Transactions on Computer Systems and Networks

Jivan S. Parab ·
Madhusudan Ganuji Lanjewar ·
Marlon Darius Sequeira · Gourish Naik ·
Arman Yusuf Shaikh

Python Programming Recipes for IoT Applications

Springer

Transactions on Computer Systems and Networks

Series Editor

Amlan Chakrabarti, Director and Professor, A. K. Choudhury School of Information Technology, Kolkata, West Bengal, India

Jivan S. Parab · Madhusudan Ganuji Lanjewar ·
Marlon Darius Sequeira · Gourish Naik ·
Arman Yusuf Shaikh

Python Programming Recipes for IoT Applications

Springer

Jivan S. Parab
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Marlon Darius Sequeira
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Arman Yusuf Shaikh
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Madhusudan Ganuji Lanjewar
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

Gourish Naik
School of Physical and Applied Sciences
Goa University
Taleigao, Goa, India

ISSN 2730-7484 ISSN 2730-7492 (electronic)
Transactions on Computer Systems and Networks
ISBN 978-981-19-9465-4 ISBN 978-981-19-9466-1 (eBook)
https://doi.org/10.1007/978-981-19-9466-1

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Contents

Chapter 4
MicroPython PyBoard for IoT

Abstract In this chapter, readers will learn how to program Pyboard for IoT applications using MicroPython. MicroPython is a member of the Python interpreter's family. MicroPython Programming language is a subset of the Python to fit and execute on a microcontroller. It has several optimizations to ensure that it works effectively and consumes little RAM. Hence MicroPython is perfect for embedded systems and IoT applications. Here, authors have focused on hands on implementation of few simple IoT applications such as home automation, smart e-waste bin, industrial environmental monitoring, green house monitoring, and aquaculture monitoring. Before implementing the IoT applications, one has to learn how to Install, run and debug the Micropython using PyCharm IDE.

Keywords Home automation · Environmental monitoring · Green house · Aquaculture · e-waste bin

MicroPython Installation in PyCharm IDE

PyCharm IDE supports Pyboard, ESP8266, and BBC Micro:bit microcontroller devices. In order to use Pyboard, one has to install MicroPython plugin in PyCharm. The detailed installation and setup process of the MicroPython plugin in PyCharm is as follows:

Step1: Download and install PyCharm IDE from the link: https://www.jetbrains.com/pycharm/

Step 2: To install the MicroPython plugin in PyCharm: open PyCharm IDE → *File* → *Settings* → *Plugins* → Enter 'MicroPython' in search window → click on 'Install' button to install MicroPython plugin.

Step 3: Create Python project: Go to *File* → *New Project (e.g TestIoT)* and then click on 'Create' button.

Step 4: Setting project structure: Go to *File* → *Settings* → *Project: TestIoT* → *Project structure*. Then right-click on '.Idea' and 'venv' and select excluded.

J. S. Parab et al., *Python Programming Recipes for IoT Applications*, Transactions on Computer Systems and Networks, https://doi.org/10.1007/978-981-19-9466-1_4

Step 5: To enable MicroPython support and path selection: Go to *File* → *Settings* → *Languages & Frameworks* → *MicroPython* and select device type as Pyboard. Tick the Check boxes '*Enable Micropython support'* and '*Auto detect device path'*. Device path can be also given manually and then click on 'detect' button and then click on 'OK'.

Step 6: Restart the PyCharm IDE and right-click on project name (TestIoT) → New → Python file → Enter the name to Python file (i.e main.py).

Step 7: After naming the file one has to compulsorily install missing packages as prompted by the IDE.

Step 8: Setting Python Read Evaluate Print Loop (REPL): click on Tools → MicroPython → MicroPython REPL. This step helps to show errors and allows to run MicroPython shell on Pyboard device.

Step 9: Flash Files to Pyboard: On taskbar, go to Run → Click on 'Flash main.py'.

Testing Simple code to Turn on LED of Pyboard:

After the successful installation of PyCharm and MicroPython plugin, the IDE is ready.

To test the simple code to turn on Pyboard LED, enter and flash the code given below:

```
import pyb
pyb.LED(1).on()
pyb.LED(2).on()
pyb.LED(3).on()
pyb.LED(4).on()
```

4.1 Home Automation

Nowadays, home automation has become more popular, and smart systems are being implemented practically in every home. Home automation, often referred to as "Smart Home Technology", which uses technology to automate your home. Home automation allows you to control almost every aspect of your home through the IoT. The concept of a smart home has gained popularity in recent years as technology made life easier (Majeed et al. 2020). Almost everything has gone digital and automated (Stolojescu-Crisan et al. 2021). The IoT concept, conceptualizes the idea of remotely connecting and monitoring real-world items (things) over the internet (Madhesh et al. 2020). One of the most touted benefits of home automation is providing peace of mind to homeowners, allowing them to monitor/control their homes remotely, countering dangers such as forgetting to turn off geyser or a front door left unlocked or lights/fan left on. Figure 4.1 shows the basic smart home system.

Fig. 4.1 Smart home systems

Here, the authors implemented home automation system with two approaches, first one with Pyboard and second one with ESP32 as IoT device, using Blynk app. The reason for using ESP32 is that Pyboard is not included in Blynk app to remotely control the appliances. Here, four electrical appliances such as fan, refrigerator, lights, and air condition (AC) are controlled using IoT device. This system has ESP32 configured as IoT device to remotely control the above appliances. The electrical appliances in the house are connected through relay module to Pyboard/ESP32. The OLED display interface is used to show the status of appliances connected to the Pyboard/ESP32.

Here, a 0.96-inch OLED SSD1306 display is used. The OLED display will communicate with the Pyboard/ESP 32 via the I2C protocol.

First Approach:

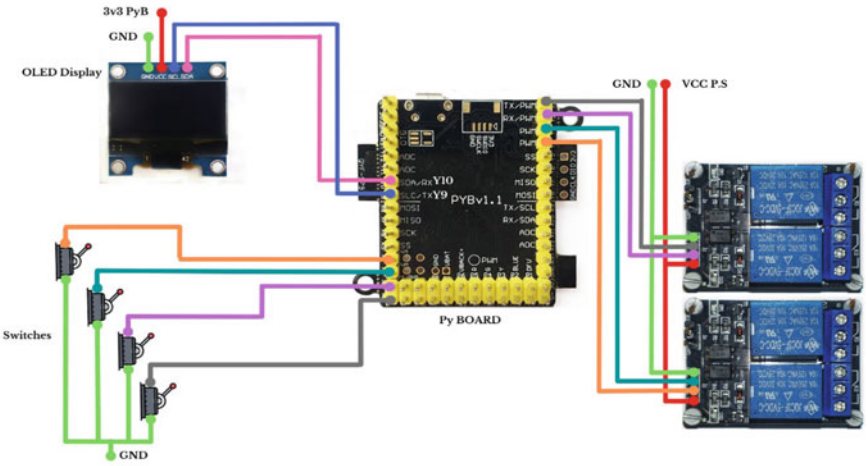
Home Appliances Control with Pyboard:

The four electrical appliances (fan, AC, refrigerator, and light) will be controlled by Pyboard through the relay as shown in Fig. 4.2a. Figure 4.2b shows the entire flowchart for controlling the four appliances. Table 4.1 tabulates the connection of Pyboard with switches, OLED, and relays.

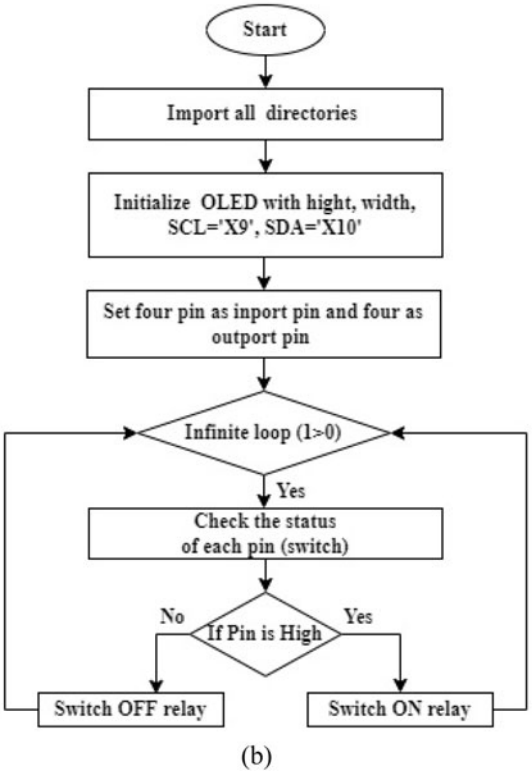
Implementation and Configuration Steps:

Step 1: Copy MicroPython OLED ssd1306 library from the following link: https://github.com/micropython/micropython/blob/master/drivers/display/ssd1306.py

Step 2: Open PyCharm IDE → paste the code in PyCharm IDE and save it as ssd1306.py.



(a)



(b)

Fig. 4.2 **a** Circuit diagram of home automation and **b** Flowchart for controlling four devices

Table 4.1 Pyboard, switches, and relay pins connections

Pyboard	Switches	Pyboard	Relay	Pyboard	OLED
X1	SW1	Y1	R1	3.3 V	Vin
X2	SW2	Y2	R2	GND	GND
X3	SW3	Y3	R3	X9	SCL
X4	SW4	Y4	R4	X10	SDA

Step 3: Make sure the connections are according to Fig. 4.2 and enter the following code in 'main.py':

```
# Program to control home appliances
import time
import ssd1306
import machine
from pyb import Pin
from micropython import const
width = const (128)
height= const (64)
ssd1306_scl= Pin('X9', Pin.OUT_PP)
ssd1306_sda= Pin('X10', Pin.OUT_PP)
i2c_ssd1306=machine.I2C(scl=ssd1306_scl, sda=ssd1306_sda)
oled = ssd1306.SSD1306_I2C(width, height, i2c_ssd1306)
oled.fill(0)
p_in1 = Pin('X1',Pin.IN, Pin.PULL_UP)
p_in2 = Pin('X2', Pin.IN, Pin.PULL_UP)
p_in3 = Pin('X3', Pin.IN, Pin.PULL_UP)
p_in4 = Pin('X4', Pin.IN, Pin.PULL_UP)
p_out1 = Pin('Y1', Pin.OUT, Pin.OUT_PP )
p_out2 = Pin('Y2', Pin.OUT_PP)
p_out3 = Pin('Y3', Pin.OUT_PP)
p_out4 = Pin('Y4', Pin.OUT_PP)
while (1):
    oled.fill(0)
    if p_in1.value() == False:
        p_out1.high()
        oled.text('AC-ON', 0, 0)
        oled.show()
    else:
        p_out1.low()
        oled.text('AC-OFF', 0, 0)
        oled.show()
    if p_in2.value() == False:
        p_out2.high()
        oled.text('Fan-ON', 0, 10)
        oled.show()
    else:
        p_out2.low()
        oled.text('Fan-OFF', 0, 10)
        oled.show()
    if p_in3.value() == False:
        p_out3.high()
        oled.text('Refrigerator-ON', 0, 20)
        oled.show()
```

```
        else:
            p_out3.low()
            oled.text('Refrigerator-OFF', 0, 20)
            oled.show()
        if p_in4.value() == False:
            p_out4.high()
            oled.text('Light-ON', 0, 30)
            oled.show()
        else:
            p_out4.low()
            oled.text('Light-OFF', 0, 30)
            oled.show()
        time.sleep(0.5)
```

This is just a simple application to control the appliances using switches. To control appliance over internet, the authors have demonstrated the same by using ESP32.

Second Approach:

Home Appliances Control with ESP32 (IoT)

The four electrical appliances (fan, AC, refrigerator, and light) will be controlled by ESP32 through the relay as shown in Fig. 4.3. Table 4.2 tabulates the connection of ESP32 with relay.

To program ESP32, the Thonny IDE is used as it supports ESP32 board. The detailed steps for home automation implementation are as follows:

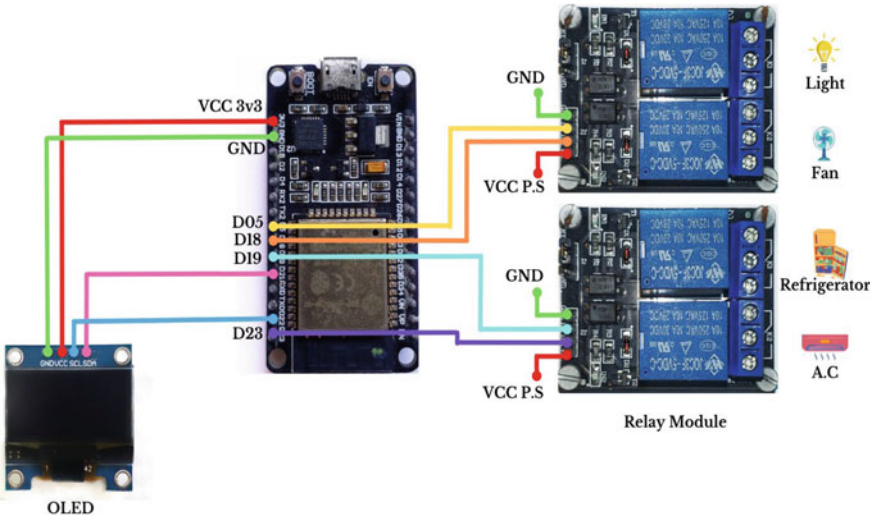


Fig. 4.3 Complete circuit diagram of home automation system

Table 4.2 ESP32 and relay pins connections

ESP32	Relay	ESP32	OLED
23	R1	3V3	VCC
19	R2	GND	GND
18	R3	D19	SDA
5	R4	D23	SCA

Step 1: Firmware installation using Thonny IDE

(1) Download the latest firmware from the link https://micropython.org/download/esp32/
(2) Installing MicroPython Firmware onto ESP32: Open Thonny IDE → Tools → options → interpreter → select MicroPython (ESP32) → select COM port → click on 'install and update firmware' → select port and firmware path (downloaded MicroPython Firmware path) → then click on install.

Now, the ESP32 is ready to use for any application after firmware installation.

Step 2: Blynk app installation process

The ESP32 board is used to control appliances with Blynk app via WiFi. Blynk app supports both iOS and Android platforms to control ESP32, Arduino, and Raspberry Pi over the internet. It is a digital dashboard, where one can build a graphic interface for project by simply dragging and dropping widgets. Follow the below steps to set up the Blynk app on mobile.

(1) Download Blynk app (legacy) from Google Play Store or App Store and install it.
(2) Create an account and log in.
(3) Create a new project and name it with a suitable name followed by selecting ESP32 device.
(4) Click on 'Create' button. You will receive an authentication key on your registered email id.
(5) Then, tap anywhere on the canvas to open the widget box. All the available widgets are located here. From the available options choose a 'button'.
(6) In this application, create four buttons to control four appliances through Blynk app. Tap on the widget to change the setting. Select the PIN → Digital → gp21. Choose button mode as 'switch'. Continue this step to create other buttons such as gp19, gp18, and gp5.
(7) Now, the Blynk app is ready. On pressing 'Play' button, it will switch from 'EDIT' mode to 'PLAY' mode where one can interact with the hardware.

Step 3: Controlling home appliances using Blynk

(1) Copy Blynk MicroPython library from https://github.com/lemariva/uPyBlynk/blob/master/BlynkLibESP32.py

(2) Edit the boot file (boot.py) of ESP32: File → open → Select 'MicroPython' → open 'boot.py' and paste the below code and save it.

```
# The boot.py file

ssid_ = "samsung"                    #Change your WiFi ssid
wp2_pass = "qwerty123"               #Change your WiFi password
def do_connect():
    import network
    sta_if = network.WLAN(network.STA_IF)
    if not sta_if.isconnected():
        print('connecting to network...')
        sta_if.active(True)
        sta_if.connect(ssid_, wp2_pass)
        while not sta_if.isconnected():
            pass
    print('network config:', sta_if.ifconfig())
do_connect()
```

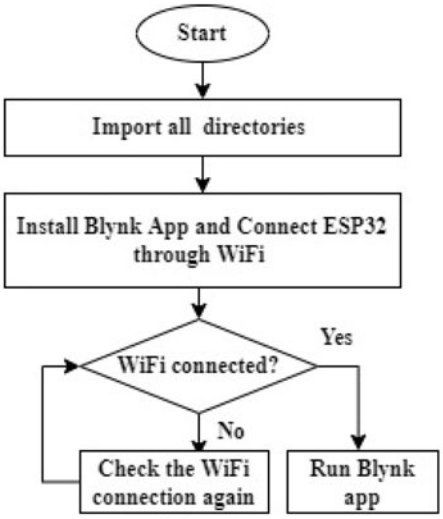
(3) Enter the below code in 'main.py' file with correct authentication token received over email.

```
from machine import Pin, SoftI2C
from time import sleep
import network
import utime as time
from machine import Pin
import BlynkLibESP32 as BlynkLib   # for ESP32
# blynk = BlynkLib.Blynk("Auth token")
blynk = BlynkLib.Blynk("6GKWghlS8rPx07pQBVox63EcNif6klBA")
blynk.run()
```

The flowchart for home automation system using Blynk app is shown in Fig. 4.4a. Run the 'main.py' program and the obtained output is shown in Fig. 4.4b.

4.2 Smart e-waste Bin

In the present days, a rapid increase in urbanization and per capita income has led to an increase in municipal solid waste generation. Society creates an unhygienic environment for its citizens with respect to waste generation. This rapid generation of waste leads to various infectious diseases in the environment. A smart waste management (SWM) system ensures real-time monitoring of collection and transportation of waste. The SWM ensures that waste is collected on time and that the cost of entire operation is kept to a minimum (Zeb et al. 2019). To deal with various sorts of waste, including biological, industrial, and home waste, a variety of techniques are used (Rahman et al. 2020). Technologies such as global positioning system (GPS), radio frequency identification (RFID), global system for mobile communications (GSM), machine-to-machine (M2M) communication, and IoT, as well as innovative mobile



(a)

(b)

Fig. 4.4 **a** Flowchart for home automation system using Blynk app and **b** Output status of home appliance

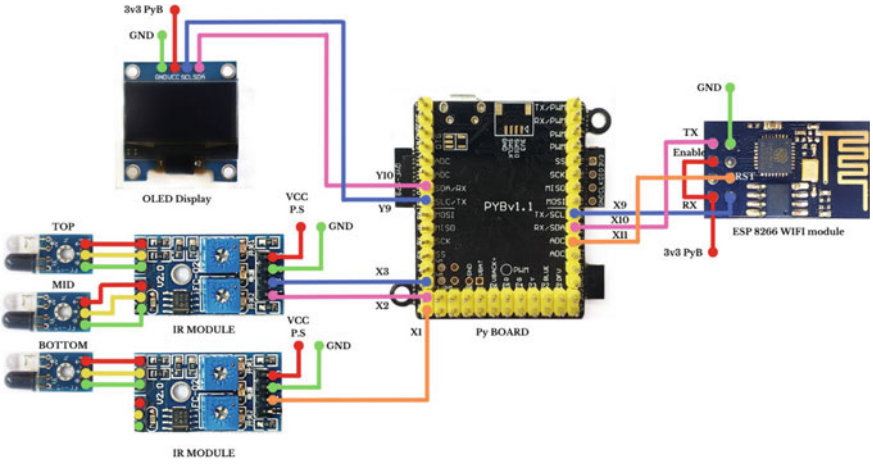


Fig. 4.5 Circuit diagram of smart e-waste bin

and web-based applications, can be used to improve and smoothen the ground-level mechanism for waste collection, processing, and recycling. Hence, the authors have implemented a smart e-waste bin using an IoT platform, which makes waste management convenient and very efficient. Here, Pyboard with ESP 8266 is configured as IoT node which updates the bin's current status, whether the bin is empty or full, on to the ThingSpeak cloud.

The smart bin system has a proximity sensor FC-45 for checking the status of bin, ESP8266 for WiFi connectivity, and OLED to display the bin status. All these modules are interfaced to Pyboard as shown in Fig. 4.5.

The proximity sensor uses an infrared (IR) transmitter and a receiver to detect an object. Here, three IR sensor modules are used to give the status of the bin. The output of these proximity sensors is connected to the Pyboard pins X1, X2, and X3. The bottom IR sensor is connected to the X1 pin, middle IR sensor to X2 pin, and top IR sensor to the X3 pin of Pyboard.

As Pyboard does not have WiFi feature, ESP8266 Serial WiFi Module is used for WiFi connectivity to upload the status of smart bin on ThingSpeak cloud. The ESP8266 WiFi Module is a self-contained SOC with an integrated TCP/IP protocol stack that allows any microcontroller to connect to a WiFi network for accessing internet. The pin-out details for interfacing Pyboard with 8266 WiFi model are given in Table 4.3.

After successful hardware setup, the next step is configuration and code implementation.

Implementation and configuration:

Table 4.3 ESP8266 WiFi module pin description and pin connection with Pyboard

8266 Pin	Description	Pyboard pins
VCC	Power pin = 3.3v	3.3 V
GND	Ground	GND
Rx	Receive serial data from another device	Tx
Tx	Transfer serial data to other devices	Rx
CH_En	Chip enable pin, connected to 3.3 V	3.3 V
GPIO 0	General-purpose input–output pin used as a normal GPIO pin and also used to enable the ESP8266 programming mode	Not connected
GPIO 2	Used as a GPIO pin	Not connected

Step 1: Configuration of ThingSpeak Channel

To create channel on ThingSpeak, one has to first sign up on ThingSpeak (https://thingspeak.com/). In case one has an account on ThingSpeak, just sign in using your id and password. For signup, fill in your details and then verify with the received e-mail and proceed. After this, click on 'New Channel' button which will subsequently ask for the 'Name and Description' of the data you want to upload on this channel as shown in Fig. 4.6

(1) In this application, smart bin status is sent to ThingSpeak. Hence, authors have named the channel 'Smart Bin'. More than one field of data can be activated by checking the box next to Field option. The authors have created one field, namely 'Bin status' (Fig. 4.7). After this, click on 'save channel' button to save the details.

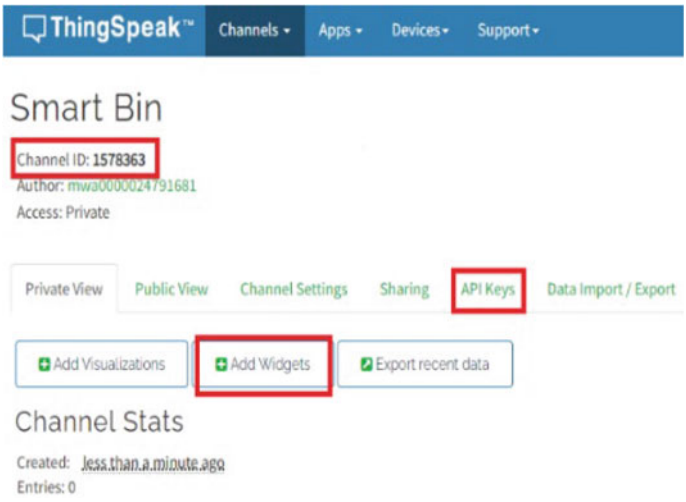


Fig. 4.6 Channel details for smart bin

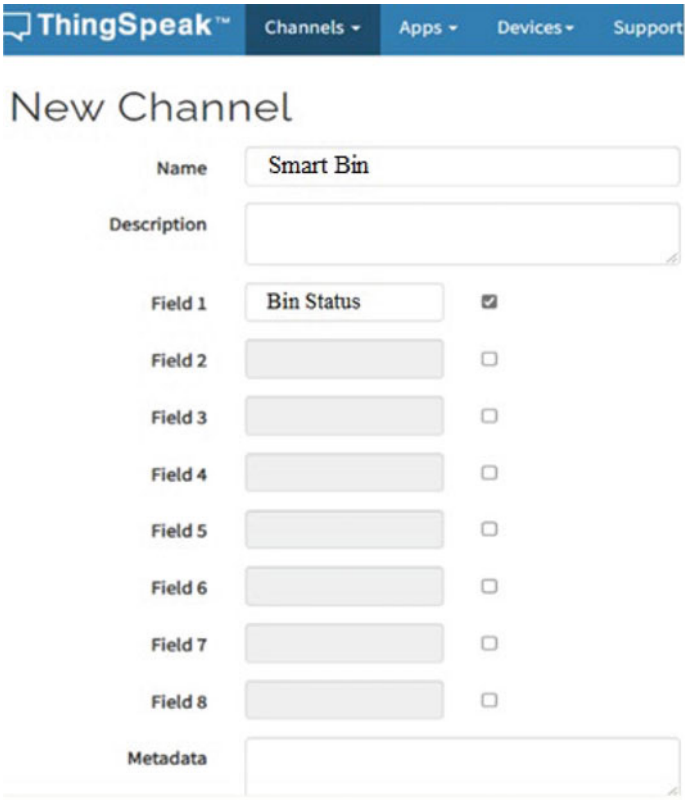


Fig. 4.7 Channel name and field details

(2) Add widget by clicking on 'Add widget' and select 'Gauge' → click on 'Next' → enter the information → Gauge created as shown in Fig. 4.8.

Step 2: Obtain the API Key

To send data to ThingSpeak, a unique API key is required, which is used in the main Python code to upload the status of bin to ThingSpeak server. Navigate to 'API Keys' (Fig. 4.6) header under the newly created above channel to get the unique API key as shown in Fig. 4.9.

Step 3: Configuring ESP8266 WiFi Wireless Module

Download Esp8266 WiFi library for Pyboard: Visit the website https://www.tinyosshop.com/wifi-skin-for-pyboard and click on 'Test code' to download the 'pywifi' library (Fig. 4.10a). After extracting the downloaded file, it shows two MicroPython files, i.e. '*main.py*' and '*pywifi.py*' as shown in Fig. 4.10a. Copy '*pywifi.py*' file and open PyCharm IDE → paste it in a PyCharm IDE by saving it with name '*pywifi.py*' in Pyboard.

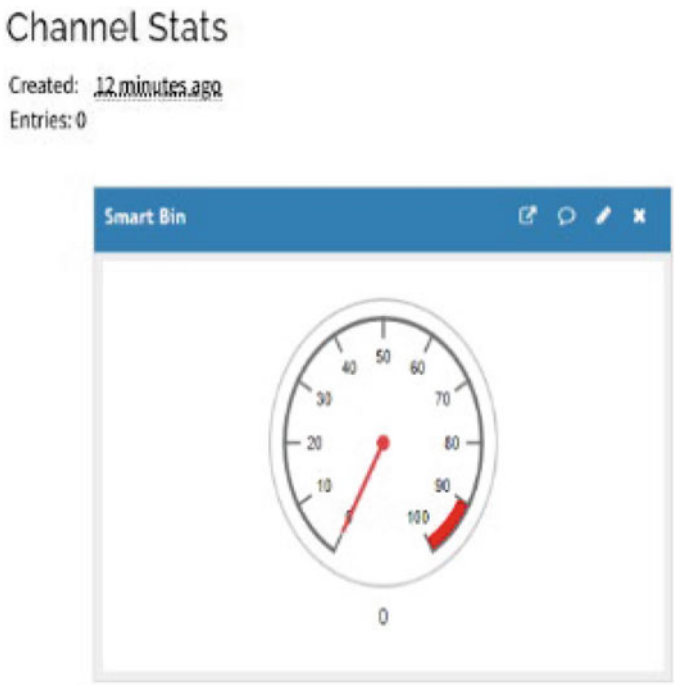


Fig. 4.8 Created widget

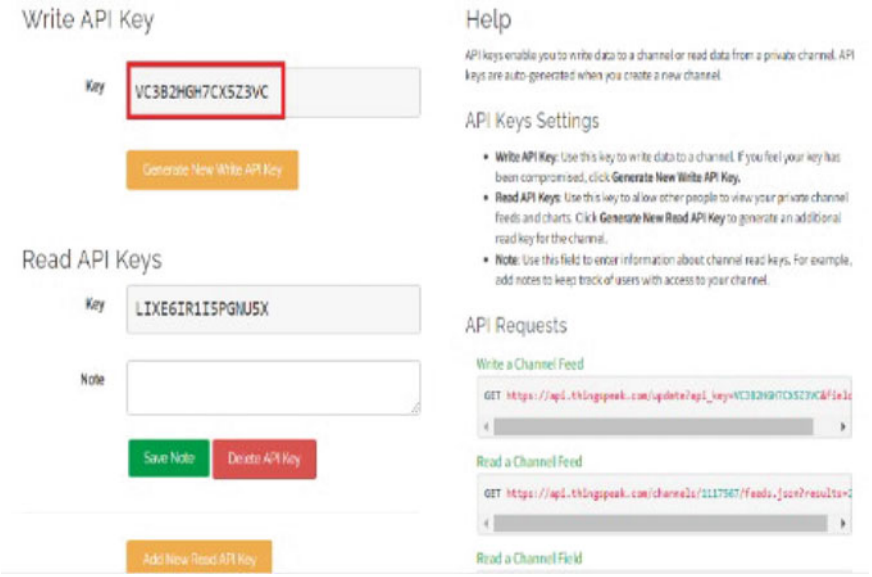


Fig. 4.9 ThingSpeak write key access

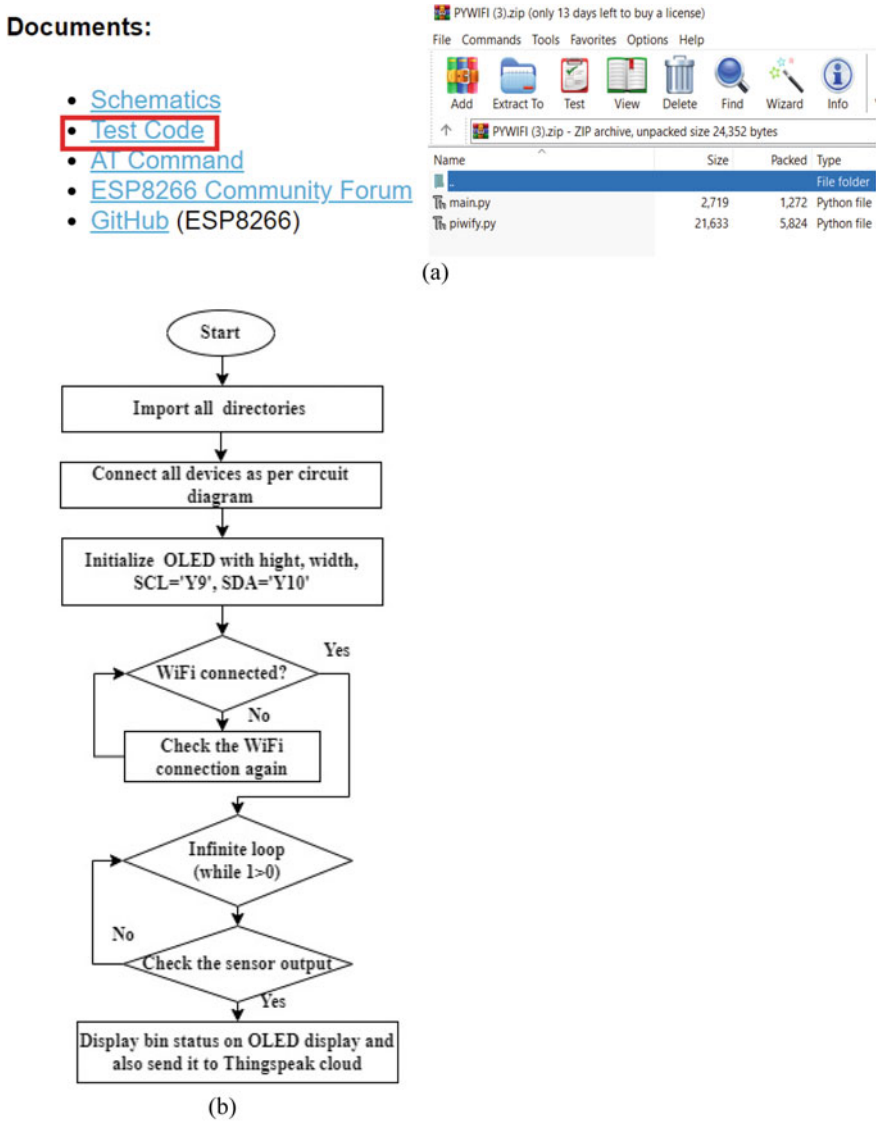


Fig. 4.10 **a** pywifi library details and **b** Flowchart for e-waste bin

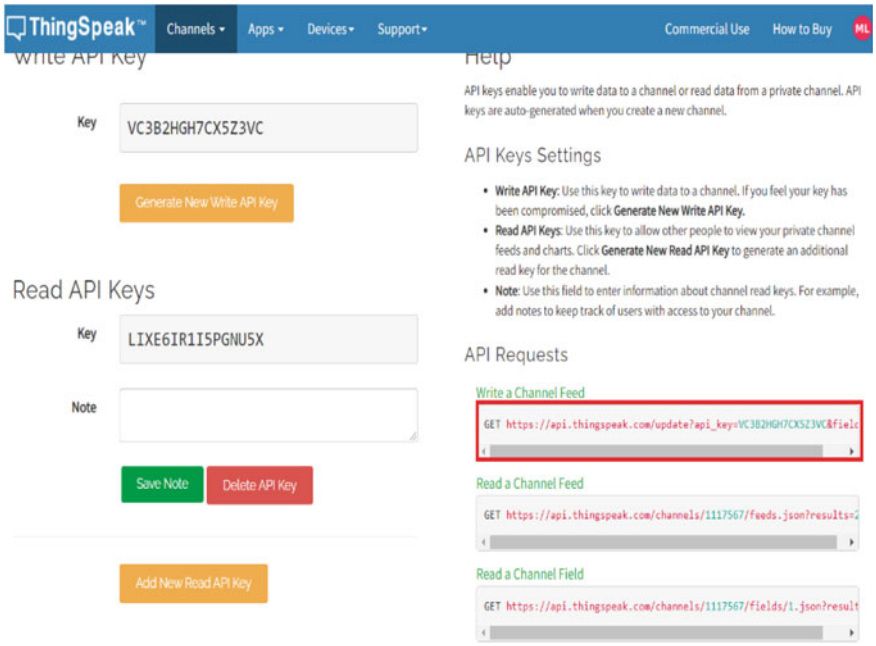


Fig. 4.11 ThingSpeak write channel feed access and channel status

Step 6: complete code (main.py)

Enter the below code in main.py (Fig. 4.11):

```
from pyb import Pin
import pyb
import ssd1306
import machine
from micropython import const
from machine import UART
import pywifi
width = const (128)
height= const (64)
ssd1306_scl= Pin('Y9', Pin.OUT_PP)
ssd1306_sda= Pin('Y10', Pin.OUT_PP)
i2c_ssd1306= machine.I2C(scl=ssd1306_scl, sda=ssd1306_sda)
oled = ssd1306.SSD1306_I2C(width, height, i2c_ssd1306)
```

```
while 1:
    rst_pyb = Pin('X11', Pin.OUT)
    rst_pyb.low()
    pyb.delay(20)
    rst_pyb.high()
    pyb.delay(500)

    Pyboard_wifi = pywifi.ESP8266(1, 115200)

    wifi_mode = 3

    Pyboard_wifi.set_mode(wifi_mode)

    pyb.delay(50)

    Pyboard_wifi.connect(ssid='AndroidAP93E9', psk='uxbm0411')

    pyb.delay(50)
    oled.fill(0)
    oled.text('WiFi Connected', 0, 0)
    oled.show()
    pyb.LED(4).on() #BLUE LED ON

    p_in1 = Pin('X1',Pin.IN, Pin.PULL_UP)
    p_in2 = Pin('X2', Pin.IN, Pin.PULL_UP)
    p_in3 = Pin('X3', Pin.IN, Pin.PULL_UP)
      # The dest_ip for Thingspeak website is 184.106.153.149

    Pyboard_wifi.start_connection(protocol='TCP',dest_ip='184.106.153.149',
                                  dest_port=80, debug=True)
    if p_in1.value() == True and p_in2.value() == True and p_in3.value() == True:
        oled.text('Bin is Empty', 0, 10)
        bin='0'
        oled.show()
    if p_in1.value() == False and p_in2.value() == True and p_in3.value() == True:
        oled.text('Bin is less than Half', 0, 10)
        bin='33'
        oled.show()
    if p_in1.value() == False and p_in2.value() == False and p_in3.value() == True:
        oled.text('Bin is Half', 0, 10)
        bin='66'
        oled.show()
    if p_in1.value() == False and p_in2.value() == False and p_in3.value() == False:
        oled.text('Bin is Full', 0, 10)
        bin='100'
        oled.show()

    #Copy write channel feed from Thingspeak website as shown in figure 4.11

    Pyboard_wifi.send('GET
    https://api.thingspeak.com/ update? api key=HKFRK1JCH5BMOCQ8&field1= '
    + str(bin) + ' HTTP/1.0\r\nHost:192.168.43.176\r\n\r\n', debug=True)

    pyb.delay(1000)
```

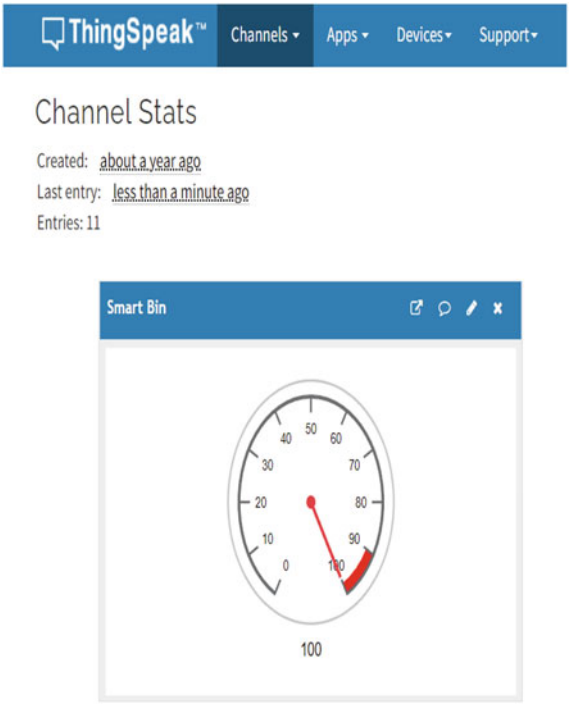


Fig. 4.12 Status of smart bin on ThingSpeak

After entering the above code, one has to run the '*main.py*' program and the status of the bin will be displayed on OLED display as well as on ThingSpeak cloud shown in Fig. 4.12.

4.3 Industrial Environmental Monitoring

Environmental monitoring is essential for protecting both human health and the ecosystem. Industries are the backbone of today's contemporary society, allowing for mass production of commodities to meet the needs of an ever-increasing population. Unfortunately, industrial pollution has a detrimental impact on the planet we live in. Hence, it's our responsibility to give our next generation a greener world for a better life. Governments around the globe have created norms and regulations to monitor the industries and to keep the pollution under check. Routine environmental monitoring data can also be used to validate and compare results about chemical behavior based on laboratory or field investigations (Artiola and Brusseau 2019).

The authors have implemented industrial environmental monitoring systems using IoT (Fig. 4.13). The designed system monitors industrial environment parameters such as temperature, humidity, butane, methane (CH_4), ammonia (NH_3), nitrogen dioxide (NO_2), and carbon monoxide (CO). The pin-out details for interfacing Pyboard with sensors are given in Table 4.4.

Configuration of ThingSpeak Channel

After creating the channel as shown in earlier Sect. 4.2, click on 'New Channel' button which will subsequently ask for the 'Name and Description' of the data you want to upload on this channel as shown in Fig. 4.14a.

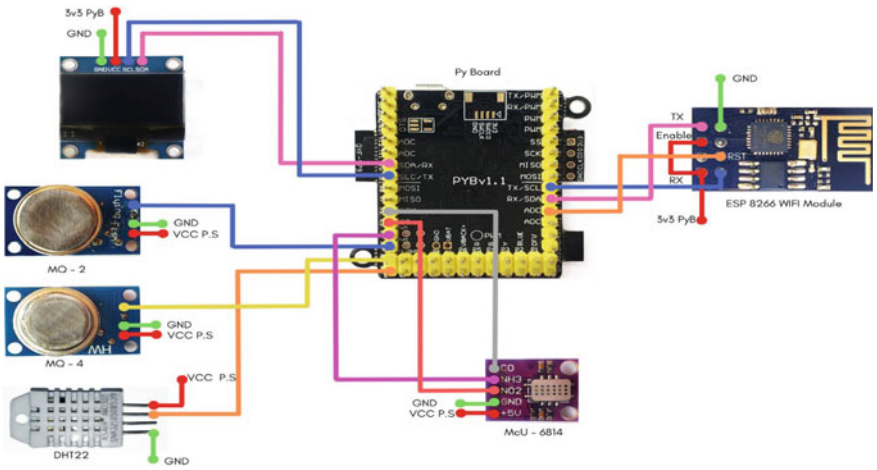


Fig. 4.13 Circuit diagram of industrial environmental monitoring

Table 4.4 Pin connection of Pyboard with various sensors

Pyboard	Sensors	Pin	Uses
X1	DHT22	data	To detect temperature and humidity
X2	MQ4	Analog out	To detect methane gas
X3	MQ2	Analog out	To detect butane
X4	CJMCU 6814	NH_3	To detect NH_3
X5	CJMCU 6814	NO_2	To detect NO_2
X6	CJMCU 6814	CO	To detect CO

In this application, the authors have named the channel 'Industrial Environmental Monitoring' and seven fields are created as shown in Fig. 4.14. After this, click on 'save channel' button to save the details.

Obtain the API Key

To send data to ThingSpeak, a unique API key is required, which is used in '*main.py*' code to upload parameters onto the ThingSpeak server. Navigate to 'API Keys' header under the newly created channel to get the unique API key. After completing these steps, the channel is ready to receive the parameters. Enter the below code in '*main.py*' and save it on Pyboard. The flowchart for industrial environmental monitoring system is shown in Fig. 4.14b.

(a)

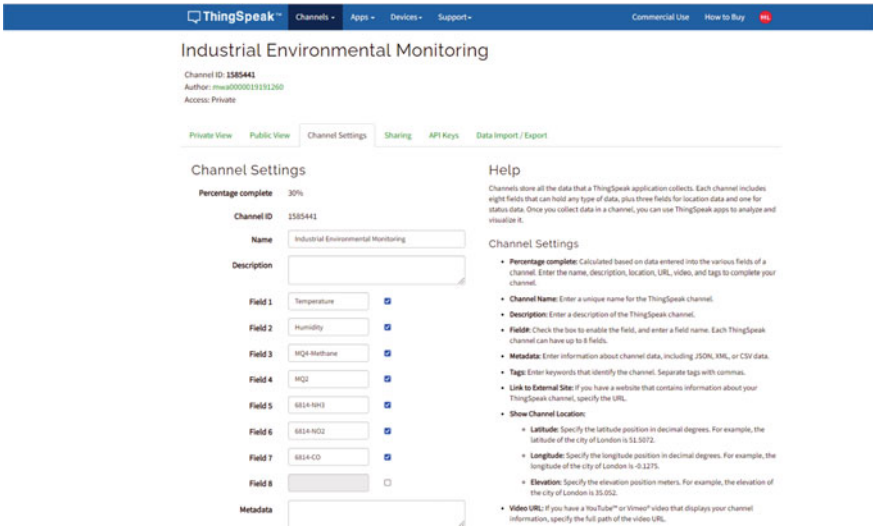


Fig. 4.14 **a** Channel settings and **b** Flowchart for Industrial Environmental Monitoring system

(b)

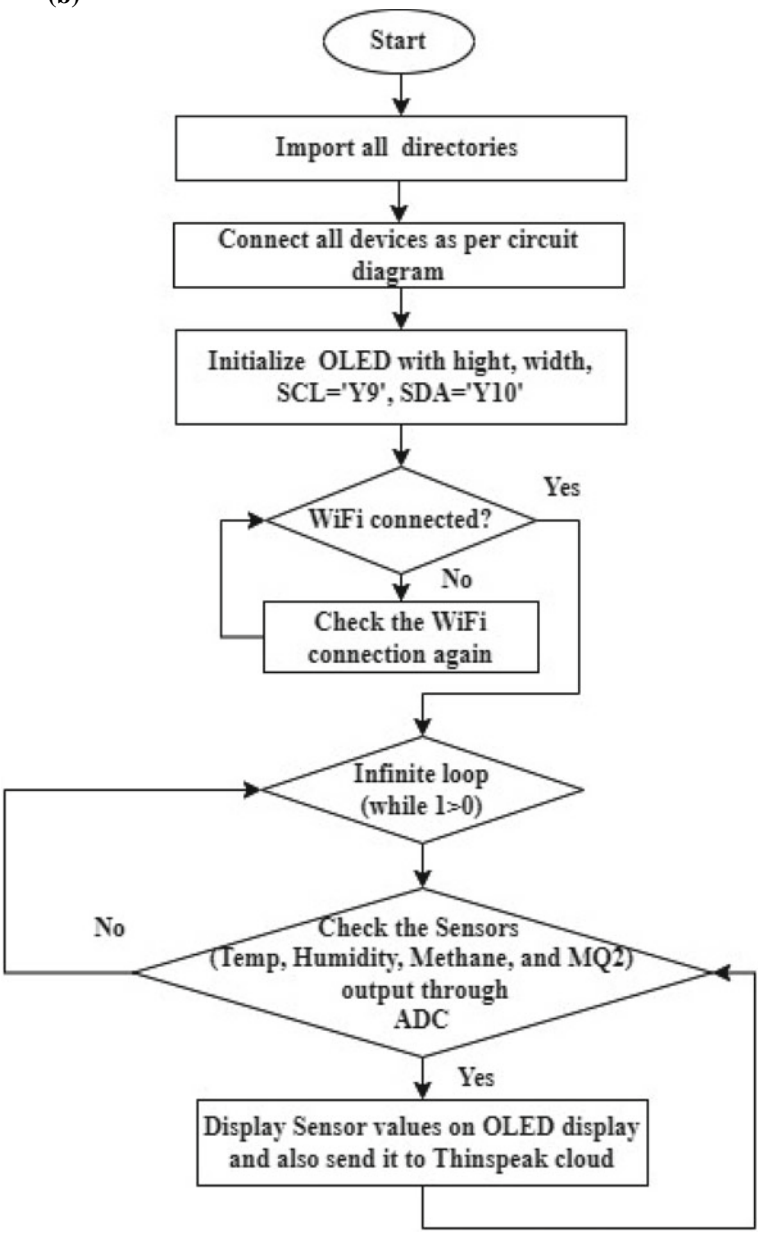


Fig. 4.14 (continued)

Complete code (main.py)

```
from pyb import Pin
import pyb
import ssd1306
import machine
from machine import Pin
from micropython import const
import dht
width = const (128)
height= const (64)
ssd1306_scl= Pin('Y9', Pin.OUT_PP)
ssd1306_sda= Pin('Y10', Pin.OUT_PP)
i2c_ssd1306=machine.I2C(scl=ssd1306_scl, sda=ssd1306_sda)
oled = ssd1306.SSD1306_I2C(width, height, i2c_ssd1306)
import pywifi
temp_hum = dht.DHT22(Pin('X1'))
while 1:
    rst_pyb = Pin('X11', Pin.OUT)
    rst_pyb.low()
    pyb.delay(20)
    rst_pyb.high()
    pyb.delay(500)
    Pyboard_wifi = pywifi.ESP8266(1, 115200)
    wifi_mode = 3
    Pyboard_wifi.set_mode(wifi_mode)
    pyb.delay(50)
    Pyboard_wifi.connect(ssid='AndroidAP93E9', psk='uxbm0411')
    pyb.delay(50)
    pyb.LED(4).on() #BLUE LED ON
    pyb.delay(2000)
    temp_hum.measure()
    temp = temp_hum.temperature()
    hum = temp_hum.humidity()
    MQ4 = pyb.ADC('X2')  # create an analog object for Methane
    Out1 = MQ4.read()  # read an analog value
    MQ2 = pyb.ADC('X3')  # create an analog object for Butane
    Out2 = MQ2.read()  # read an analog value
    NH3 = pyb.ADC('X4')  # create an analog object for Ammonia
    Out3 = NH3.read()  # read an analog value
    NO2 = pyb.ADC('X5')  # create an analog object for Nitrogen Dioxide
    Out4 = MQ2.read()  # read an analog value
    CO = pyb.ADC('X6')  # create an analog object for Carbon Monoxide
    Out5 = CO.read()  # read an analog value'''
    oled.text("Tem: " + str(temp), 0, 10)
    oled.text("Humidity:" + str(hum), 0, 20)
    oled.text("Methane:" + str(Out1), 0, 30)
    oled.text("MQ2:" + str(Out2), 0, 40)
    oled.show()
    oled.fill(0)
    pyb.delay(1000)
    oled.text("NH3:" + str(Out3), 0, 10)
    oled.text("NO2: " + str(Out4), 0, 20)
    oled.text("CO:" + str(Out5), 0, 30)
    oled.show()
    pyb.delay(1000)
    Pyboard_wifi.start_connection(protocol='TCP',dest_ip='184.106.153.149',
    dest_port=80,debug=True)
```

```
Pyboard_wifi.send('GET
https://api.thingspeak.com/update?api_key=7RZ93ORH43F2RYP7&field1=' +
str(temp)+'&field2='+ str(hum) +'&field3='+ str(Out1)+'&field4='+
str(Out2)+'&field5='+ str(Out3)+'&field6='+ str(Out4)+'&field7='+ str(Out5)+'
HTTP/1.0\r\nHost: 192.168.43.176\r\n\r\n', debug=True)
pyb.delay(1000)
```

After entering the above code, one has to execute the '*main.py*' program and the industrial environmental parameters will be displayed on OLED display as well as on ThingSpeak cloud as shown in Fig. 4.15.

Fig. 4.15 Industrial environmental parameters monitoring system on ThingSpeak

4.4 Greenhouse Monitoring

Flora such as flowers and vegetables are grown in a greenhouse. Greenhouses warm up during the day as sunlight passes through them, heating the plants, soil, and structure. Greenhouses protect crops from a variety of illnesses, notably these are soil-borne and splash onto plants in the rain. The greenhouse effect is a natural phenomenon that is advantageous to humans. To achieve optimal plant growth, the system must be continuously monitored and managed, for example, temperature, moisture, soil humidity, light intensity, and so on (Environmental and Pollution Science 2019). Many farmers fail to profit from greenhouse crops because they are unable to control two critical elements that influence plant development and output. The temperature of the greenhouse should be maintained at a specified level. Crop transpiration and condensation of water vapor on various greenhouse surfaces can all be caused by high humidity. This greenhouse monitoring and management system comes to the rescue in the face of such obstacles. The design and implementation of several sensors for greenhouse environment monitoring, such as humidity, temperature, light condition, and soil moisture, are discussed here.

The authors have implemented greenhouse monitoring systems using IoT. The designed system monitors parameters such as temperature, humidity, water level, and light. The pin-out details for interfacing Pyboard with sensors are given in Table 4.5. The temperature and humidity sensors detect temperature and humidity, the soil moisture sensor detects water level, and the LDR sensor detects light. The detailed interfacing diagram is shown in Fig. 4.16.

Configuration of ThingSpeak Channel

After creating the channel as shown in earlier Sect. 4.2, click on 'New Channel' button which will subsequently ask for the 'Name and Description' of the data you want to upload on this channel as shown in Fig. 4.17.

In this application, the authors have named the channel 'Greenhouse Monitoring' and four fields are created as shown in Fig. 4.17a. After this, click on 'save channel' button to save the details. Figure 4.17b shows the flowchart for greenhouse monitoring system.

Table 4.5 Pin connection of Pyboard with sensors to monitor Greenhouse

Pyboard	OLED	DHT22	LDR	Moisture
Y9	SCL	–	–	–
Y10	SDA	–	–	–
X1	–	Data	–	–
X2	–	–	Output of circuit	–
X3	–	–	–	Aout

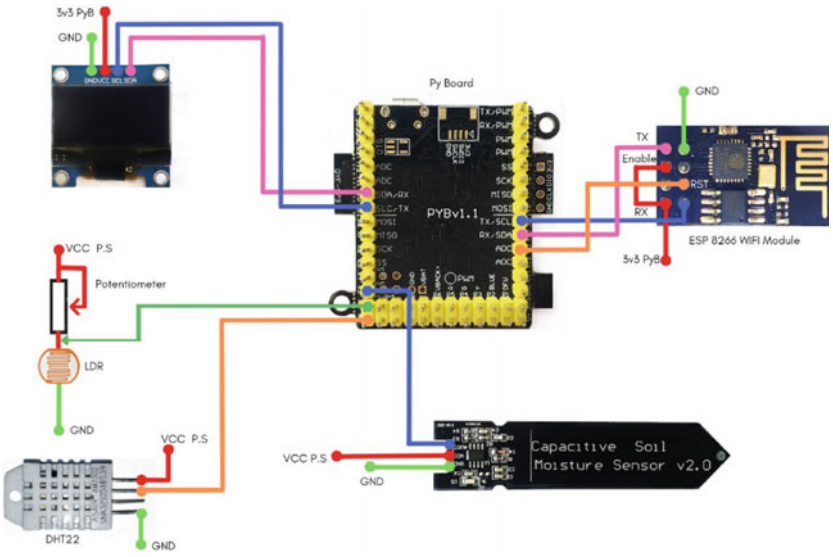


Fig. 4.16 Circuit diagram of Greenhouse monitoring

(a)

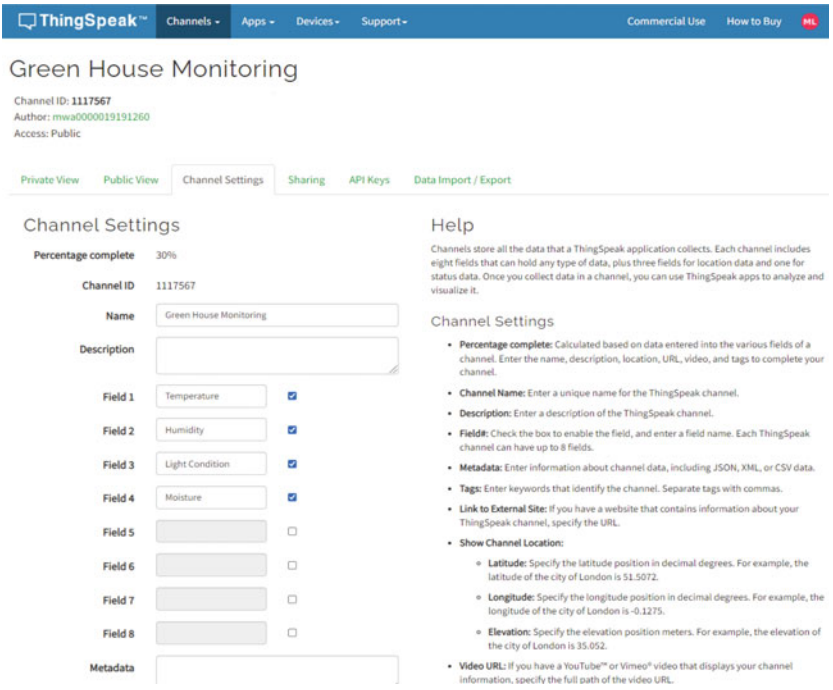


Fig. 4.17 **a** Channel settings and **b** flowchart for greenhouse monitoring system

(b)

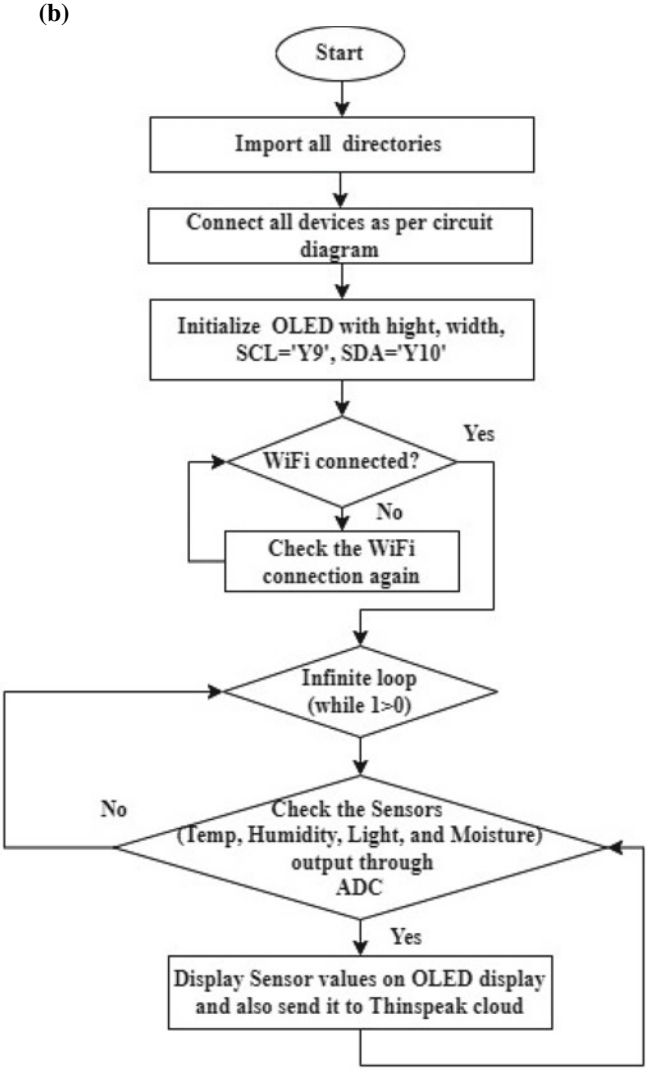


Fig. 4.17 (continued)

Obtain the API Key

To send data to ThingSpeak, a unique API key is required, which is used in the main python code to upload parameters onto the ThingSpeak server. Navigate to 'API Keys' header under the newly created above channel to get the unique API key. After completing these steps, the channel is ready to receive the parameters. Enter the below code in '*main.py*' and save it on Pyboard.

#Complete code (main.py)

```
from pyb import Pin
import pyb
import ssd1306
import machine
from machine import Pin
from micropython import const
import dht
width = const (128)
height= const (64)
ssd1306_scl= Pin('Y9', Pin.OUT_PP)
ssd1306_sda= Pin('Y10', Pin.OUT_PP)
i2c_ssd1306=machine.I2C(scl=ssd1306_scl, sda=ssd1306_sda)
oled = ssd1306.SSD1306_I2C(width, height, i2c_ssd1306)

import pywifi
temp_hum = dht.DHT22(Pin('X1'))
while 1:
    rst_pyb = Pin('X11', Pin.OUT)
    rst_pyb.low()
    pyb.delay(20)
    rst_pyb.high()
    pyb.delay(500)
    Pyboard_wifi = pywifi.ESP8266(1, 115200)
    wifi_mode = 3
    Pyboard_wifi.set_mode(wifi_mode)
    pyb.delay(50)
    Pyboard_wifi.connect(ssid='AndroidAP93E9', psk='uxbm0411')
    pyb.delay(50)
    pyb.LED(4).on() #BLUE LED ON
    pyb.delay(2000)
    temp_hum.measure()
    temp = temp_hum.temperature()
    hum = temp_hum.humidity()
    LDR = pyb.ADC('X2')
    Out1 = LDR.read()  # read an analog value
    Moisture = pyb.ADC('X3')  # create an analog object from a pin
    Out2 = Moisture.read()  # read an analog value
    mois = ((Out2 - 3700) * (100 - 1) / (1300- 3700) +1)

    oled.text("Tem: " + str(temp), 0, 10)
    oled.text("Humidity:" + str(hum), 0, 20)
    oled.text("Light:" + str(Out1), 0, 30)
    oled.text("Moist-sense:" + str(Out2), 0, 40)
    oled.text("Moisture %:" + str(mois), 0, 50)
    oled.show()
    oled.fill(0)
    pyb.delay(1000)
    Pyboard_wifi.start_connection(protocol='TCP',dest_ip='184.106.153.149',
    dest_port=80,debug=True)
```

```
Pyboard_wifi.send('GET
https://api.thingspeak.com/update?api_key=E3SLU3MT69BN8G3V&fiel
d1=' + str(temp)+'&field2='+ str(hum) +'&field3='+ str(Out1)+  '&field4='+
str(mois)+' HTTP/1.0\r\nHost: 192.168.43.176\r\n\r\n', debug=True)

pyb.delay(1000)
```

After entering the above code, one has to execute the '*main.py*' program and the greenhouse monitoring parameters will be displayed on OLED display as well as on ThingSpeak cloud shown in Fig. 4.18.

Fig. 4.18 Greenhouse monitoring and control system on ThingSpeak

4.5 Aquaculture Monitoring

Aquaculture is a fast-expanding food production method that has successfully increased fish and shellfish production which helps to feed the world's growing population. Aquaculture is the primary means of survival, being the main source of income for a large strata of society. However, this kind of food production has numerous problems, including rising costs, stricter government regulations, and restricted water supplies. These difficulties have necessitated the development of more complex monitoring and feeding equipment in order to provide tightly controlled and long-term growth conditions. Aquaculture, often known as aqua farming, is the breeding, rearing, and harvesting of fish, seaweed, algae, and a variety of other creatures. It is also characterized as a breeding species that develops in a controlled aquatic habitat. Aquaculture is one of the most dependable and low-impact processes for providing high-quality protein for humans.

The authors have implemented aquaculture monitoring systems using IoT. The designed system monitors parameters such as turbidity, water temperature, pH, and Total Dissolved Solid (TDS).

The pin-out details for interfacing Pyboard with sensors for aquaculture monitoring system is given in Table 4.6. The detailed interfacing diagram is shown in Fig. 4.19. The Grove—PH sensor detects the pH of the water, turbidity sensor will give the turbidity of the water, TDS sensor will give the TDS of the water, and temperature sensor (DS18B20) detects the water temperature.

Configuration of ThingSpeak Channel

After creating the channel as shown in earlier Sect. 4.2, click on 'New Channel' button which will subsequently ask for the 'Name and Description' of the data you want to upload on this channel as shown in Fig. 4.20.

In this application, the authors have named the channel as 'Aquaculture monitoring' and four fields are created as shown in Fig. 4.20a. After this, click on the save channel button to save the details. Figure 4.20b shows the flowchart for aquaculture monitoring system.

Table 4.6 Pin connection of Pyboard with sensors for aquaculture monitoring

Pyboard	OLED	Turbidity	pH	TDS	DS18B20
Y9	SCL	–	–	–	–
Y10	SDA	–	–	–	–
Y11	–	Output of circuit	–	–	–
Y12	–	–	SIG	–	–
X7	–	–	–	Aout	–
X8	–	–	–	–	Sensor output

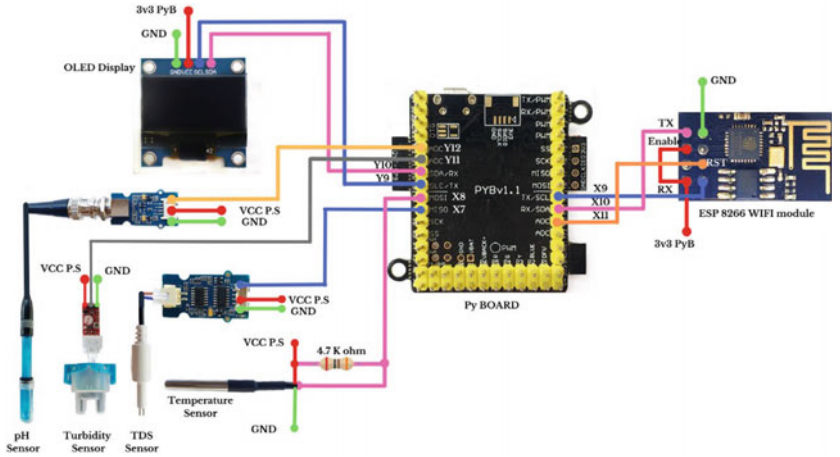


Fig. 4.19 Circuit diagram of aquaculture monitoring

(a)

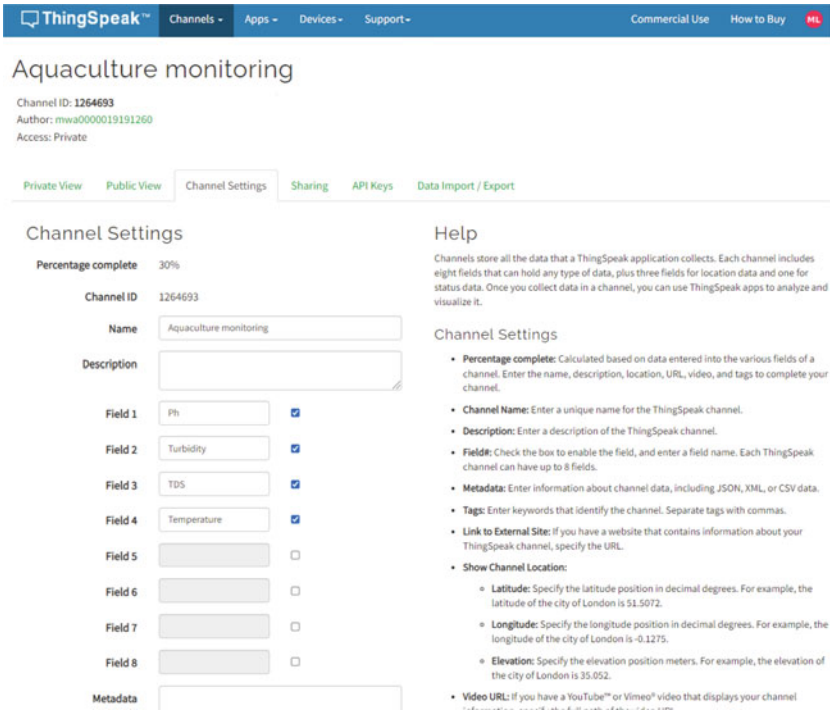


Fig. 4.20 **a** Channel settings. **b** Flowchart for aquaculture monitoring system

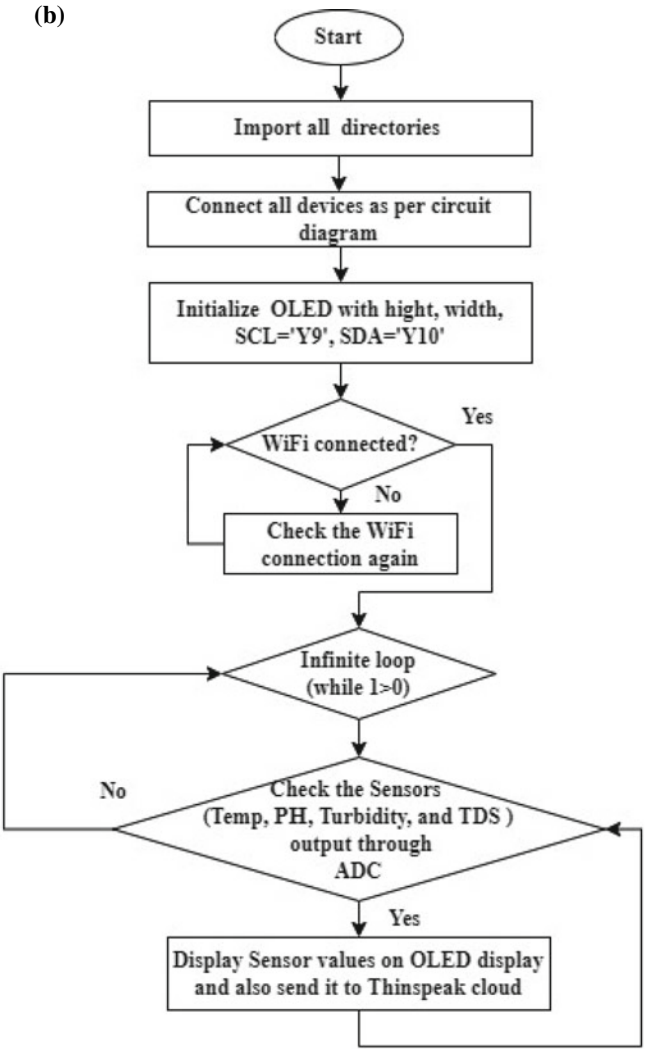


Fig. 4.20 (continued)

Obtain the API Key

To send data to ThingSpeak, a unique API key is required, which is used in the main python code to upload parameters onto the ThingSpeak server. Navigate to 'API Keys' header under the newly created above channel to get the unique API key. After completing these steps, the channel is ready to receive the parameters. Enter the below code in '*main.py*' and save it on Pyboard.

```
#Complete code (main.py)
import time
import ssd1306
import machine
from pyb import Pin
from micropython import const
import pyb
import onewire
import ds18x20

width = const(128)
height = const(64)
ssd1306_scl = Pin('Y9', Pin.OUT_PP)
ssd1306_sda = Pin('Y10', Pin.OUT_PP)
i2c_ssd1306 = machine.I2C(scl=ssd1306_scl, sda=ssd1306_sda)
oled = ssd1306.SSD1306_I2C(width, height, i2c_ssd1306)
import pywifi
while True:
    rst_pyb = Pin('X11', Pin.OUT)
    rst_pyb.low()
    pyb.delay(20)
    rst_pyb.high()
    pyb.delay(500)
    Pyboard_wifi = pywifi.ESP8266(1, 115200)
    wifi_mode = 3
    Pyboard_wifi.set_mode(wifi_mode)
    pyb.delay(50)
    Pyboard_wifi.connect(ssid='AndroidAP93E9', psk='uxbm0411')
    pyb.delay(50)
    pyb.LED(4).on()  # BLUE LED ON
    pyb.delay(2000)
    oled.fill(0)
    adc = pyb.ADC('Y12')   # Ph sensor
    adc1 = pyb.ADC('Y11')   #Turbidity sensor
    val1 = adc1.read()
    adc2 = pyb.ADC('X8')  # TDS sensor
    val2 = adc2.read()
    ds_pin = Pin('X7')     # temperaute sensor
    sensorSum = 0
    for i in (0, 50):
        sensorValue = adc.read()
        sensorSum = sensorSum + sensorValue
    sensorValue = sensorSum / 50
    ph = 7 - 1000 * (sensorValue - 372) * 3.3/59.16/4095
    scaled_value = (val1 - 60) * (0.2 - 5) / (2200 - 60) + 5
    V = val2 * (3.3 / 4095.0)
    tdsValue = (133.42 / V * V * V - 255.86 * V * V + 857.39 * V) * 0.5
    ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
    roms = ds_sensor.scan()
    ds_sensor.convert_temp()
    time.sleep(1)
```

```
for rom in roms:
    oled.text("Ph:" + str(ph), 0, 0)
    oled.text("Turbidity:" + str(scaled_value), 0, 10)
    oled.text("TDS:" + str(tdsValue), 0, 20)
    a=ds_sensor.read_temp(rom)
    oled.text("Temp " + str(a), 0, 30)
    oled.show()
    time.sleep(1)

oled.show()

Pyboard_wifi.start_connection(protocol='TCP',dest_ip='184.106.153.149',
dest_port=80,debug=True)

Pyboard_wifi.send('GET
https://api.thingspeak.com/update?api_key=UBY89DB50V2UPGGU&field1=' + str(ph) +
'&field2=' + str(scaled_value) + '&field3=' + str(tdsValue) + '&field4=' + str(a)
+ ' HTTP/1.0\r\nHost: 192.168.43.176\r\n\r\n', debug=True)

pyb.delay(1000)
```

After entering the above code, one has to execute the '*main.py*' program and the greenhouse monitoring parameters will be displayed on OLED display as well as on ThingSpeak cloud shown in Fig. 4.21.

Conclusion:

PyCharm supports various boards including Pyboard. To start with, MicroPython installation and testing steps of Pyboard in PyCharm IDE are given in detail. The

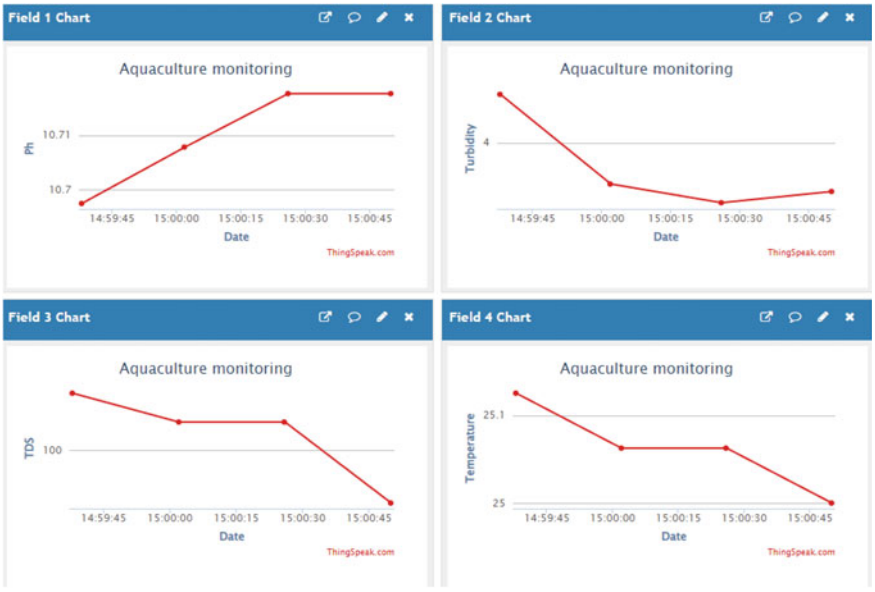


Fig. 4.21 Aquaculture monitoring system on ThingSpeak

various applications, such as, home automation, smart e-waste bin, industrial environmental monitoring, greenhouse monitoring and aquaculture monitoring are implemented to track and control remotely. There is no WiFi support on Pyboards; hence, authors have interfaced external WiFi module. Home automation system is first implemented with switches and appliances interfaced directly to Pyboard. The same application is implemented with ESP32 and controlled remotely with Blynk app. For other applications, the authors have provided detailed interfacing and configuration steps to monitor/control remotely through ThingSpeak cloud.

Exercise

(1) Blink eight LEDs using Pyboard
(2) Design a system to control six appliances with switches connected to ESP32. Also, display the status of each appliance on the OLED screen.
(3) Interface a temperature sensor to ESP32. Using a relay, build a system to turn ON/OFF the fan when temperature rises/decreases of a particular threshold temperature and monitor the temperature.
(4) Assume there are two rooms, Room1 and Room2. Each room has a temperature sensor and a fan respectively. Design a system that can monitor the temperature of each room separately and turn ON/OFF the fan in the respective room if temperature crosses the threshold.
(5) Using turbidity sensor along with ESP32, build a system to monitor cleanliness of water. If the value of turbidity is higher than a set threshold, give out an alarm signaling the need for water replacement/cleaning. The system can then be applied to monitor cleanliness of water in aquariums.
(6) Interface five IR sensors to get the accurate status of the e-waste bin by placing appropriately and displaying the status on ThingSpeak Cloud.
(7) Interface other industrial sensors to monitor Environmental parameters.

References

https://doi.org/10.1016/j.jksuci.2020.08.016.
https://doi.org/10.1016/B978-0-12-814719-1.00010-0.
https://doi.org/10.1016/B978-0-12-386454-3.01041-1.

Artiola JF, Brusseau ML (2019) The role of environmental monitoring in pollution science. In: Brusseau ML, Pepper IL, Gerba CP (eds) Environmental and pollution science, 3rd edn. Academic Press, pp 149–162. https://doi.org/10.1016/B978-0-12-814719-1.00010-0

Covaci A (2014) Environmental fate and behavior. In: Wexler P (ed) Encyclopedia of toxicology, 3rd edn. Academic Press, pp 372–374. https://doi.org/10.1016/B978-0-12-386454-3.01041-1

Environmental and Pollution Science (Third Edition) (2019) Academic Press, pp 149–162

Madhesh A, Kowsigan M, Khishore R, Balasubramanie P (2020) IoT based home automation and security system. Int J Adv Sci Technol 29(9s):2733–2739. http://sersc.org/journals/index.php/IJAST/article/view/15439

Majeed R, Abdullah NA, Ashraf I, Zikria YB, Mushtaq MF, Umer M (2020) An intelligent, secure, and smart home automation system. Sci Program. ID 4579291, 14 p. https://doi.org/10.1155/2020/4579291

Nasir OA, Mumtazah S (2020) IOT-based monitoring of aquaculture system. Matter: Int J Sci Technol 6(1):113–137. https://doi.org/10.20319/mijst.2020.61.113137

Rahman MW, Islam R, Hasan A, Bithi NI, Hasan MM, Rahman MM (2020) Intelligent waste management system using deep learning with IoT. J King Saud Univ-Comput Inf Sci. https://doi.org/10.1016/j.jksuci.2020.08.016

Stolojescu-Crisan C, Crisan C, Butunoi B-P (2021) An IoT-based smart home automation system. Sensors 21:3784. https://doi.org/10.3390/s21113784

Sujin JS et al (2021) IOT based greenhouse monitoring and controlling system. J Phys: Conf Ser 1916 012062

Zeb A, Ali Q, Saleem MQ, Awan KM, Alowayr AS, Uddin J, Iqbal S, Bashir F (2019) A proposed IoT-enabled smart waste bin management system and efficient route selection. J Comput Netw Commun. ID 7043674, 9 p. https://doi.org/10.1155/2019/7043674