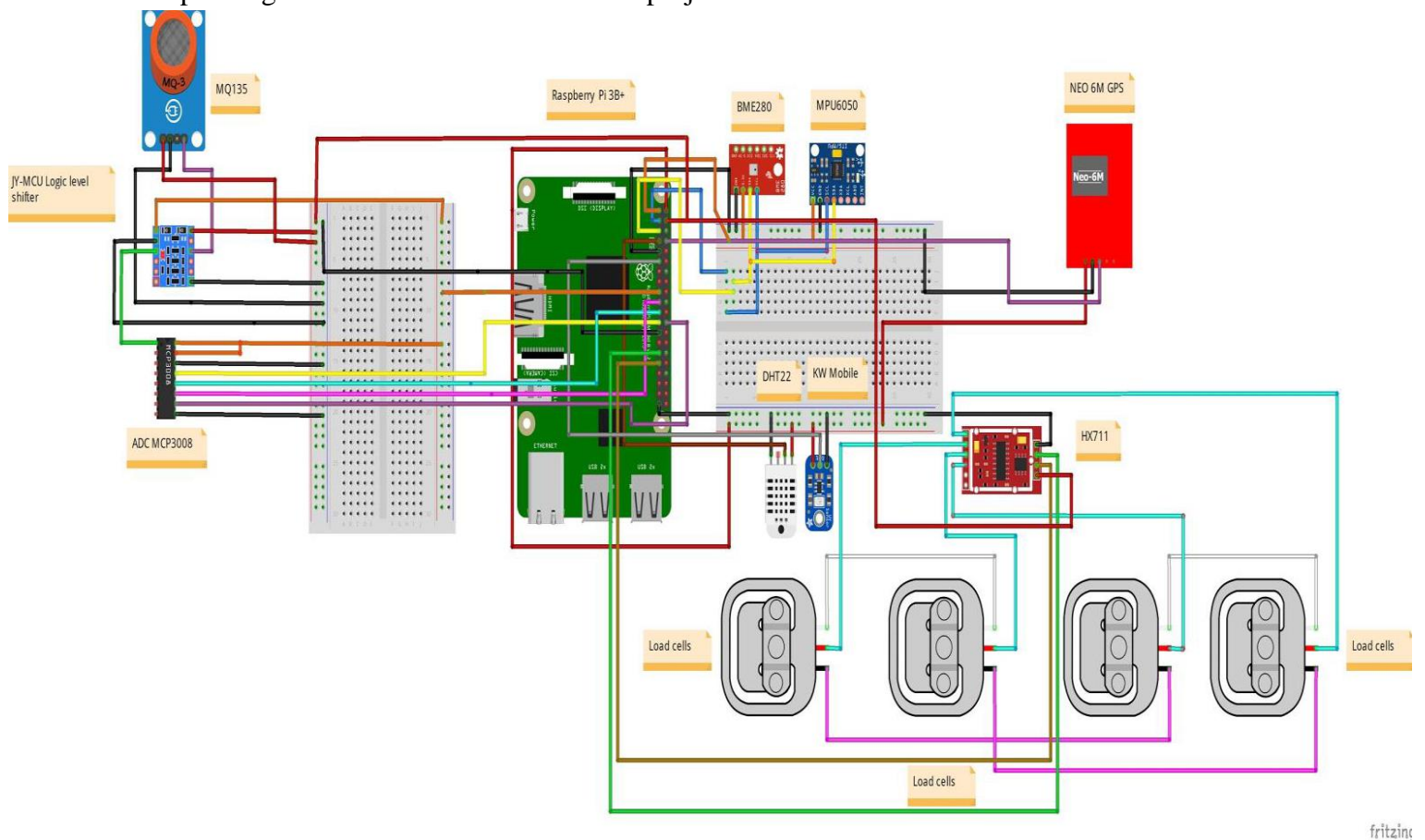


Fig 2: Circuit Connection Diagram

## IV. Sensor Implementation

### 1. Gas Sensor (*Shrikanth Singh Balaji Singh*)

The need for sensing the gaseous concentration inside the beehive is because carbon-di-oxide is the byproduct of the metallization of the honey by bees. It is also indicative of the presence of the queen bee in the beehive and their mating seasons [1]. The sensor which can serve for the measurement of the concentration of carbon-di-oxide is MQ135 called as the air quality sensor. Electrochemical gas sensors measure the concentration of a target gas by oxidizing or reducing it at an electrode and then computing the resulting current. MQ135 is a stable, low electrochemical gas sensor which uses  $\text{SnO}_2$  as its sensitive material. The datasheet claims that the MQ135 is suitable for  $\text{CO}_2$ , Alcohol, Benzene,  $\text{NO}_x$ ,  $\text{NH}_3$  [2].

Since the purpose of the gas sensor in this project is to sense the content of carbon-di-oxide and not it's accurate ppm value, a primitive sensor usually used for sensing the smoke and fire alarm systems is economical. Hence the ppm value has to be analyzed from the recordings on the field. After the analysis, a range of ppm values can be fixed in order to determine the status of the beehive. The sensor MQ135 can be identified by looking at the image below.

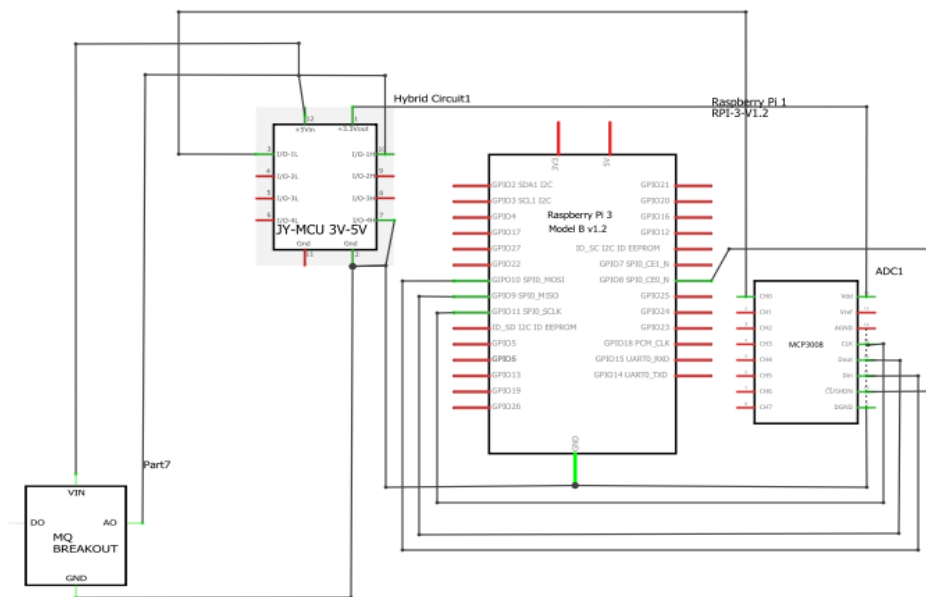


Fig 3: Gas sensor MQ135 [Source: Google Images]

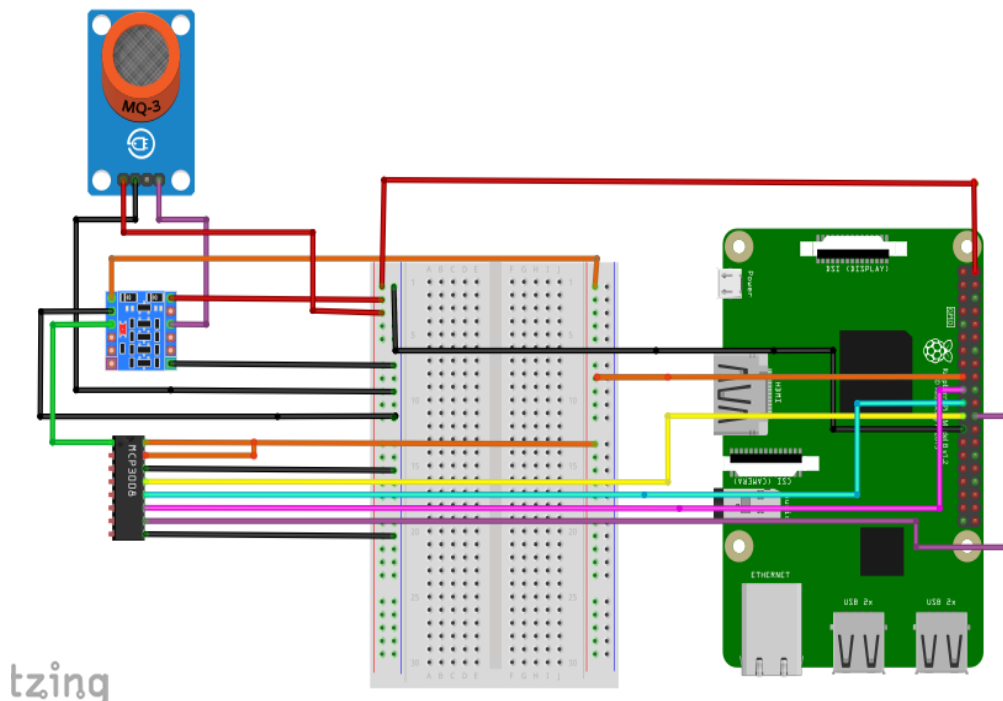
#### a. Pin Configuration

| Pin No<br>MQ135 | Pin<br>Name    | Pin Description   |
|-----------------|----------------|---|
| 1               | VCC            | Used to power the sensor, generally the operating voltage is +5V.                       |
| 2               | Ground         | Used to connect the module to the system ground.  |
| 3               | Digital<br>Out | Get digital output from this pin, by setting a threshold value using the potentiometer. |
| 4               | Analog<br>Out  | This pin outputs 0-5V analog voltage based on the intensity of the gas.                 |

### **b. Schematic representation and Interfacing connection of MQ135 with Raspberry Pi**



**Fig 4: Schematic Diagram of gas sensor module MQ135**



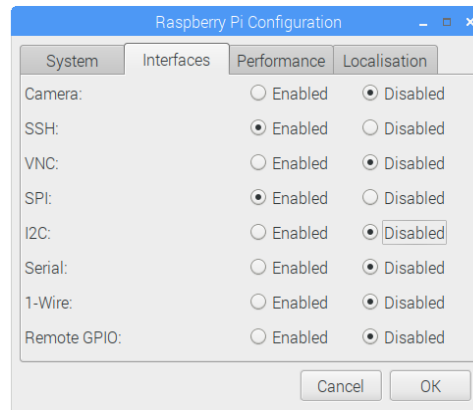
**Fig 5: Interfacing connection of gas sensor module MQ135**

### c. Interface configuration

The sensor MQ135 uses the SPI interface to read out the analog signals from the pin 4 as mentioned above. The first and foremost step to make the sensor work is, enabling the SPI interface



in the configuration settings of the Raspberry Pi. The screenshot below shows how to enable the interfaces.



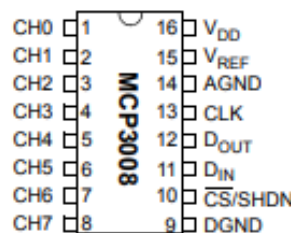
#### d. Challenges Encountered

- i. The gas sensor in itself is a complex sensor the reason behind it is that we require the analog measurements in terms of ppm rather than digital outputs. On the other hand, the Raspberry Pi works only on digital signals so this creates a necessity of analog to digital conversion.

Technical details of MCP3008

- The Microchip Technology Inc. MCP3008 devices are successive approximation 10-bit Analog-To-Digital (A/D) converters with on-board sample and hold circuitry.
- The MCP3008 is programmable to provide four pseudo-differential input pairs or eight single-ended inputs.
- The devices are capable of conversion rates of up to 200 ksps. The MCP3004/3008 devices operate over a broad voltage range (2.7V - 5.5V).
- Communication with the devices is accomplished using a simple serial interface compatible with the SPI protocol.
- The MCP3008 is offered in 16- pin PDIP and SOIC packages.

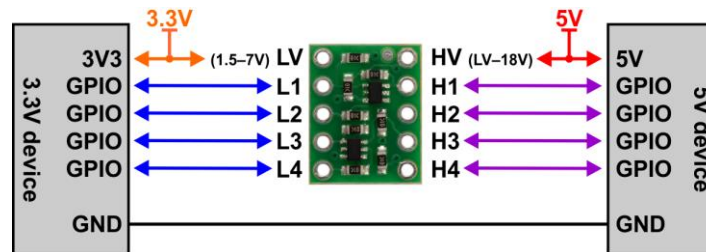
A basic pin diagram is shown below (see Fig 6). The same chip can be identified as the Black coloured one in the breadboard diagram(see Fig 5)[3].



**Fig 6: MCP3008 IC [3]**

- ii. We noticed that the output of the gas sensor is 5V whereas the SPI pins on raspberry pi can handle voltage levels only up to 3.3V. Hence a step down of the voltage is required and this

can be achieved by using JY-MCU 3V 5V logic level shifter. A short working description is given below in the form of a diagram (see Fig 7). The same chip can be identified as a blue coloured one in the breadboard connection diagram (see Fig 5).



**Fig 7: Breadboard connections of gas sensor module MQ135 [4]**

### e. Things to remember

- i. Check if the level shifter 3V and 5V are properly excited from respective sources and ensure that it is not interchanged.
- ii. MOSI and MISO connections should be given precisely.
- iii. The sensor should be preheated before installing onto the field.
- iv. The sensitivity of the sensor can be adjusted by rotating the potentiometer knob.
- v. SPI interface should be enabled in the raspberry pi configuration settings.

### f. Limitation

Though MQ-135 is an appropriate sensor for our application in this project, it possesses a few drawbacks which have to be compromised [2]:

- i. The MQ135 gas sensor detects a number of gases like ammonia, CO<sub>2</sub>, SO<sub>2</sub> etc. collectively but is unable to identify the individual gas concentration in a polluted environment.
- ii. The sensor also uses an inbuilt heater to warm up the air near the sensitive part for oxidation or reduction to take place
- iii. It has been advised not to use this with a small battery source as it will quickly drain your battery.
- iv. Sensor requires 24-48 hours of warm-up time to start emitting stable readings of gas concentration.

After successful implementation of the hardware connections, a sample results window from the firebase database has been shown below, where it can be noticed that ppm values of respective gases can be viewed.

```
C02 : 0.006983812364339607
Methane: 0.009805533222498577
Smoke : 0.02660606308723068
```

**Fig 8: Sample Output of Gas Sensor**

## 2. Temperature, Pressure and Humidity Sensor - BME 280

(Shrikanth Singh Balaji Singh)

The measurement of atmospheric temperature, humidity and pressure are essential to monitoring the environmental conditions surrounding the beehive. Depending on the temperature and humidity outside the beehive we can estimate the efficiency of labour mechanism of the bees. Because there are a group of bees called “Worker bees” whose function is to maintain the temperature inside the beehive within an optimum range of 32-36 deg Celsius even though the temperature outside is freezing cold [5]. On the other hand if the temperature falls below 12 deg C, bees cannot fly and they will be confined to the hive and they cannot crawl from the edge of their warm cluster to the honey they will starve or freeze.

The appropriate sensor which would benefit us in measuring all the three parameters together is BME 280. The module comes with an on-board LM6206 3.3V regulator and I2C Voltage-Level Translator, so you can use it with a 3.3V or 5V logic. The sensor provides both SPI and I2C interfaces and can be supplied using 1.71 to 3.6 V for the sensor supply VDD and 1.2 to 3.6 V for the interface supply. The sensor BME280 is shown below (see Fig 9) which will help you to find out the sensor in the hardware module as well as in the circuit diagram. The same chip can be identified as the red coloured one in the breadboard connection diagram (see Fig 2 and 11).

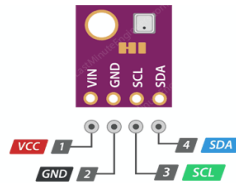
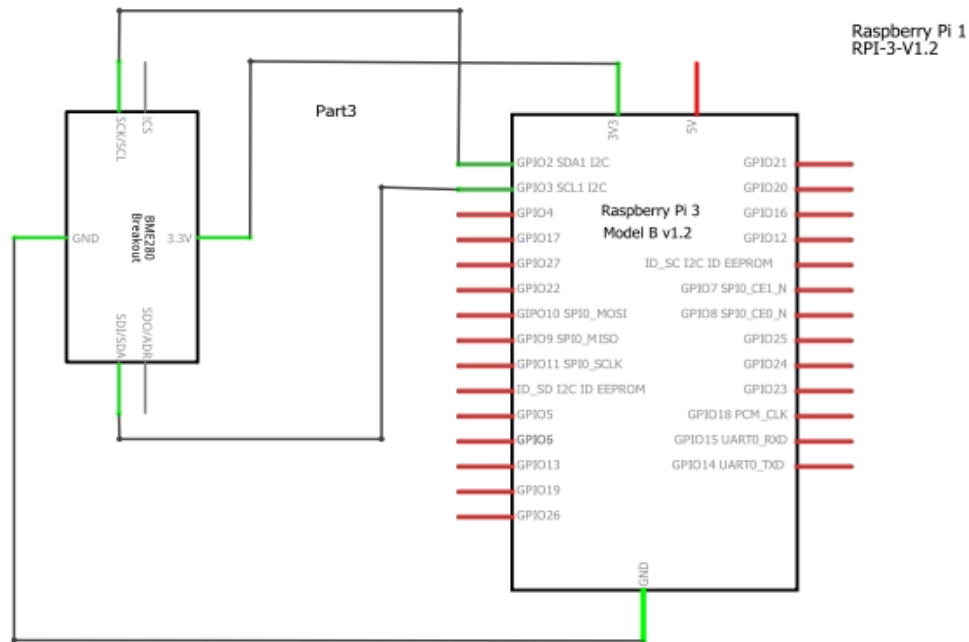


Fig 9: Environment Sensor BME280 [6]

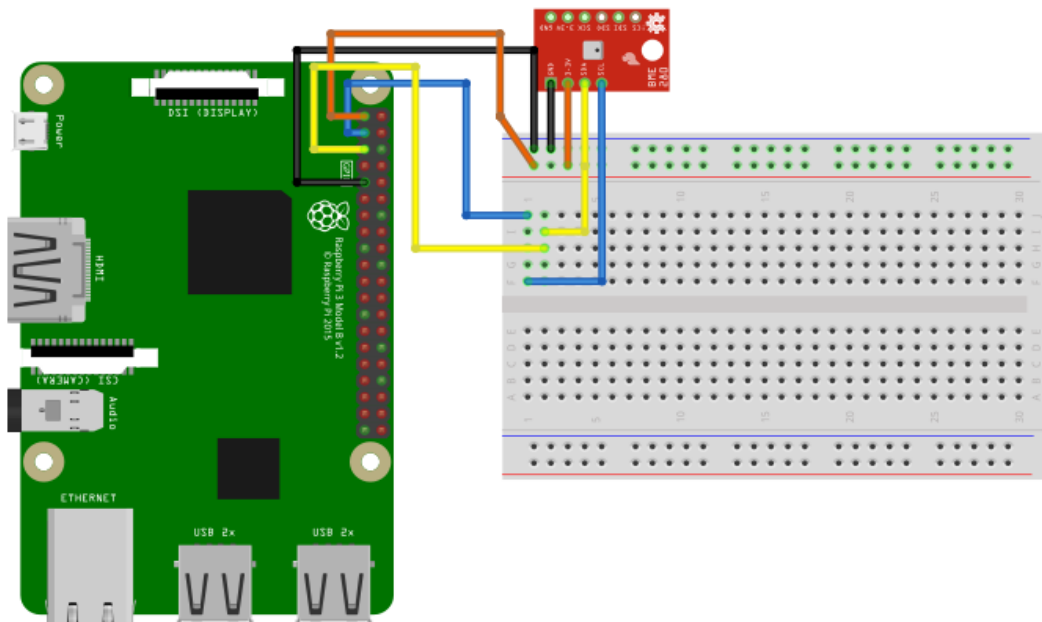
### a. Pin Configuration of BME280

| Pin.No<br>BME280 | Pin<br>Name | Description  | Raspberry Pi Pins |
|------------------|-------------|--|-------------------|
| 1                | VCC         | Used to power the sensor, generally the operating voltage is +3.3V.                | Pin No. 1         |
| 2                | Ground      | Used to connect the module to system ground.                                       | Pin No. 20        |
| 3                | SCL         | Serial clock line - Maintains synchronization between the master and slave devices | Pin No. 5         |
| 4                | SDA         | Serial data line - Real data transfer between the master and slave devices.        | Pin No. 3         |

## b. Schematic Representation



**Fig 10: Schematic diagram of environmental sensor BME280**



**Fig 11: Breadboard connections of environmental sensor BME280**

### c. Interface configuration

Though BME280 has a facility of both protocols i.e I2C and SPI, we have chosen I2C because it offers multiple master-slave configurations. This feature will be useful when we implement gyroscope and acceleration sensor which also makes use of I2C for communicating with raspberry pi. Similar to gas sensor the I2C interface has to be enabled in the configuration settings of the raspberry pi. Follow the steps as shown in the figure under the gas sensor.


#### Installing I2C tools

**Step i:** The sensor has to be tested for port detection, unless and until the communication port is detected BME280 cannot communicate with raspberry pi. Before testing the port one has to install i2c tools using the following command:

**`sudo apt-get install -y i2c-tools`**

**Step ii:** Now in order to check for port detection one has to execute the following command:

**`i2cdetect -y 1`**



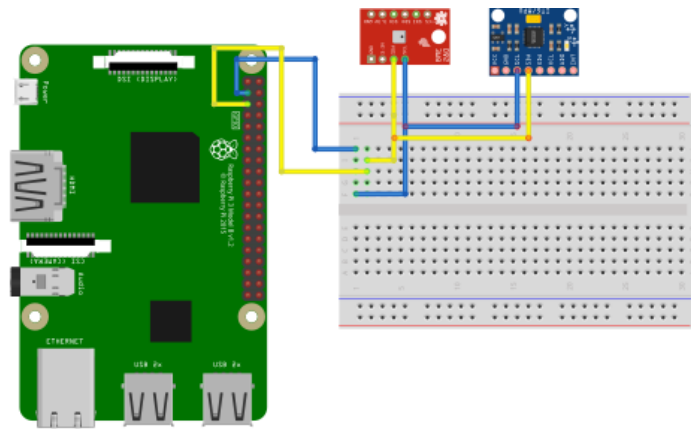
```
pi@raspberrypi ~ $ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70:  -- -- -- -- -- 76 --
```

**Step iii:** If the port is detected then the output will be as shown below. 0x76 is the default address of the sensor but there are chances few manufacturers designs it with 0x77 which can be identified with the command mentioned above. After detecting the right address of the sensor, the same address has to be used in the program since the communication with Raspberry Pi will take place via this address.

### d. Challenges Encountered

- i. The sensor address detection fails unless the proper connection is ensured between the i2c pins of the sensor and the raspberry pi. Even a hint of loose connection will yield null output when the address detect command is executed. The only solution to overcome this issue is to solder the connections between the sensor and the Raspberry Pi.
- ii. The raspberry pi offers only one set of I2C pins (SDA, SCL) which is already used by BME280. In the upcoming sections, we have incorporated gyroscope and accelerometer sensor (MPU6050) which also requires I2C pins to communicate with raspberry pi. The solution for this is to create a bus for SDA and SCL connection from raspberry pi and plug in the SDA and SCL connections of the respective sensors in our case BME280 and MPU6050 in parallel. By doing this one can connect as many I2C devices in parallel with

the same raspberry pi. The connection for the parallelizing of I2C pins is shown below (red coloured one BME280 and blue coloured one MPU6050).



**Fig 12: Paralleling of I2C pins**

#### **d. Things to remember**

The most important point to remember is to check for the address of the sensor before executing the program.

- i. The communication port of 0x76 or 0x77 should be updated in the program without fail.
- ii. The sensor's physical dimension is pretty small so one must take immense care while soldering the pins for connecting with raspberry pi.

Once all the connections are verified and the interface configuration is done we can dump the program into the raspberry pi. The sensor will output the temperature in deg C, humidity in percentage and the pressure in hPa. Sample output is shown below which is from the cloud storage of the firebase database. The efficiency of the sensor was tested with the Google weather data and the accuracy is up to +3 to -3 deg C which is quite acceptable.

```
Outside_humidity: 46.66185071008953
Outside_pressure: 978.5834300559609
Outside_temp: 26.414107622025767
timestamp: "20190809_124600"
```

**Fig 13: Sample Output of BME280**

### 3. Load Cells (*Nilesh Solanki*)

We are measuring the weight of the beehive for the automated beehive because it gives the beekeeper the indication if the food consumption is high and whether there is a need for feeding. It also gives an idea about how much honey is inside the beehive. Here in this project, we are using a strain gauge weight sensor (see Fig 14). Strain gauge is a sensor whose resistance varies with the change of applied force or weight so by measuring the change in resistance, we can measure the weight of beehive. The range of one sensor is 50kg so we are using four sensors to form a full-bridge measurement, which increases the measuring range to 200kg. Also with the sensors, we are using HX711 (see Fig 16 orange sensor) which is a 24-bit analog to digital converter designed for load cells. In this ADC, we have two-channel A and B for different gain. Channel A can be programmed with a gain of 128 or 64, corresponding to a full-scale differential input voltage of  $\pm 20\text{mV}$  or  $\pm 40\text{mV}$  respectively, while channel B has a fixed gain of 32. The sensor looks like below:

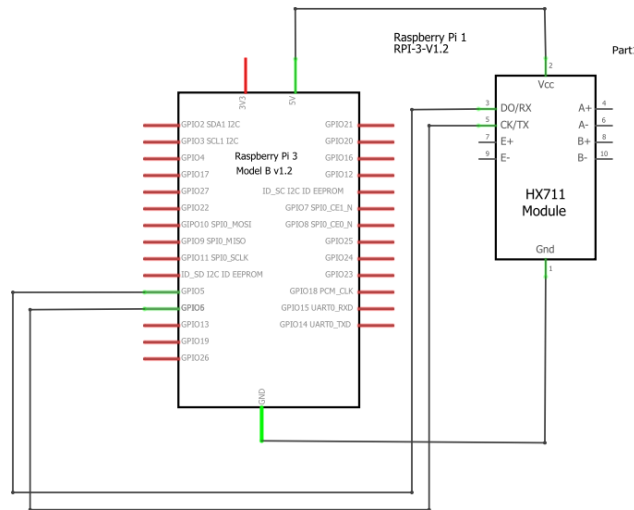


**Fig 14: Load Cell [7]**

#### a. Pin Configuration

| Pin No.<br>HX711 | Pin Name | Description                       | Raspberry Pi Pin Number |
|------------------|----------|-----------------------------------|-------------------------|
| 1                | VCC      | Used to power the sensor          | Pin no. 4 (+5V)         |
| 2                | GND      | Used to connect to ground         | Pin no. 39 ( GND)       |
| 3                | DT       | Serial Data Output                | Pin no. 29 (GPIO 5)     |
| 4                | SCK      | Power Down and Serial Clock Input | Pin no. 31 ( GPIO 6)    |

## b. Schematic Representation

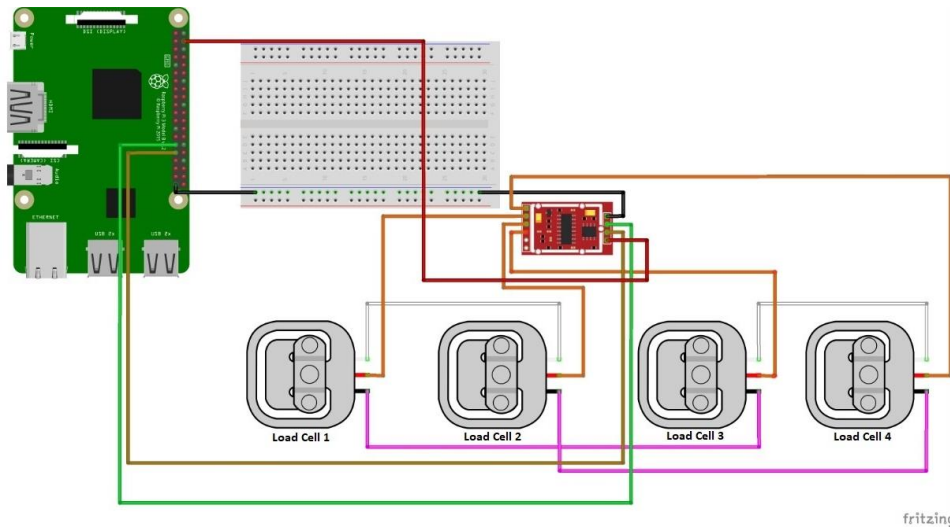


**Fig 15: Schematic diagram of HX711 and Raspberry Pi**

## c. Interfacing Connection with Raspberry Pi 3b+

As I have mentioned earlier the output of sensors is analog so an ADC HX711 is used, of which connection is shown in the below diagram. Here in the circuit diagram as you can see every sensor has three cables. Of which red cable is connected with HX711 and other cable are interconnected as shown in the picture. Here two of the sensors' red cable are connected to E+ and E- of HX711, while the other two will be either connected to A+ and A- or B+ and B- as per the selected gain. Meaning all four sensors are connected in a Wheatstone bridge, where E+ and E- can be considered as input of a Wheatstone bridge are connected to load cell 4 and load cell 1 respectively while A+ and A- can be considered as output of a Wheatstone bridge are connected to the load cell 3 and load cell 2 respectively. Now the HX711 is connected with Raspberry Pi as per shown in below diagram made using Fritzing:





**Fig 16: Connection of load cells and hx711 with Raspberry Pi**

#### **d. Calibration**

Before running the main load cells program, first, we have to calibrate the sensors. For that, we have to run the calibration file and get the value of OFFSET and SCALE and insert this value in the main load cells python file.

#### **e. Challenges Encountered**

Load cell was a difficult sensor to install and calibrate. And even after calibration it was giving false and different values. The values were changing as per the material on which we have fixed sensors. So at the end, we made the platform of wood with a hole in the middle so that the middle part of the sensor would not touch the ground and it can move freely when there is a weight. After that our output became linear with the weight. A sample result window from the firebase database has been shown below:

```
Beehive_weight_kg: 19.845993706944125
```

**Fig 17: Sample Output of Load cell**

#### **f. Things to remember**

- i. Here we are using four load cell sensors, so connection should be exactly as per the circuit diagram
- ii. The platform which is kept on load cells has to be mounted properly to touch all sensors so that the weight distribution on all four sensors are equal.
- iii. Follow the calibration process explained in the calibration python file.

#### 4. DHT22 (Nilesh Solanki)

DHT22 sensor is being used to measure temperature and humidity inside the beehive. Humidity inside the beehive affects vital processes within the hive, such as brood development, nectar processing and disease evolution. In addition to ambient humidity, bees create moisture via living processes such as bringing in water, moist nectar and even respiration. While it is important to keep the brood above a certain humidity level it is important that wet air does not condense on the inner hive walls or even the comb itself [8]. Temperature inside a beehive is also very important because if a beehive gets too hot, the bee brood dies and the honey gets dehydrated too quickly. If a beehive gets too cold, broods die off and the nectar cannot dehydrate fast enough to make honey. 35 degrees is the exact right temperature for a hive [9].

This sensor works on 3V to 5V and gives humidity in range of 0-100% with 2-5% accuracy and temperature range -40 to 80°C with  $\pm 0.5^\circ\text{C}$  accuracy. Below is the picture of the sensor DHT22:

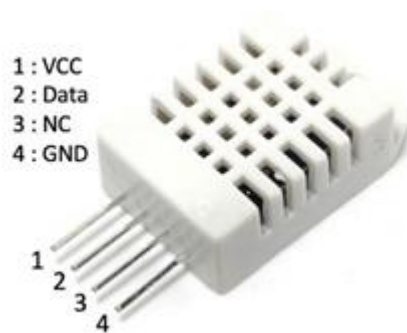


Fig 18: DHT22 [10]

##### a. Pin Configuration

| Pin.No<br>DHT22 | Pin Name | Description  | Raspberry Pi Pin<br>Number |
|-----------------|----------|--|----------------------------|
| 1               | VCC      | Power supply 3.5V to 5.5V                                    | Pin no. 2 (+5V)            |
| 2               | GND      | Connected to the ground of the circuit                       | Pin no. 39 ( GND)          |
| 3               | Data     | Outputs both Temperature and Humidity<br>through serial Data | Pin no. 7 (GPIO 4)         |

## b. Schematic Representation

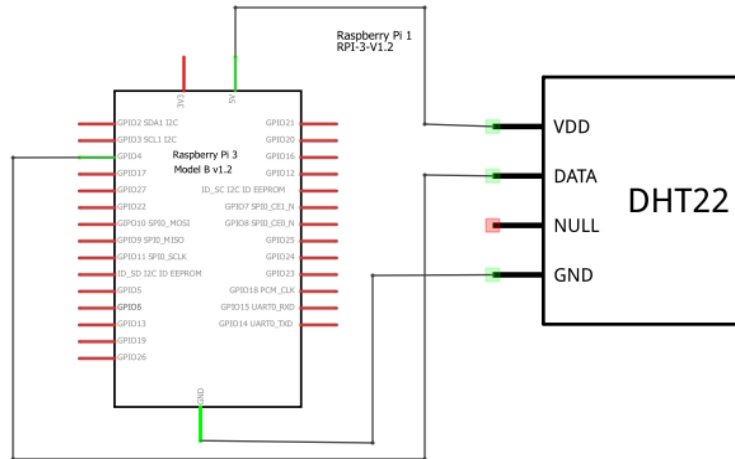


Fig 19: Schematic diagram of DHT22 and Raspberry Pi

## c. Interfacing Connection with Raspberry Pi 3b+

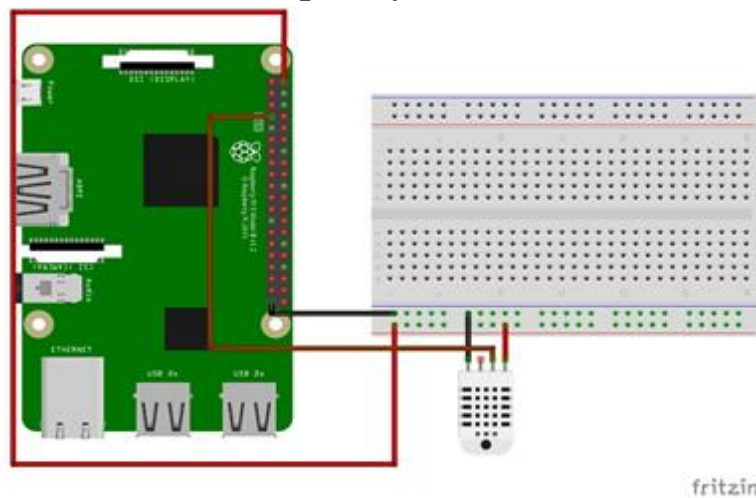


Fig 20: Connection of DHT22 with Raspberry Pi

A sample result window from the firebase database has been shown below:

```
Beehive_humidity: 50
Beehive_temp: 25.600000381469727
timestamp: "20190809_124623"
```

Fig 21: Sample Output of DHT22

#### d. Things to remember

- i. Incorrect wiring is common. Pin 2 is data and Pin 3 is not used. It is easy to accidentally swap them.
- ii. Keep the wiring short (less than 15cm). Longer wires can cause interference. The datasheet states that wire distances greater than 100cm requires 5V and should use shielded cable for better result.

### 5. GPS (*Dilip Maharjan*)

GPS sensor is used in the beehive to know the exact location. GPS stands for Global Positioning System. GPS system uses satellite based communication system protocol that uses satellites and ground station to determine the exact location on earth. In this project we are using Neo 6M GPS sensor. It is a serial device which uses UART (Universal Asynchronous Transmitter) communication.



**Fig 22: Neo 6M GPS module [Source: Google Images]**

#### a. Working principle of GPS module

To get exact position using GPS we need 4 satellites. A signal of frequency about 1.1 GHz to 1.5 GHz is transmitted from the satellite which is then received by the GPS module and the position and time is calculated. The receiver GPS module receives the radio frequency (information signal 1.1-1.5 GHz) from the satellites, then it calculates its distance from the satellite. The distance can be calculated by the time that is required for the signal to travel from the satellite to the GPS receiver.

The GPS module transmits data in NMEA (National Marine Electronics Association) format using 9600 baud rate. The NMEA strings contain different values. A description of a NMEA string is shown in the below table [11].

eg: \$GPGGA,163241.000,1226.5639,N,06346.6174,W,1,08,2.1,507.1,M,-74.7,M,,0000\*7C

|                         |            |        |                                  |
|-------------------------|------------|--------|----------------------------------|
| UTC time                | 163241.00  |        | hhmmss                           |
| Latitude                | 1226.5639  |        | ddmm.mmmm                        |
| N/S indicator           | N          |        | N=North ,S=South                 |
| Longitude               | 06346.6174 |        | dddmm.mmmm                       |
| E/W indicator           | W          |        | E=East ,W=West                   |
| Position fix indicator  | 1          |        | Fix GPS SPS mode                 |
| Satellites used         | 08         |        | Number of satellites used        |
| HDOP                    | 2.1        |        | Horizontal Dilution of Precision |
| MSL altitude            | 507.1      | Meters | Mean Sea Level                   |
| Units                   | M          | Meters |                                  |
| Geoid Separation        | 74.7       | Meters |                                  |
| Units                   | M          | Meters |                                  |
| Age of Diff Corr        | -          |        | Null field if DPGS is not used   |
| Diff Ref station id     | 0000       |        |                                  |
| Checksum                | *7c        |        |                                  |
| Carriage return line fd | <CR><LF>   |        | End of Message                   |

## b. Pin Configuration

| Pin no | GPS module | Description   | Raspberry Pi pin no. |
|--------|------------|---|----------------------|
| 1      | VCC        | A 5v voltage is required to power on the GPS module                                 | Pin no.2(VCC)        |
| 2      | GND        | The Ground pin (GND) is connected to the Gnd  | Pin no.6(GND)        |
| 3      | Tx         | The Transmitter (Tx) pin should be connected to the Receiver Rx pin of Raspberry Pi | Pin no.10(Rx)        |

### c. Circuit Diagram

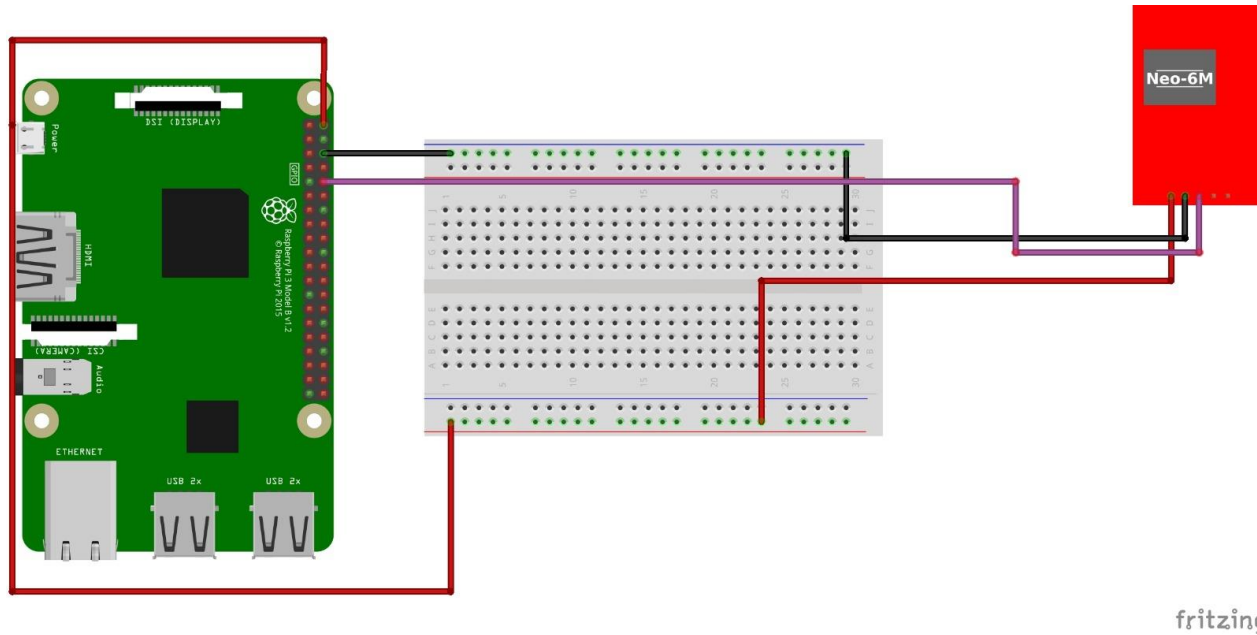


Fig 23: Connection of GPS module with Raspberry Pi

### d. Schematic Representation

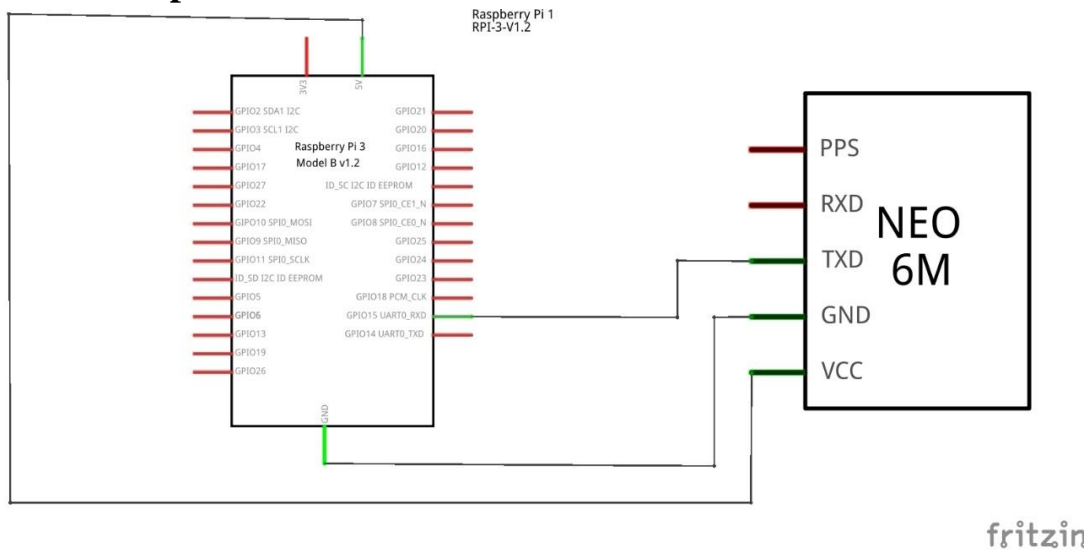


Fig 24: Schematic Representation of GPS module with Raspberry Pi

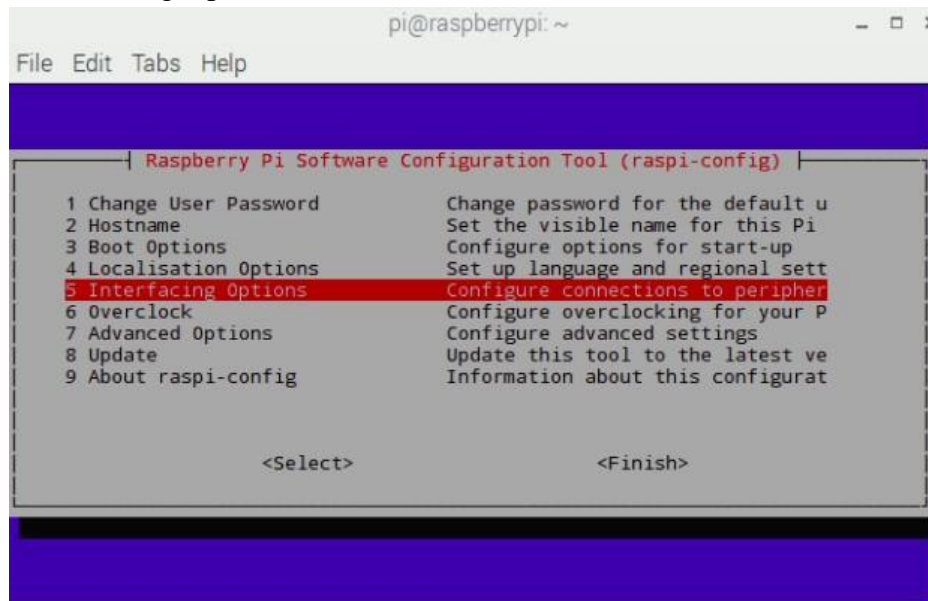
### e. Interface configuration

Raspberry Pi-3B+ has two inbuilt UART Mini-UART and PL011. PL011 is ARM based UART which is better than Mini UART. In Raspberry pi-3 Mini UART is used for accessing Linux console whereas PL011 is used for Bluetooth connection. So following process should be followed to change the UART serial ports.

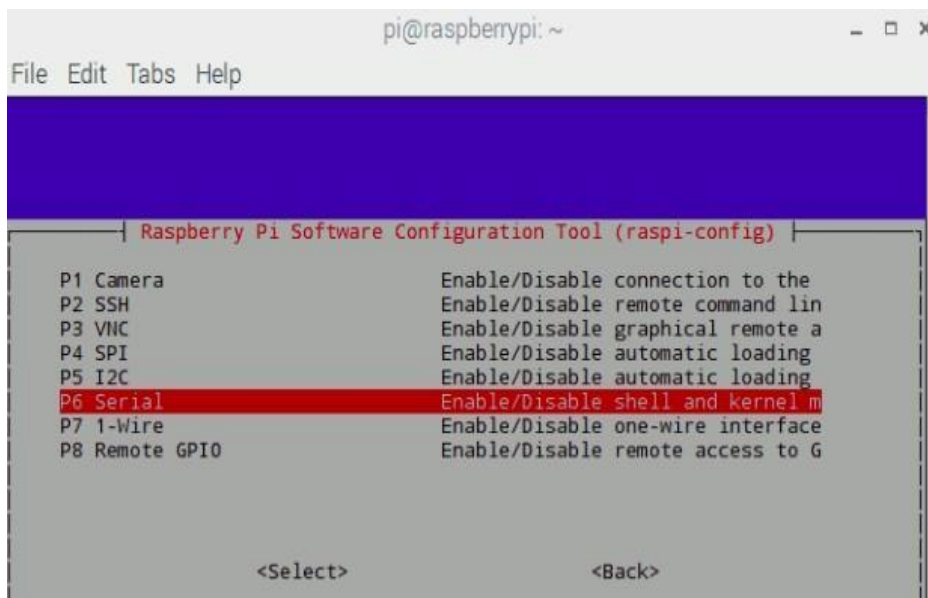
**Configuration of UART communication in Raspberry-pi**

**Step i:** In Raspberry Pi terminal, type the command **sudo raspi-config**

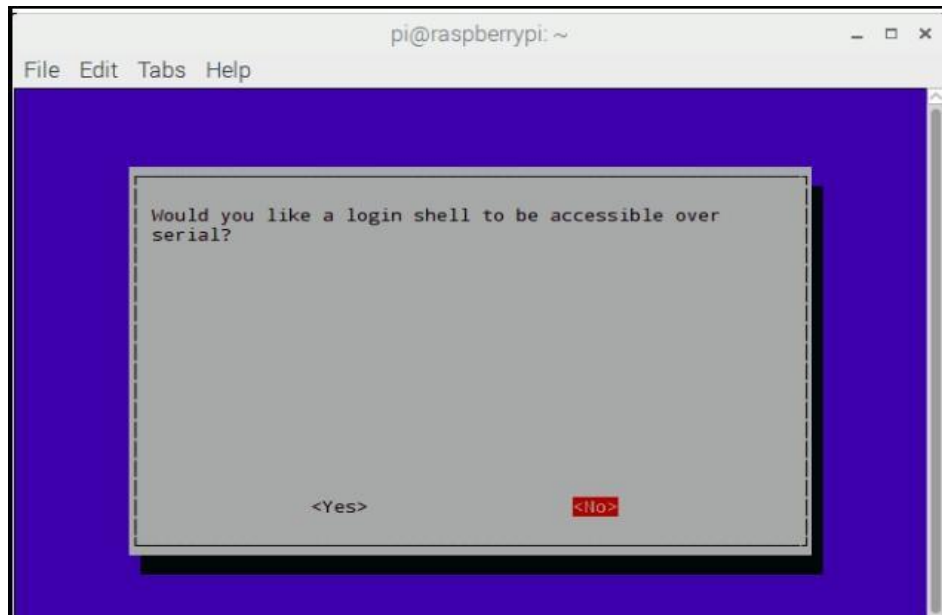
**Step ii:** Select Interfacing Options



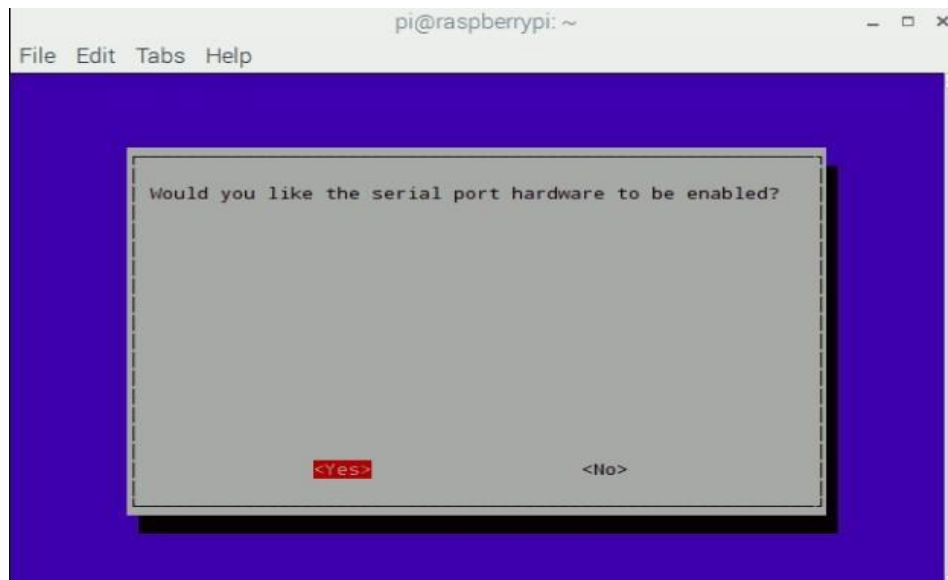
**Step iii:** Select serial option to enable UART



**Step iv:** It will ask for shell accessible over serial. Select 'No' as below picture.



**Step v:** At the end it will ask enabling hardware serial port. Select 'Yes'



**Step vi:** Then Reboot Raspberry pi.

Now we can check whether the UART port is mini-UART (ttys0 ) or PL011-UART(ttyAMA0) which is mapped to UART pin 14 and 15 by entering the following command **ls -l /dev**

The UART mapping is shown as below figure.



```

pi@raspberrypi: ~
File Edit Tabs Help
brw-rw---- 1 root disk 1, 4 Aug 11 05:52 ram4
brw-rw---- 1 root disk 1, 5 Aug 11 05:52 ram5
brw-rw---- 1 root disk 1, 6 Aug 11 05:52 ram6
brw-rw---- 1 root disk 1, 7 Aug 11 05:52 ram7
brw-rw---- 1 root disk 1, 8 Aug 11 05:52 ram8
brw-rw---- 1 root disk 1, 9 Aug 11 05:52 ram9
crw-rw-rw- 1 root root 1, 8 Aug 11 05:52 random
drwxr-xr-x 2 root root 60 Jan 1 1970 rfkill
crw-rw-r-- 1 root root 10, 58 Aug 11 05:52 rfkill
lrwxrwxrwx 1 root root 7 Aug 11 05:52 serial0 -> ttyS0
lrwxrwxrwx 1 root root 5 Aug 11 05:52 serial1 -> ttyAMA0
drwxrwxrwt 2 root root 40 Aug 11 06:01 tmp
drwxr-xr-x 3 root root 160 Aug 11 05:52 tmp
crw-rw---- 1 root spi 153, 0 Aug 11 05:52 spidev0.0
crw-rw---- 1 root spi 153, 1 Aug 11 05:52 spidev0.1
lrwxrwxrwx 1 root root 15 Jan 1 1970 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 Jan 1 1970 stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 Jan 1 1970 stdout -> /proc/self/fd/1
crw-rw-rw- 1 root tty 5, 0 Aug 11 05:52 tty
crw-rw-rw- 1 root tty 4, 0 Aug 11 05:52 tty0
crw-rw-rw- 1 pi tty 4, 1 Aug 11 05:52 tty1
crw-rw-rw- 1 root tty 4, 10 Aug 11 05:52 tty10
crw-rw-rw- 1 root tty 4, 11 Aug 11 05:52 tty11
crw-rw-rw- 1 root tty 4, 12 Aug 11 05:52 tty12

```

## f. Changing serial ports of Raspberry pi-3

For better performance of GPS sensor we can swap the serial port mini-UART (ttys0) to PL011 (ttyAMA0). To do this we should use the following steps [11]:

**Step i:** First we need to open config.txt file using below command

**sudo nano/boot/config.txt**

**Step ii:** Then add the line at the end of file as shown below

**dtoverlay=piminiuart-bt**

**Step iii:** After making changes save and reboot raspberry pi.

Now check the serial port using the below command

**sudo -ls/ dev**

## g. Challenges Encountered

While interfacing GPS module we faced following challenges:

- i. GPS module took time to turn on** - GPS module took some time to turn on. It didn't have a fixed time to turn on. It took 3-5 minutes sometime to turn on whereas it sometimes took around 15-20 minutes to turn on
- ii. GPS module works only in open area** - We tried turning on the GPS module in closed building. It took longer to turn on but when we place near to the window then it took less time.
- iii. Swapping serial ports** - For swapping serial ports we need to make changes in config.txt file, while adding new line to the config.txt file we first need to backup all the content of Raspberry Pi because Raspberry Pi sometimes cannot boot while making changes in the config.txt file. So to be on the safe side we must need to backup all the contents while changing the serial port.

## h. Things to remember

While taking the NMEA string in the code we are just taking the time, Longitude and latitude values so in the code, we are just taking the 1st, 2nd and 4th place value of the array. As the 3rd array shows direction north "N" so we are discarding it. Also while we are interfacing our

GPS module we are just connecting three pins of GPS module (VCC, GND, Tx) to Raspberry Pi as Raspberry pi is only receiving data from GPS sensor so Rx pin of GPS sensor is not connected to the Tx pin of Raspberry pi.

```
lat in degrees: "48.5745"
```

```
long in degree: "13.4555"
```

**Fig 25: Sample output of GPS sensor**

## 6. Light sensor KW mobile (*Dilip Maharjan*)

The light sensor is used in beehive to know whether the beehive is kept on the sensor box or not. In this project we have used KW mobile light sensor. KW mobile Light sensor is a digital light sensor equipped with LDR. The digital output high/low can be adjusted using the potentiometer. This sensor is very sensitive to ambient light and it is very useful in detecting the brightness of ambient light. If the brightness of the light is lower than the predefined threshold maintained by using the potentiometer the output is high if the brightness reaches or exceeds the predefined threshold the output is low. The light module is made up of LM393 comparator, high-quality photoresistor, a potentiometer to adjust sensitivity [12].



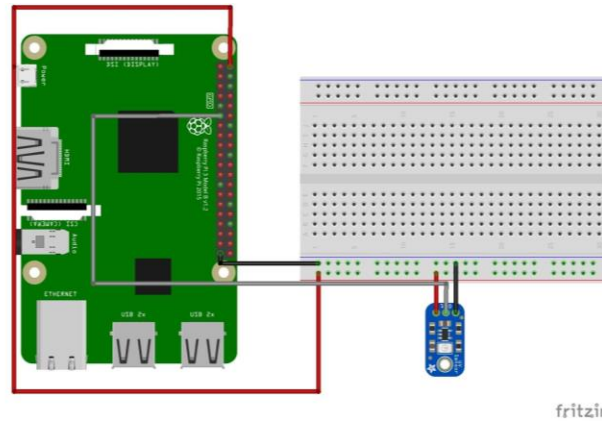
**Fig 26: kwmobile light sensor [Source: Google Images]**

### a. Pin Diagram & Breadboard connection

| Pin No. | Pin name(Light sensor) | Description   | Raspberry Pi Pin no. |
|---------|------------------------|---|----------------------|
| 1       | Vcc                    | 5v voltage is required to turn on Light sensor        | Pin no.2(Vcc)        |
| 2       | DO                     | The DO pin gives the digital output                   | Pin no.11(GPIO 17)   |
| 3       | GND                    | The ground must be connected to make circuit complete | Pin no.39(GND)       |

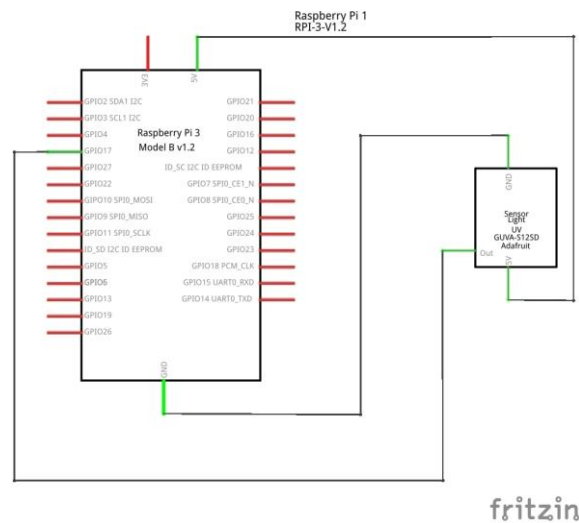
**Table:** Pin configuration of light sensor with Raspberry Pi.

## b. Circuit Diagram



**Fig 27: Circuit connection of kwmobile light sensor with Raspberry Pi**

## c. Schematic Representation



**Fig 28: Schematic Representation of the light sensor with Raspberry Pi 3B+**

## d. Challenges Encountered

While interfacing the light sensor we had complications with the brightness setting for the light sensor. We need to adjust the digital output of the sensor low/high by adjusting the potentiometer. So we faced a problem while adjusting the sensitivity of the light sensor.

```
light: 0
```

**Fig 29: Sample Output of Light Sensor**

## 7. USB Microphone (*Abhinandan Shetty Rathnakar*)

The bees sound come from the buzz created by the wing movement. The sound and frequency of the bees are important to understand the characteristics of the bees inside a beehive. Along with it the data of the sound, the frequency of the bees helps the beekeepers to know about the population of the bees inside the beehive as with the increase in the population the wing movement of the bees decrease [13].

The Raspberry Pi-3b+ is capable of recording audio through its USB 2.0 ports using the Advanced Linux Sound Architecture (ALSA). The RPi can sample at 48 kHz at a bit depth of 16-bits, which allows the user to record fairly good quality audio and also estimate the frequency of the sound. The advantage of USB is that generally microphone will be detected automatically by Raspbian, it's plug & play [14].

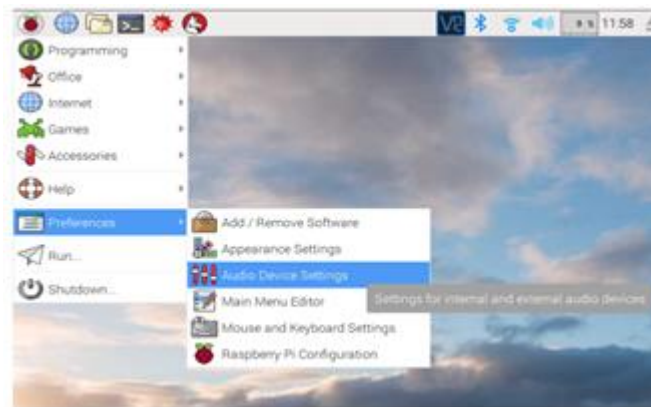


**Fig 30: USB Microphone [Source: Google Images]**

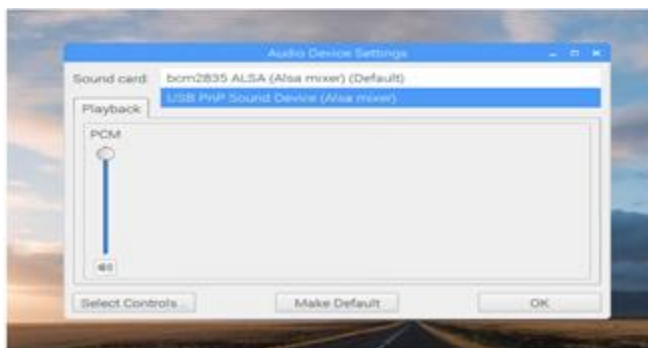
### a. Steps of implementation:

After acquiring the USB mic and plugging it into one of the RPi USB ports, make the following settings:

**Step i:** Open the Audio device settings in the Raspberry Pi



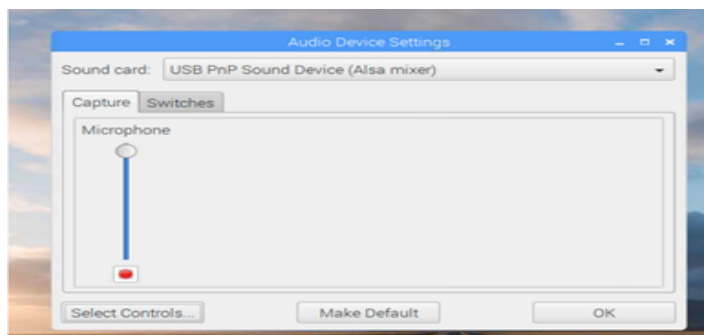
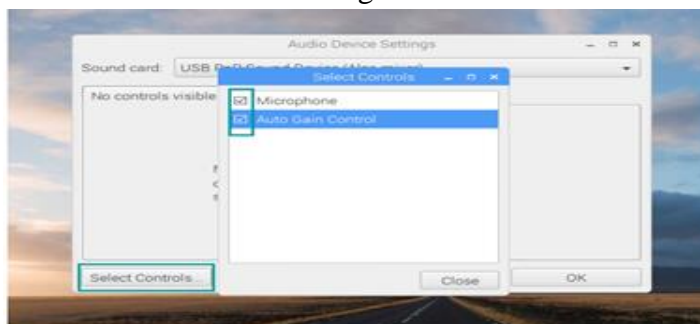
**Step ii:** Then see the USB PNP Sound Card in the drop-down menu of the Sound Card.



**Step iii:** After selecting, the default below does not show any settings, need to click Select Controls, and check the two controls, then click 'Close' button to close the page.



**Step iv:** Now, we can see two tabs, Capture and Switches. You can set the volume of Capture to the max, and click OK to close the setting.



## b. Installing the Correct Audio Tools for the Raspberry Pi

The Python library ‘pyaudio’ is used to record and play audio data from the USB mic. Before we can get started with ‘pyaudio,’ we need to ensure that the RPi has all the necessary prerequisites by installing the following package:

```
sudo apt-get install libportaudio0 libportaudiocpp0 port audio19-dev
```

If the above is successful, then download the ‘pyaudio’ library by installing to Python 3.x with ‘pip3’.

## c. Testing the USB Mic

Open Python 3.x and type the following:

```
>>> import pyaudio
>>> p = pyaudio.PyAudio()
>>> for ii in range(p.get_device_count()):
>>>     print(p.get_device_info_by_index(ii).get('name'))
```

This should output the index of each audio-capable device on your Pi.

Take note of the index of the USB device, because we will need to adjust the pyaudio device index according to the sequence. For example, our USB device index is “1”.

Now that we’ve identified the USB mic’s index, we can record the audio with pyaudio and apply the Fast Fourier Transform (FFT) to estimate the frequency of the sound.

## d. Frequency Estimation

We can compute the FFT of the signal and investigate the frequency content. The Python FFT function in Python is used as follows:

```
np.fft.fft(signal)
```

## e. Challenges Encountered

- i. During the import of numpy within python3 code got the error of unable to open shared Object file
  - ii. During the installation of the pyaudio the broken package error was detected.
- In order to overcome these challenges we flashed the Raspbian OS again

## f. Things to remember

Always check for the device ID number before configuring the microphone.

Once all the connections are verified and the interface configuration is done we can dump the program into the raspberry pi. The output consists of the sound and the frequency value. The sample output is shown below:

```
Sound_Frequency : 102.28271484375
Sound_Volume: 0.00041762691252982063
```

**Fig 31: Sample output of USB Microphone**

## **8. MPU6050 (Gyroscope & Accelerometer) (*Abhinandan Shetty Rathnakar*)**

The Gyroscope and accelerometer are required to know whether the beehive has fallen down or it's disturbed from its original position by reading the data given by this sensor regarding the orientation and acceleration.

MPU6050 sensor module is an integrated 6-axis Motion tracking device. It has a 3-axis Gyroscope, 3-axis Accelerometer all in a single IC. It can accept inputs from other sensors like 3-axis magnetometer or pressure sensor using its Auxiliary I2C bus. If external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output. A Raspberry Pi 3b+ can communicate with this module using I2C communication protocol. Various parameters can be found by reading values from addresses of certain registers using I2C communication. Gyroscope and accelerometer reading along X, Y and Z axes are available in 2's complement form. Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit [15].



**Fig 32: MPU6050 Gyroscope and Accelerometer [Source: Google Images]**

**a. Pin Configuration of MPU-6050**

| <b>Pin No MPU-6050</b> | <b>Pin Name</b> | <b>Pin Description</b>  | <b>Raspberry Pi Pin Number</b> |
|------------------------|-----------------|---|--------------------------------|
| 1                      | VCC             | Power supply pin. Connect this pin to +5V DC supply   | Pin No 1 (3.3V supply)         |
| 2                      | GND             | Ground pin. Connect this pin to Ground Connection   | Pin No 6 (GND)                 |
| 3                      | SCL             | Serial Clock pin. Connect this pin to Raspberry pi SCL pin  | Pin No 5 (GPIO2 SCL1 I2C)      |
| 4                      | SDA             | Serial Data Pin. Connect this pin to Raspberry pi SDA pin   | Pin No 3 (GPIO3 SDA1 I2C)      |
| 5                      | XDA             | Auxiliary serial Data pin. This pin is used to connect other I2C interface enabled sensors SDA pin to MPU-6050  | No Connection                  |
| 6                      | XCL             | Auxiliary Serial clock pin. This pin is used to connect other I2C interface enabled sensors SCL pin to MPU-6050   | No Connection                  |
| 7                      | AD0             | I2C slave address LSB pin. This is 0 <sup>th</sup> bit in 7-bit slave address of the device. If connected to VCC then it is read as a logic one and slave address changes | No Connection                  |
| 8                      | INT             | Interrupt digital output pin  | No Connection                  |



## b. Circuit Connection

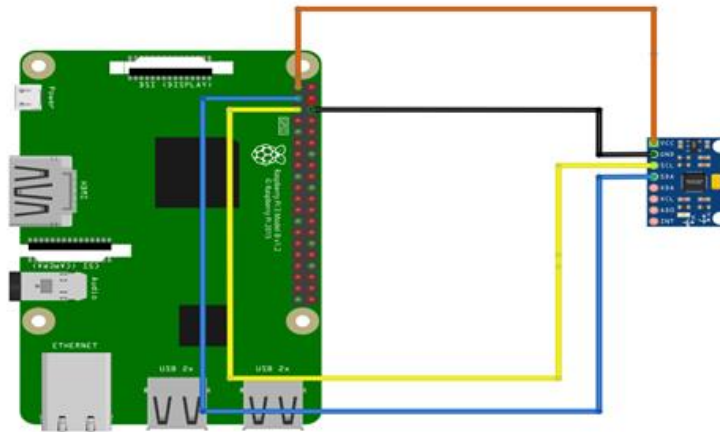


Fig 33: Circuit Connection of MPU – 6050 with Raspberry Pi

## c. Schematic Representation of the connection of Raspberry pi and MPU-6050

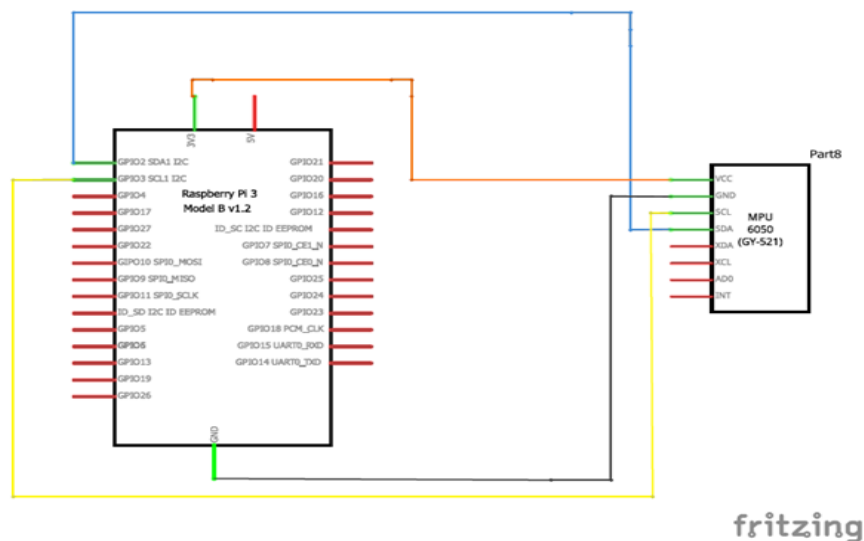


Fig 34: Schematic Diagram

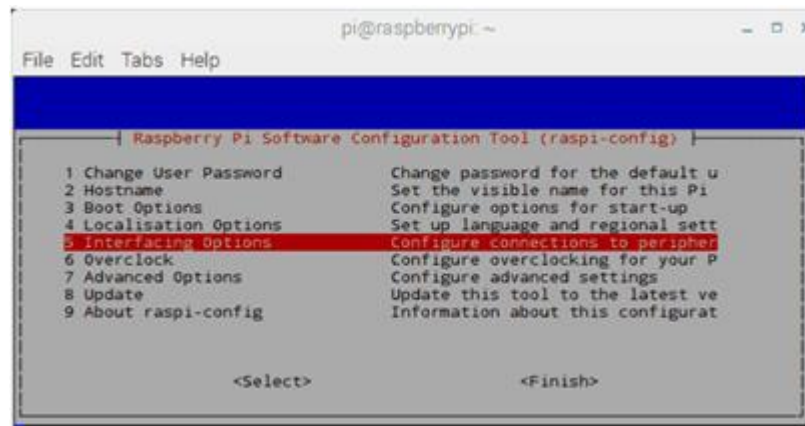
## d. Raspberry Pi I2C Configuration

Before start interfacing I2C devices with Raspberry some prior configurations need to be done. These configurations are given as follows:

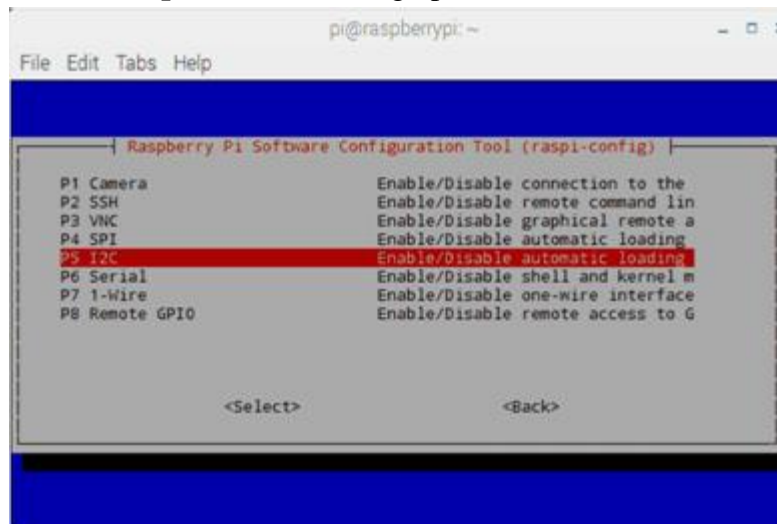
First, we should enable I2C in Raspberry Pi. We can enable it through the terminal which is given below:

**Step i:** `sudo raspi-config`

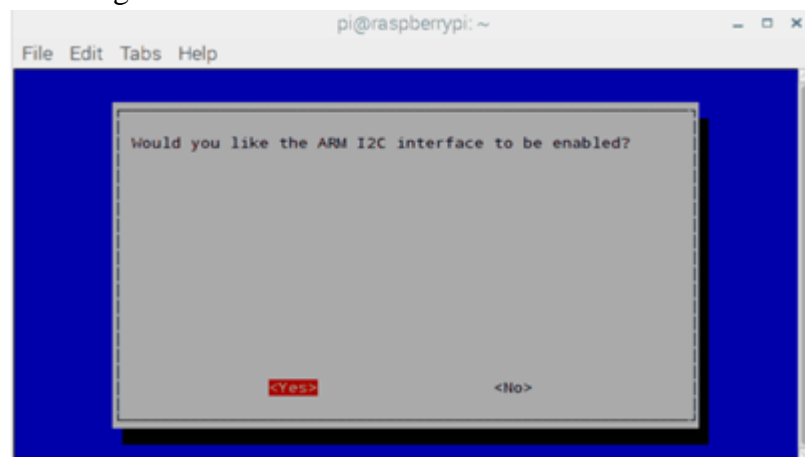
**Step ii:** Select Interfacing Configurations



**Step iii:** In Interfacing option, Select -> I2C



**Step iv:** Enable I2C configuration



**Step v:** Select 'Yes' when it asks to Reboot.

### e. Scan or Test I2C device on Raspberry Pi

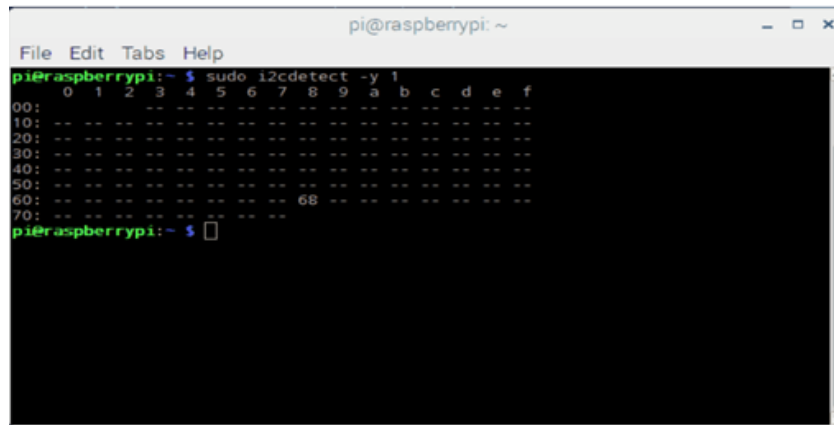
Now, we can test/scan for any I2C device connected to our Raspberry Pi board by installing i2c tools. We can get i2c tools by using apt package manager. Use following command in Raspberry Pi terminal.

#### Step i: `sudo apt-get install -y i2c-tools`

Now connect any I2C based device to the user-mode port and scan that port using following command

#### Step ii: `sudo i2cdetect -y 1`

Then it will respond with the device address.



```

pi@raspberrypi:~$ sudo i2cdetect -y 1
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- 68 -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~$

```

i2cdetect command will scan the I2C port to get the address of device if connected. If no device is connected to I2C port then it will return field with (- -).

### f. Access I2C devices using SMBus

We can access I2C bus on Raspberry Pi using SMBus. SMBus is a subset of I2C bus/interface. SMBus provides support for I2C based devices. While writing the program to access I2C based device, make use of SMBus commands.

While developing program for Raspberry Pi I2C communication in python, use SMBus library package which has great support to access I2C devices. So, we should add SMBus support for Python by using apt packet manager,

**`sudo apt-get install python-smbus`**

### g. Challenges Encountered

The I2C device address detection fails unless the proper connection is ensured between the I2C pins of the sensor and the Raspberry Pi. Improper connection will give a null output when the address detect command is executed. Make sure the connection is robust by soldering.

### h. Things to remember

Even though the sensor is at rest it does not always show the result 0 with respect to orientation and acceleration as finding the exact 0 reference is difficult.

Once all the connections are verified and the interface configuration is done we can dump the program into the raspberry pi. The sensor outputs the acceleration and orientation in X, Y, Z axis. The sample output is as shown below.

```
Acceleration_xout: -0.008056640625
Acceleration_yout: -0.01025390625
Acceleration_zout: -1.02099609375
Gyro_xout: -6.343511450381679
Gyro_yout: -1.3129770992366412
Gyro_zout: -2.2748091603053435
```

**Fig 35: Sample Output of Gyroscope and Accelerometer**

## **9. Raspberry Pi Night Vision Camera** *(Chaitanyananda Buruganahalli Sreedhar)*

### **a. Camera Module Introduction**

For the beekeeper and the bee-parental ships having real-time images from inside the beehive would yield a greater perspective to understand the behaviour of the bees. By capturing the images of the interior of the beehive, it enables the beekeeper to see the speed bees build their combs and collect honey, the live stream is just an add-on feature that every beekeeper will enjoy the live stream. *(Note: Currently, the camera live stream is not a part of Database/Integration and is at staging level. This needs to be integrated with the APP at later stages).*

The sole purpose for selecting Raspberry Pi Night Vision Camera instead of a normal camera is that the interior of the beehive remains in darkness for most of the time. In order to capture the behaviour of the bees in such dark lighting, setup requires high precision Night Vision Camera which works with the help of Infrared lights.

Raspberry Pi Infrared Camera Module is a low-cost Night Vision, adjustable-focus Camera module, designed for Raspberry Pi. It features two high-intensity Infrared LED spotlights for night time recording. The IR LED's are powered directly from the CSI port and are capable of lighting an area at a distance of up to 8m. The camera also features an adjustable 3.6mm focal length lens and 75.7 degree viewing angle. This Raspberry Pi night vision camera uses the same OV5647 as the standard Raspberry Pi camera and is, therefore, able to deliver a crystal clear 5MP resolution image or 1080p HD video recording at 30fps [16].

The working principle is that in the dark environment without visible light or low light, the infrared light-emitting device actively projects infrared light onto the object, and the infrared light is reflected by the object.

## b. Hardware Configuration

The camera module attaches to Raspberry Pi with the help of a 15 Pin Ribbon Cable (as shown in Fig 36), to the dedicated 15-pin MIPI Camera Serial Interface (CSI), which is designed especially for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the BCM2835 processor!



**Fig 36: Hardware connection of Night Vision Camera to Raspberry Pi [Source: Google Images]**

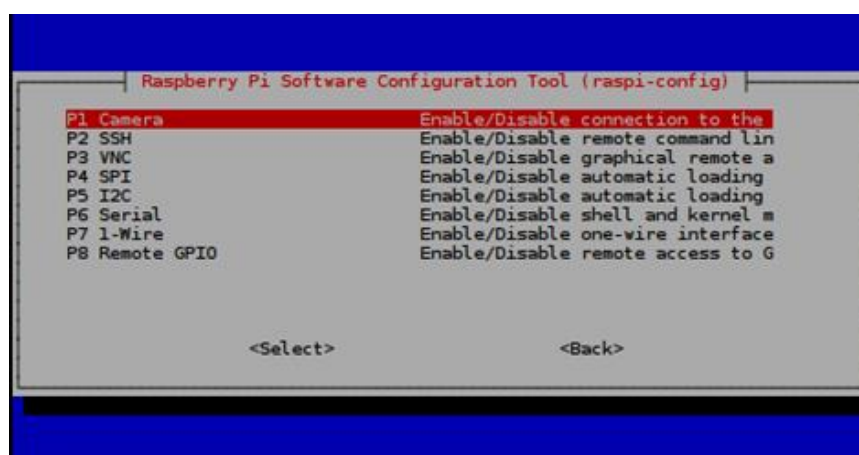
## c. Interface Configuration

After the hardware attachment of camera module to Raspberry Pi. We need to enable the Camera Interface in the Raspberry Pi. Below are the steps to enable Camera Interface:

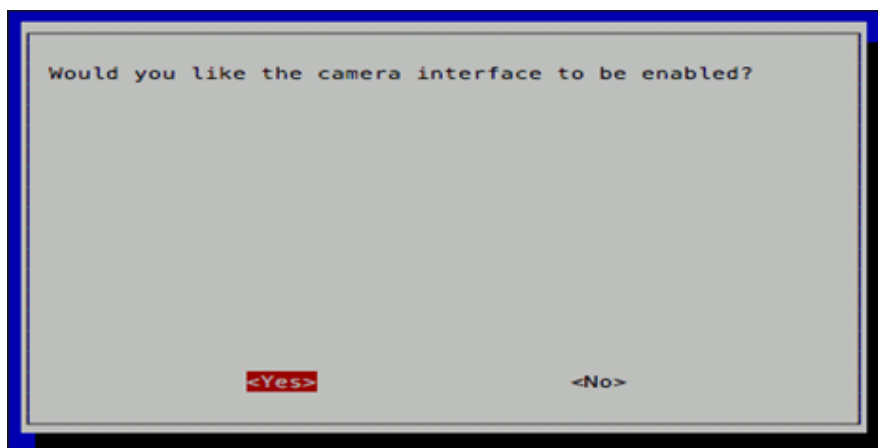
**Step i:** Open a Terminal and execute the command **sudo raspi-config**. Select 'Interfacing Options' from the config tool screen.



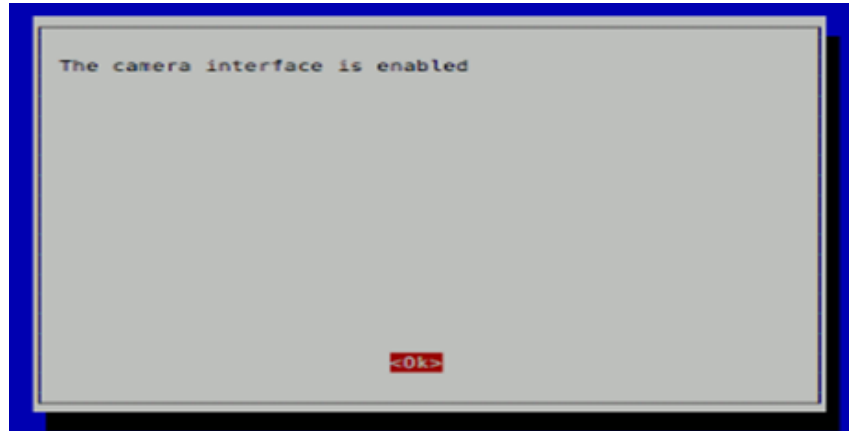
**Step ii:** Select 'Camera' interfacing option



**Step iii:** System prompts us to choose whether Camera interface should be enabled, select 'Yes'.



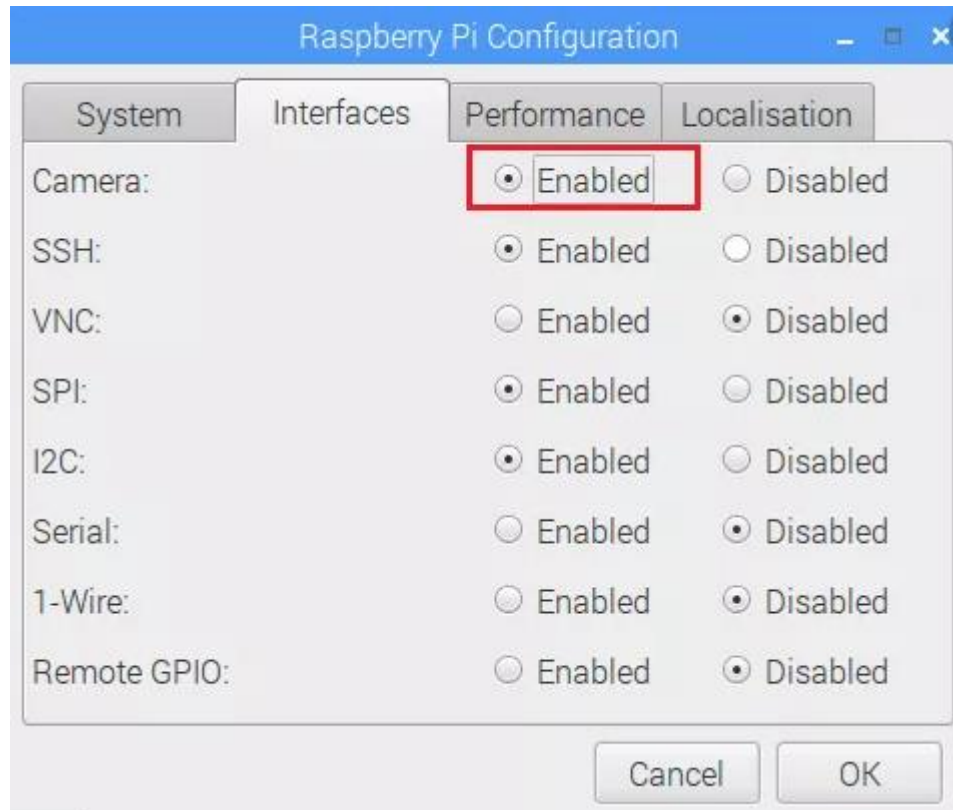
**Step iv:** Camera Interface is Enabled.



**Step v:** Systems prompts for the reboot, select 'Yes'



The other way to enable Camera Interface is through GUI. To do so, first navigate to Raspberry Pi symbol on the menu bar in the desktop of Raspberry Pi. Select 'Preferences' and 'Raspberry Pi Configuration'. Go to 'Interfaces' tab -> Click on 'Enabled' radio button corresponding to Camera Interface and click 'Ok. Then, manually reboot the system.



The program for the camera module is intended to capture the images in **.jpeg** format. Here, we have employed ‘**picamera**’ module of python to capture the images. The orientation of each image to be uploaded has been set to 180 deg (change the orientation as required). The resolution of each image is **640×480**. The image recorded is saved on the system as well as uploaded to Firebase CloudStore in real-time at designated intervals.

#### d. Challenges Encountered

- i. **Manual Adjustment of the focus** – As this camera is equipped with manual focus component, we have to ensure that the target (beehive interior) is at optimum focus. It is quite hard to attain optimal focus, often leading to blurry images. In order to obtain high clarity images, we need to take multiple images by trial and error method, check if the image is sharp enough so as to achieve optimum focus.
- ii. **Storing Image data in Firebase DB** – Another challenge was to store the image data in real-time DB Firebase CloudStore. As it is a multimedia data, the data size is usually large and conventional data types such as integers, characters or string can’t be employed. So, the solution was to store the images as **BLOB** (binary large object). This is handled by the ‘**bucket**’ method of Firebase module. The images are stored in the ‘**Storage**’ bucket of



Firebase. We added separate configuration APIs in the firebase code so as to send the images to DB.

- iii. **Size of image** – One more challenge was the size of the image. Earlier, we used **1920×1080** resolution which yielded images >1 MB. This consumed a lot of data per day to be transmitted over the internet to DB. So, we reduced the resolution to **640×480** resolution which reduced the size of an image to approx. 200 KB. Hence saving a lot of data. And, the quality of the image is also retained. This trade-off between the size and quality is admissible.

#### e. Things to remember

- i. Please ensure camera focus is completed prior to plugging the device into the Raspberry Pi and switching it on.
- ii. The LED fill lights on both sides will warm up (generally in 40 to 50°C) when they working, but it is normal. You can adjust the micro-adjustable resistor (besides the lights) to turn down the lights to Lower temperatures.
- iii. If the program yields Camera Module not found error, then –
  - Check if the physical connection of camera is proper.
  - Check if the Camera Interface is turned on in the Raspberry Pi.

#### f. Sample Image –



**Fig 37: Sample image taken by Night Vision Camera**

## V. Software Integration – (*Chaitanyananda Buruganahalli Sreedhar*)

### 1. Database

#### a. Brief introduction

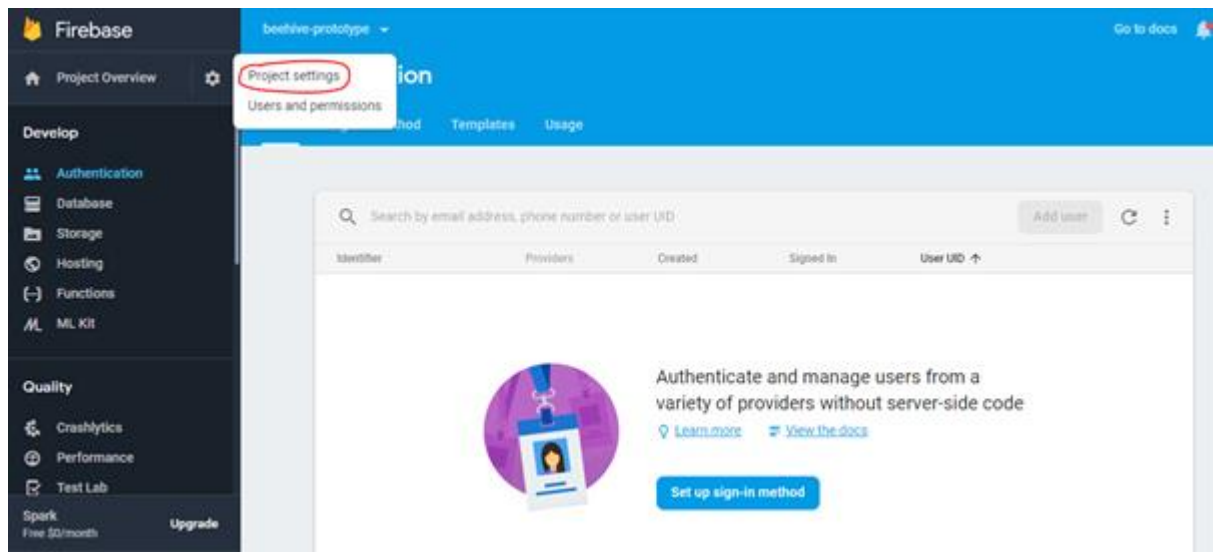
As stated in the Requirement Analysis document, for our project we have used Google's Firebase Cloud Firestore database. The Cloud Firestore is a real-time database where-in the data recorded by the sensor can be stored in real-time through HTTP protocol. Since it is a No-SQL database, there are high scalability and high flexibility characteristics which provide us with the provision to expand the documents in the DB.

The main reason for selecting the Firebase database is as follows - Firebase, being an app development platform that provides integrated tools to help us build, grow and monetize our apps [17]. Since 'PlanBee' project's mobile application is built using this platform, we have chosen the Firebase database-related services such as 'Cloud Firestore' and 'Storage Bucket' to store the data from all the sensors and camera images respectively. Small snippets of codes pertaining to sending the sensor data to database have been written in respective sensor module codes. The sensor data of all the sensors are sent to Firebase Cloud Store as JSON strings, whereas the image data is sent as a blob. To facilitate the time recording, timestamp (YYYYMMDD\_HHMMSS) is also sent along with the sensor data and each sensor's data is stored under such specific time stamps. Data pertaining to each sensor are added as separate collections. Please refer the database snapshots in the results section.

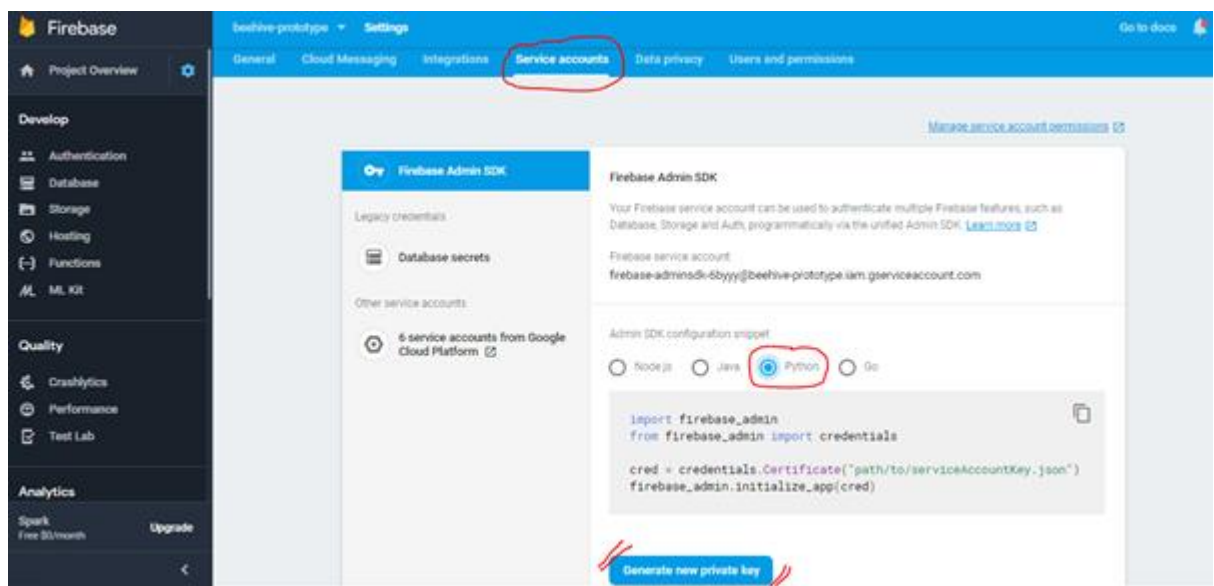
#### b. Setting up Database

**Pre-requisite** – The Firebase project is created and the database has read/write access.

**Step i:** Open Firebase project with the designated URL, click on the settings icon and select 'Project Settings' as shown in the snapshot

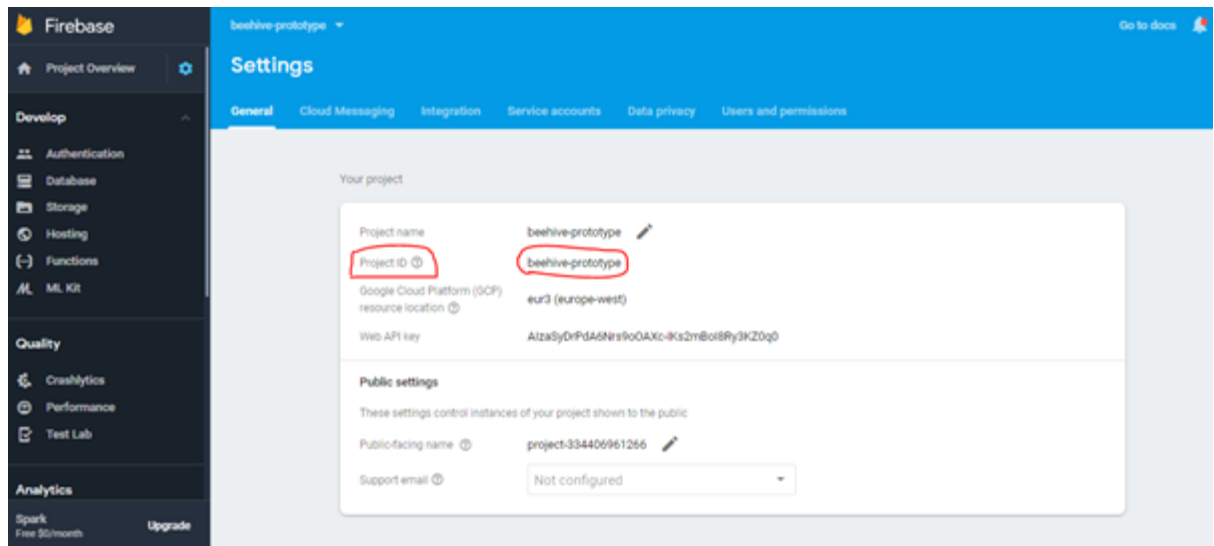


**Step ii:** Select Service Accounts-> Choose ‘Python’ in Admin SDK configuration snippet and click on ‘Generate new private key’

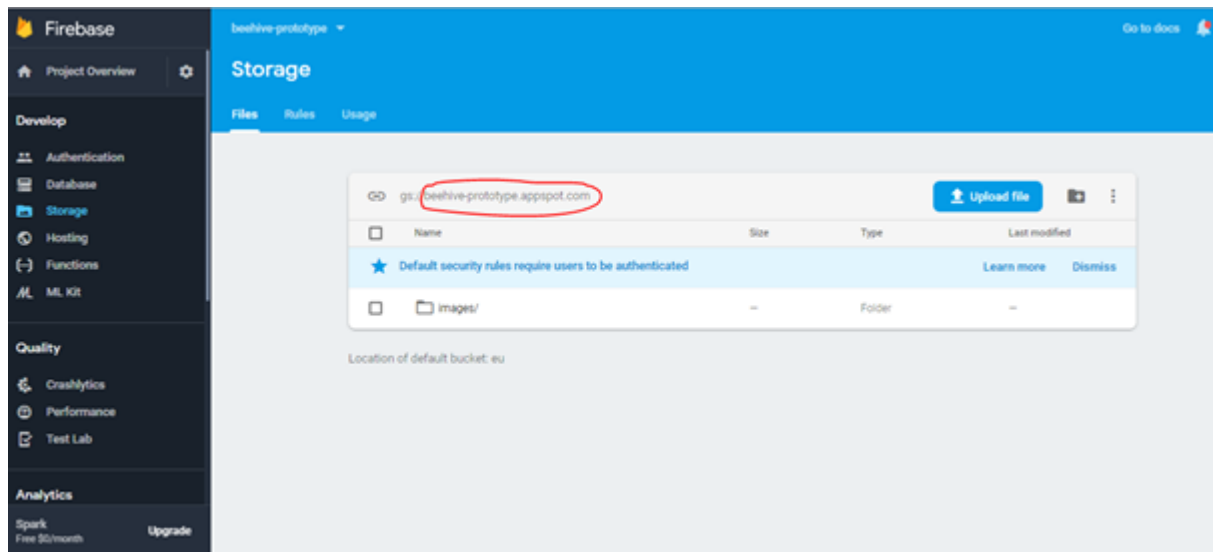


**Step iii:** Download the Private Key. The key will be in **.json** format. This key is used by the firebase setup program ‘**firebase.py**’ (used to set credentials) in our project. The firebase program requires the key generated, Project-ID and ‘Storage’ bucket URL for establishing proper connection with the database. The project-id and Storage bucket URL can be retrieved as follows.

**Step iv:** Project-id can be located by navigating to ‘General’ tab



**Step v:** Storage bucket URL can be found under ‘Storage’ as shown in the snapshot below



### c. Challenges Encountered

- i. **Change of database** – Initially we had used Firebase ‘Real-time Database’. In this database, the data was stored in unstructured JSON trees. This is not useful as we have a lot of data from several sensors. Hence, we migrated to Firebase ‘Cloud Firestore’ version of Firebase database. This solution enabled us to structure the data properly with the help of collections and documents. The migration of database involved in setting up new configurations, code change in all the sensor modules.
- ii. **Limited number of requests** – The Firebase offers a quota of 50000 read operations, 20000 write operations and 20000 delete operation requests per day. Although it might

seem sufficient, the read operations might exceed the designated limit due to large dataset recorded. There were few instances where-in the entire dataset was read by the web application, which resulted in temporary blockage of dataset display in Firebase website. However, the data will still be recorded in real-time but won't be displayed. The dataset can be accessed the next day. In order to prevent this, the read operations must be carefully handled.

#### d. Things to remember

- i. The private key JSON file in the raspberry pi must be properly defined in the firebase program. Else the connectivity won't be established.
- ii. The credentials must be properly provided in the firebase program so that it ensures the real-time data transmitted to the database.
- iii. Read operations must be carefully handled so as to prevent blockage of access to datasets.

## 2. Scheduling and Logging

The main program '**start\_sensors.py**' integrates all sensor module codes by importing individual sensors as functions into the main program. It also facilitates the creation of scheduled background jobs for each sensor programs. Scheduling and logging functionalities have been incorporated in this main program.

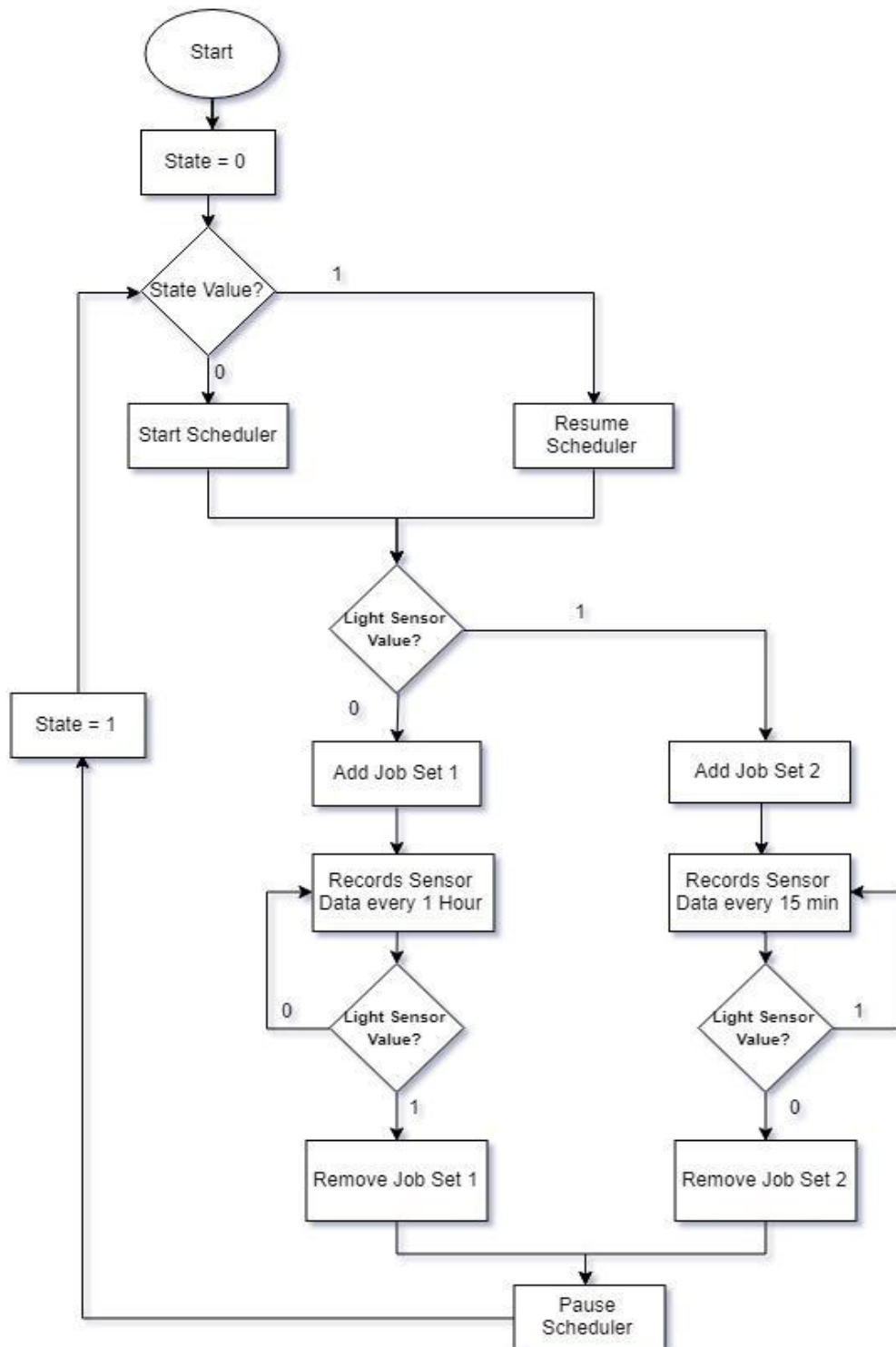
#### a. Scheduling

The scheduling is done to ensure that the individual sensor codes run at a scheduled interval of time. The scheduling is handled by python's Advanced Python Scheduler (APScheduler) library.

APScheduler lets us schedule the Python code to be executed later, either just once or periodically. We can add new jobs or remove old ones on the fly. We have used Interval-based execution which runs jobs on even intervals. The jobs which we added to the Background Scheduler runs in the background [18].

The main condition which defines the program flow is the light sensor's value. The light sensor is kept in the sensor box surface, but near the camera below the see-through opening covered by glass. On the sensor box, the beehive is kept. The light sensor indicates whether the beehive is on the sensor box or not. If Light sensor yields value '**0**' then it means that there is no light entering the light sensor which is due to beehive kept which is kept on the sensor box. When the beehive is on the sensor box, as per the requirement, the sensors must record the data every **1 hour**. The other condition is when the beehive is **not** on the sensor box. Now, the light sensor yields value '**1**', indicating that there is no beehive on the sensor box. For this, the schedule defined is that the sensor data must be recorded every **15 min**. To facilitate this, we created two job sets, the 'Job Set 1' has a schedule of 1 hour for each sensors pertaining to beehive closed condition, the 'Job Set 2' has a

schedule of 15 min for each sensors pertaining to beehive open condition. Fig 38 flow chart explains the scheduling process:



**Fig 38: Flow Chart of Main Program Workflow**

### Things to remember –

Though the APScheduler supports concurrent execution, it is always a good practice to specify ‘jitter’ parameter in the scheduler of each sensor. The jitter option enables us to add a random component to the execution time. This is helpful in our case so as to prevent multiple jobs with similar options from always running concurrently. We have used a jitter of 60 seconds. So, the jobs are triggered randomly in this 60 seconds jitter time for every 1 hour or 15 min schedule.

### b. Logging

Logging is a means of tracking events that happen when some software runs. The software’s developer adds logging calls to their code to indicate that certain events have occurred. The logging module of Python provides a lot of functionality and flexibility. Out of several classes in logging, we have employed ‘Handlers’ in order to send the log records (created by loggers) to the appropriate destination [19].

Level of logger is set to INFO, this logs a message with level INFO on this logger. Information regarding job trigger, execution and completion status will be recorded. Other logging levels are - Critical, Error, Warning, Debug. The format of records in the log file includes timestamp in ‘asctime’ format, followed by ‘Levelname’ which indicates the name of the logging level (in this case 'INFO') or 'ERROR' if any. This is followed by, ‘messages’ which indicates ‘Job status’ in terms of whether the job is executed successfully or not, error messages and next scheduled run information is also provided in messages segment. The log file is stored as ‘.txt’ file in the designated path mentioned. This file gives a clear perspective on the behaviour of all sensor modules including the main program. Any error will be logged in this log file which helps us to debug the code.

## 3. Adding the main program to boot file

To enable the program to run continuously as soon as the raspberry pi is powered on, the ‘start\_sensors.py’ script should be added in **rc.local** boot file. Below are the steps to do so –

- i. In the terminal enter the command ``sudo nano /etc/rc.local``
- ii. Enter the command ``sudo python3 /home/pi/PlanBee/start_sensors.py &`` in-between 'fi' and 'exit0' lines.
- iii. Press 'Ctrl+O' (to write the lines)
- iv. Press 'Ctrl+X' (to exit)
- v. Enter ``sudo reboot`` (to restart the system)
- vi. Once the system restarts, the script is executed automatically and the sensor data is uploaded without manually running the code!

**Things to remember –**

- i. Do not forget to include '&' symbol at the end of the script as it might affect the boot sequence. '&' symbol ensures that even if the script added gets struck, the remaining boot sequence will execute. This should be mainly done for the scripts which run infinitely, as in our case.
- ii. Make sure to reference absolute file names rather than relative to the home folder.
- iii. Be highly cautious as to which code you are trying to run at boot and test the code a couple of times as the code is added in the boot sequence.



## VI. Installation steps (*Team*)

This section explains the end-to-end procedure for installation. It provides a quick walkthrough from the installation of sensors, configuring interfacing options, installation of dependencies related to python, to enabling boot execution.

1. Connect the raspberry pi with the respective sensors as per the circuit diagram (Fig 2) provided in System Architecture.
2. Enable the following interfacing options in Raspberry Pi by entering ``sudo raspi-config`` in the terminal -> select 'Interfacing Options':
  - a. Camera
  - b. SSH
  - c. SPI
  - d. I2C
  - e. Serial
  - f. Remote GPIO
3. Enable Audio settings for USB Microphone as explained in the USB Mic section
4. Download or Clone the repository to `'/home/pi/PlanBee/'`
5. Install all python library dependencies at once by running the ``pip3 install -r requirements.txt`` command.

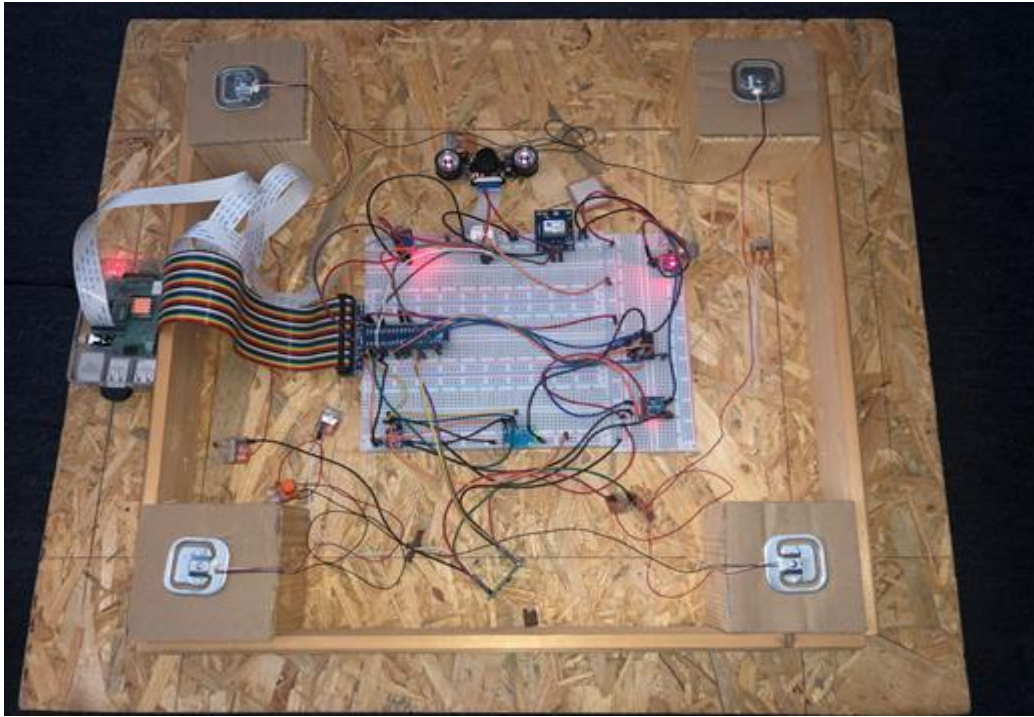
**Note:** `'requirements.txt'` file can be generated by `'pipreqs'` python library –

- Install the `'pipreqs'` library using `'sudo pip3 install pipreqs'`
  - In terminal execute command `'pipreqs <Path of the project folder>'`
  - `'requirements.txt'` file gets generated in the project path.
6. Manual run: Run the `start_sensors.py` file manually from the local path using the command ``sudo python3 start_sensors.py`` in the terminal to record and upload data to the database by all sensors. Or, automatically enable the program run using Step 7.
  7. Add the script in the boot file `rc.local` as explained in the Integration section 'Adding the main program in the boot file'.

**Note:** The code is written for Python 3.

Now, the Raspberry Pi connected with all the sensors is ready with the program. The user just needs to turn on the power supply of the Raspberry Pi. After turning on the Raspberry Pi, we could observe that the data from various sensors are recorded at scheduled intervals and sent to the Firebase database in real time. This ensures that the installation and deployment process is complete. If there are any issues, it can be easily traced with the help of log file generated.

## VII. Results (Team)



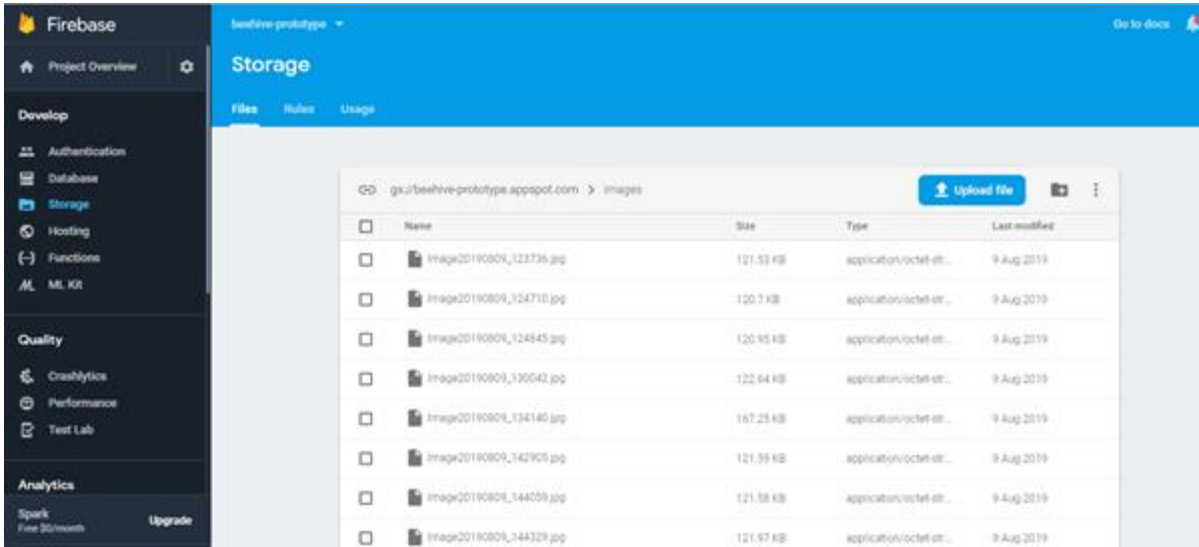
**Fig 39: Bread board connection of the prototype**

The above figure is the breadboard prototype we developed for the project Plan Bee.

| beehive-prototype  |   |   |
|--|---|---|
| Database   |   |   |
| Data   | Rules   | Indexes   |
| <div> <div>BME280</div> <div>20190809_124600</div> </div>  |   |   |
| <div> <div>beehive-prototype</div> <div>+ Start collection</div> </div>  | <div> <div>BME280</div> <div>+ Add document</div> </div>  | <div> <div>20190809_124600</div> <div>+ Start collection</div> </div>   |
| <div> <div>BME280</div> <div>DHT22</div> <div>Load_Cells</div> <div>MPU_6850</div> <div>MQ135</div> <div>NEO_6M</div> <div>USB_MLc</div> <div>kxmobilie</div> </div> | <div> <div>20190809_124600</div> <div>20190809_124625</div> <div>20190809_124650</div> <div>20190809_124715</div> <div>20190809_124740</div> <div>20190809_124805</div> <div>20190809_124830</div> <div>20190809_124855</div> <div>20190809_124921</div> <div>20190809_125728</div> <div>20190809_125841</div> </div> | <div> <div>+ Add field</div> <div>Outside_humidity: 46.66185071008953</div> <div>Outside_pressure: 978.5834300559609</div> <div>Outside_temp: 26.414107622025767</div> <div>timestamp: "20190809_124600"</div> </div> |

**Fig 40: Data collection of all the sensor in the database**

As seen in the Fig 40 for all the sensors each collection is created and respective sensor data are stored. All the sensor data except Night vision camera the data are sent as a json string.



**Fig 41: Image storage in the database.**

All the sensor data are stored by creating collection whereas the Images captured from the Night vision camera are transferred as blob so stored in the Storage Bucket of the Firebase database.

## Important Links:

- **Database:**
  - Test: <https://console.firebase.google.com/u/0/project/raspi-acd09/database>
  - Production: <https://console.firebase.google.com/u/0/project/beehive-prototype/database>
- **Gitlab:** <https://gitlab.com/planbee-project/raspberrypi>

## VIII. System Level Challenges and Solutions (*Team*)

Along with the challenges faced at sensor level as explained in the above section, there are few challenges that we faced at system level. Below table explains such challenges and the respective solutions

| Nr | Challenge                       | Challenge Description  | Solution  |
|----|---------------------------------|--|---|
| 1  | <b>Hardware connections</b>     | The system encompasses several hardware connections which include rigging of various sensor components to Raspberry Pi. These connections are quite complex in nature and requires the utmost care while connecting. | Checking connectivity thoroughly using multimeter and soldering of sensors ensured that there are no loose connections.   |
| 2  | <b>Setting up of Interfaces</b> | Several sensors are connected to the Raspberry Pi through serial interfaces such as I2C, UART and SPI. Interfaces are complex compared to normal GPIO connections.   | Modification of config files which are customized according to sensor requirements helped us achieve the interfacing task.  |
| 3  | <b>Reading Analog data</b>      | Few sensors give analog data and requires conversion to digital formats. As Raspberry Pi is compatible with digital signals, it is a challenging task.   | We employed Analog-Digital Converters for this.   |
| 4  | <b>Power distribution</b>       | Raspberry Pi has two 3.3V and two 5V power pins. Few sensors don't work when connected to the same 3.3V pin or same 5V pin.  | Distributing sensors to separate 3.3 V or 5V pins based on the power consumption of the sensor helped us attain proper power distribution across all the sensors. |

## **IX. Limitation** (*Team*)

There are some limitations of this project:

1. We are using regular power supply so we will need continuous powered connection near the beehive meaning it cannot be set up in some remote area.
2. Using WiFi we are sending all the data from sensors to the firebase platform. This project has a dependency on continuous internet connection.
3. We have used bread-board and jumper wires to connect the sensors with Raspberry Pi. The jumper wire connection is not intact and reliable so sometimes sensors data might not be recorded.
4. We are using economic sensors, so some of the sensors' reliability is not good enough.

## **X. Future Scope (*Team*)**

1. Instead of normal power supply if we use a battery-operated power supply or solar power supply then we can use the device at any remote place.
2. Instead of the WiFi, we can use LoRa to transmit sensor data from raspberry Pi to the gateway. Using LoRa we can send data up to tens of miles without the internet.
3. Now instead of bread-board or matrix board, we can solder the sensors on PCB, which will make the connection intact and reliable, and also will make the prototype compact.
4. Incorporate additional sensors to find out the presence of any foreign insects and implement an appropriate combination of sensors to detect the approximate number of bees inside the beehive.
5. To devise an automatic honey collector after certain amount of honey is produced.

## XI. Extended work (*Team*)

In addition to the prescribed milestone to be achieved at the end of the project our client i.e. PlanBee requested us to come up with more robust hardware design which can be installed inside the beehive for a longer duration of at least 6 months. In order to fulfill their demands our team worked to build a second prototype and this time the entire arrangement of sensors was made on a matrix board. The matrix board designed as of now has a very good advantage, as the sensors can be just plugged in the slots provided for it. The internal connections are soldered underneath matrix board. In case of any malfunction of a particular sensor then that sensor can be easily plugged out of the slot and replaced with the new one. The connections at the backend will not be disturbed. Here is the image shown below will provide an understanding of the prototype.



**Fig 42: Prototype designed using Matrix board**



**Fig 43: Sensor Box containing our prototype being installed in Beehive**

Fig 43 shown is the installation of the prototype board in one of the beehives located in Passau. The model will be monitored by the Southern Bayern Bee Association along with the PlanBee team for next 6 months. We have even received positive feedback from the client that the data recording is working as expected.

#### **Placement of sensors**

- a. Inside sensor box: Light sensor, Gyroscope, GPS sensor, Load cell NV camera.
- b. Inside beehive: DHT22, Gas sensor, Sound sensor
- c. Outside the beehive: BME280



## **XII. Conclusion** (*Team*)

As explained in the introduction regarding the difficulty in beekeeping using conventional practices, our project Automated Beehive will overcome all those limitations. This project will also help in reducing the manpower and time consumption required to continuously monitor the beehive condition. Amateurs and beginners can be benefited as they lack the essential knowledge required for beekeeping. Customers who consider beekeeping as their hobby can remotely monitor the beehive by looking into the data available online. The project encompasses of the mentioned tasks and technologies setting up the working platform, raspberry pi architecture, communication protocols, hardware connections, python programming, and understanding the behavior of bees and linking them to the physical parameter, sensing of the parameters using various sensors, configuration and troubleshooting of the same.

## References

- [1] Carbon Dioxide Sensing: Fundamentals, Principles, and Applications by Gerald Gerlach, Ulrich Guth, Wolfram Oelßner pg 379 - 381.
- [2] Influence of Temperature and Humidity on the Output Resistance Ratio of the MQ-135 Sensor Vandana Kalra, Chaitanya Baweja, Dr. Simmarpreet, Supriya Chopra.
- [3] 2.7V 4-Channel/8-Channel 10-Bit A/D Converters with SPI Serial Interface Datasheet <https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf> .
- [4] <https://www.pololu.com/product/2595>
- [5] <https://annemariemaes.net/humidity-temperature-measurements/>
- [6] <https://lastminuteengineers.com/bme280-arduino-tutorial/>
- [7] <https://www.instructables.com/id/Arduino-Bathroom-Scale-With-50-Kg-Load-Cells-and-H/>
- [8] <https://www.arnia.co.uk/humidity-in-the-hive/>
- [9] <https://allisonsapiaries.com/ideal-beehive-temperature-bees-honey/>
- [10] <https://www.espruino.com/DHT22>
- [11] <https://www.electronicwings.com/sensors-modules/gps-receiver-module>
- [12] <http://www.uugear.com/portfolio/using-light-sensor-module-with-raspberry-pi/>
- [13] <https://americanbeejournal.com/sounds-of-the-hive-part-1/>
- [14] <https://makersportal.com/blog/2018/8/23/recording-audio-on-the-raspberry-pi-with-python-and-a-usb-microphone>
- [15] <https://www.electronicwings.com/raspberry-pi/mpu6050-accelerometergyroscope-interfacing-with-raspberry-pi/>
- [16] <https://thepihut.com/products/raspberry-pi-night-vision-camera>
- [17] <https://opensource.google.com/projects/firebase-sdk>
- [18] <https://apscheduler.readthedocs.io/en/latest/>
- [19] <https://docs.python.org/3/library/logging.html>